
WESTPA Documentation

Release 2022.08

Matthew C. Zwier and Lillian T. Chong

Apr 16, 2024

FOR USERS (V2022.XX):

1	Overview	3
2	Requirements	5
3	Obtaining and Installing WESTPA	7
4	Getting started	9
5	Getting help	11
6	Developers	13
6.1	westpa.cli package	13
6.1.1	w_init	13
6.1.1.1	Overview	13
6.1.1.2	Command-Line Options	14
6.1.1.2.1	State Options	14
6.1.1.3	Examples	15
6.1.1.4	westpa.cli.core.w_init module	15
6.1.2	w_bins	16
6.1.2.1	Overview	16
6.1.2.2	Command-Line Options	17
6.1.2.2.1	Options Under ‘info’	17
6.1.2.2.2	Options Under ‘rebin’	18
6.1.2.3	Input Options	18
6.1.2.4	Examples	19
6.1.2.5	westpa.cli.tools.w_bins module	19
6.1.3	w_run	21
6.1.3.1	Overview	21
6.1.3.2	Command-Line Options	22
6.1.3.2.1	Segment Options	22
6.1.3.3	Example	22
6.1.3.4	westpa.cli.core.w_run module	22
6.1.4	w_truncate	22
6.1.4.1	Overview	22
6.1.4.2	Command-Line Options	22
6.1.4.2.1	Iteration Options	23
6.1.4.3	Examples	23
6.1.4.4	westpa.cli.core.w_truncate module	23
6.1.5	w_fork	23
6.1.5.1	westpa.cli.tools.w_fork module	24
6.1.6	w_assign	26

6.1.6.1	Overview	26
6.1.6.2	Command-Line Options	27
6.1.6.3	Input/output Options	27
6.1.6.4	Binning Options	27
6.1.6.5	Macrostate Options	27
6.1.6.6	Specifying Progress Coordinate	28
6.1.6.7	Examples	28
6.1.6.8	westpa.cli.tools.w_assign module	28
6.1.7	w_trace	35
6.1.7.1	positional arguments	36
6.1.7.2	optional arguments	36
6.1.7.3	general options	37
6.1.7.4	WEST input data options	37
6.1.7.5	output options	37
6.1.7.6	westpa.cli.tools.w_trace module	37
6.1.8	w_ipa	43
6.1.8.1	westpa.cli.tools.w_ipa module	45
6.1.9	w_pdist	50
6.1.9.1	Overview	50
6.1.9.2	Command-Line Options	50
6.1.9.2.1	Input/output options	50
6.1.9.2.2	Iteration range options	51
6.1.9.2.3	Probability distribution binning options	51
6.1.9.3	Examples	51
6.1.9.4	westpa.cli.tools.w_pdist module	52
6.1.10	w_succ	56
6.1.10.1	westpa.cli.core.w_succ module	57
6.1.11	w_crawl	60
6.1.11.1	Command-line options	60
6.1.11.2	westpa.cli.tools.w_crawl module	63
6.1.12	w_direct	66
6.1.12.1	westpa.cli.tools.w_direct module	68
6.1.13	w_select	75
6.1.13.1	Predicate function	75
6.1.13.2	Output format	75
6.1.13.3	Command-line arguments	76
6.1.13.4	westpa.cli.tools.w_select module	78
6.1.14	w_states	81
6.1.14.1	westpa.cli.core.w_states module	84
6.1.15	w_eddist	87
6.1.15.1	Source data	87
6.1.15.2	Histogram binning	87
6.1.15.3	Output format	88
6.1.15.4	Subsequent processing	88
6.1.15.5	Parallelization	88
6.1.15.6	Command-line options	88
6.1.15.7	westpa.cli.tools.w_eddist module	91
6.1.16	w_ntop	94
6.1.16.1	Output format	95
6.1.16.2	Command-line arguments	95
6.1.16.3	westpa.cli.tools.w_ntop module	96
6.1.17	w_multi_west	100
6.1.17.1	Overview	100
6.1.17.2	Command-Line Options	100

6.1.17.2.1	Input/output options	101
6.1.17.3	Examples	101
6.1.17.4	westpa.cli.tools.w_multi_west module	102
6.1.18	w_red	104
6.1.18.1	westpa.cli.tools.w_red module	105
6.1.19	plothist	107
6.1.19.1	Overview	107
6.1.19.1.1	‘instant’ mode	107
6.1.19.1.2	‘average’ mode	107
6.1.19.1.3	‘evolution’ mode	108
6.1.19.2	Command-Line Options	108
6.1.19.2.1	Input/output options	108
6.1.19.2.2	Plotting options	109
6.1.19.2.3	Iteration selection options	109
6.1.19.2.4	Specifying progress coordinate dimension	110
6.1.19.3	Examples	110
6.1.19.3.1	Basic plotting	110
6.1.19.3.2	Specifying progress coordinate	110
6.1.19.4	westpa.cli.tools.plothist module	111
6.1.20	ploterr	116
6.1.20.1	Command-line options	116
6.1.20.2	westpa.cli.tools.ploterr module	116
6.1.21	westpa.cli package	120
6.1.21.1	w_kinavg	120
6.1.21.1.1	Output format	120
6.1.21.1.2	Command-line options	122
6.1.21.1.3	westpa.cli.tools.w_kinavg module	123
6.1.21.2	w_kinetics	128
6.1.21.2.1	Output format	128
6.1.21.2.2	Command-line options	129
6.1.21.2.3	westpa.cli.tools.w_kinetics module	129
6.1.21.3	w_stateprobs	133
6.1.21.3.1	Output format	133
6.1.21.3.2	Command-line options	134
6.1.21.3.3	westpa.cli.tools.w_stateprobs module	136
6.1.21.4	w_dumpsegs	140
6.1.21.4.1	westpa.cli.tools.w_dumpsegs module	140
6.1.21.5	w_postanalysis_matrix	142
6.1.21.5.1	westpa.cli.tools.w_postanalysis_matrix module	142
6.1.21.6	w_postanalysis_reweight	145
6.1.21.6.1	westpa.cli.tools.w_postanalysis_reweight module	145
6.1.21.7	w_reweight	147
6.1.21.7.1	westpa.cli.tools.w_reweight module	147
6.1.21.8	w_fluxanl	156
6.1.21.8.1	Overview	156
6.1.21.8.2	Command-Line Options	156
6.1.21.8.3	Examples	157
6.1.21.8.4	westpa.cli.tools.w_fluxanl module	157
6.2	westpa.core package	162
6.2.1	westpa.core.binning package	162
6.2.1.1	westpa.core.binning module	162
6.2.1.2	westpa.core.binning.assign module	165
6.2.1.3	westpa.core.binning.bins module	167
6.2.2	Minimal Adaptive Binning (MAB) Scheme	168

6.2.2.1	westpa.core.binning.mab module	168
6.2.2.2	westpa.core.binning.mab_driver	168
6.2.2.3	westpa.core.binning.mab_manager	170
6.2.3	westpa.core.kinetics package	174
6.2.3.1	westpa.core.kinetics module	174
6.2.3.2	westpa.core.kinetics.events module	175
6.2.3.3	westpa.core.kinetics.matrates module	175
6.2.3.4	westpa.core.kinetics.rate_averaging module	177
6.2.4	westpa.core.propagators package	179
6.2.4.1	westpa.core.propagators module	179
6.2.4.2	westpa.core.propagators.executable module	180
6.2.5	westpa.core.reweight package	187
6.2.5.1	westpa.core.reweight module	187
6.2.5.2	westpa.core.reweight.matrix module	187
6.2.6	westpa.core modules	188
6.2.6.1	westpa.core module	188
6.2.6.2	westpa.core.data_manager module	188
6.2.6.3	westpa.core.extloader module	198
6.2.6.4	westpa.core.h5io module	198
6.2.6.5	westpa.core.progress module	222
6.2.6.6	westpa.core.segment module	224
6.2.6.7	westpa.core.sim_manager module	225
6.2.6.8	westpa.core.states module	230
6.2.6.9	westpa.core.systems module	233
6.2.6.10	westpa.core.textio module	234
6.2.6.11	westpa.core.we_driver module	234
6.2.6.12	westpa.core.wm_ops module	238
6.2.6.13	westpa.core.yamlcfg module	239
6.3	westpa.work_managers package	241
6.3.1	westpa.work_managers package	241
6.3.1.1	westpa.work_managers module	241
6.3.1.2	westpa.work_managers.core module	242
6.3.1.3	westpa.work_managers.environment module	245
6.3.1.4	westpa.work_managers.mpi module	246
6.3.1.5	westpa.work_managers.processes module	249
6.3.1.6	westpa.work_managers.serial module	252
6.3.1.7	westpa.work_managers.threads module	254
6.3.2	westpa.work_managers.zeromq package	256
6.3.2.1	westpa.work_managers.zeromq module	256
6.3.2.2	westpa.work_managers.zeromq.core module	259
6.3.2.3	westpa.work_managers.zeromq.node module	263
6.3.2.4	westpa.work_managers.zeromq.work_manager module	265
6.3.2.5	westpa.work_managers.zeromq.worker module	272
6.4	westpa.tools package	276
6.4.1	westpa.tools module	276
6.4.2	westpa.tools.binning module	283
6.4.3	westpa.tools.core module	285
6.4.4	westpa.tools.data_reader module	288
6.4.5	westpa.tools.dtypes module	290
6.4.6	westpa.tools.iter_range module	290
6.4.7	westpa.tools.kinetics_tool module	292
6.4.8	westpa.tools.plot module	294
6.4.9	westpa.tools.progress module	294
6.4.10	westpa.tools.selected_segs module	295

6.4.11	westpa.tools.wipi module	297
6.5	Other Packages	298
6.5.1	westpa.fasthist package	298
6.5.1.1	Module contents	298
6.5.2	westpa.mclib package	298
6.5.2.1	Module contents	298
6.5.3	westpa.trajtree package	299
6.5.3.1	westpa.trajtree module	299
6.5.3.2	westpa.trajtree.trajtree module	300
6.5.4	WESTPA Old Tools	300
6.5.4.1	westpa.oldtools package	300
6.5.4.1.1	westpa.oldtools module	300
6.5.4.1.2	westpa.oldtools.files module	300
6.5.4.1.3	westpa.oldtools.miscfn module	301
6.5.4.2	westpa.oldtools.iframe package	301
6.5.4.2.1	westpa.oldtools.iframe	301
6.5.4.2.2	westpa.oldtools.iframe.atool module	307
6.5.4.2.3	westpa.oldtools.iframe.base_mixin module	307
6.5.4.2.4	westpa.oldtools.iframe.binning module	308
6.5.4.2.5	westpa.oldtools.iframe.data_reader module	308
6.5.4.2.6	westpa.oldtools.iframe.iter_range module	312
6.5.4.2.7	westpa.oldtools.iframe.kinetics module	313
6.5.4.2.8	westpa.oldtools.iframe.mcbs module	313
6.5.4.2.9	westpa.oldtools.iframe.output module	314
6.5.4.2.10	westpa.oldtools.iframe.plotting module	315
6.5.4.2.11	westpa.oldtools.iframe.trajwalker module	315
6.5.4.2.12	westpa.oldtools.iframe.transitions module	315
6.5.4.3	westpa.oldtools.cmds package	317
6.5.4.3.1	westpa.oldtools.cmds module	317
6.5.4.3.2	westpa.oldtools.cmds.w_ttimes module	317
6.5.4.4	westpa.oldtools.stats package	317
6.5.4.4.1	westpa.oldtools.stats module	317
6.5.4.4.2	westpa.oldtools.stats.accumulator module	317
6.5.4.4.3	westpa.oldtools.stats.edfs module	318
6.5.4.4.4	westpa.oldtools.stats.mcbs module	319
6.6	westpa.westext package	319
6.6.1	Currently Supported	319
6.6.1.1	westpa.westext.adaptvoronoi package	319
6.6.1.1.1	Submodules	319
6.6.1.1.2	westpa.westext.adaptvoronoi.adaptVor_driver module	319
6.6.1.1.3	Module contents	320
6.6.1.2	westpa.westext.stringmethod package	321
6.6.1.2.1	Submodules	321
6.6.1.2.2	westpa.westext.stringmethod.fourier_fitting module	321
6.6.1.2.3	westpa.westext.stringmethod.string_driver module	321
6.6.1.2.4	westpa.westext.stringmethod.string_method module	321
6.6.1.2.5	Module contents	321
6.6.1.3	westpa.westext.hamsm_restarting package	321
6.6.1.3.1	Description	321
6.6.1.3.2	Usage	322
6.6.1.3.3	Extensions	323
6.6.2	Deprecated	326
6.6.2.1	westpa.westext.weed package	326
6.6.2.1.1	Submodules	326

	6.6.2.1.2	westpa.westext.weed.BinCluster module	326
	6.6.2.1.3	westpa.westext.weed.ProbAdjustEquil module	326
	6.6.2.1.4	westpa.westext.weed.UncertMath module	326
	6.6.2.1.5	westpa.westext.weed.weed_driver module	327
	6.6.2.1.6	Module contents	327
	6.6.2.2	westpa.westext.wess package	328
	6.6.2.2.1	Submodules	328
	6.6.2.2.2	westpa.westext.wess.ProbAdjust module	328
	6.6.2.2.3	westpa.westext.wess.wess_driver module	328
	6.6.2.2.4	Module contents	329
	6.6.3	Module contents	329
6.7		westpa.analysis package	329
	6.7.1	How To	329
	6.7.2	Retrieving Trajectories	331
	6.7.2.1	Built-in Reader	331
	6.7.2.2	Custom Readers	331
	6.7.3	westpa.analysis.core module	332
	6.7.4	westpa.analysis.trajectories module	338
	6.7.5	westpa.analysis.statistics module	340
6.8		HDF5 File Schema	340
	6.8.1	Overall structure	340
	6.8.2	The root group (/)	341
	6.8.2.1	The iteration summary table (/summary)	341
	6.8.3	Per iteration data (/iterations/iter_XXXXXXXX)	341
	6.8.3.1	The segment summary table (/iterations/iter_XXXXXXXX/seg_index)	342
	6.8.4	Bin Topologies group (/bin_topologies)	342
6.9		Overview	342
6.10		Style Guide	342
	6.10.1	Preface	342
	6.10.2	Style and Usage	343
	6.10.2.1	Acronyms and abbreviations	343
	6.10.2.2	Capitalization	343
	6.10.2.3	Contractions	344
	6.10.2.4	Internationalization	344
	6.10.2.5	Italics	344
	6.10.2.6	Non-English words	344
	6.10.2.7	Specially formatted characters	344
	6.10.2.8	Subject	345
	6.10.2.9	Tense	345
	6.10.2.10	Voice	345
	6.10.2.11	Weighted ensemble	345
	6.10.2.12	WESTPA	345
	6.10.3	Computer Language Elements	346
	6.10.3.1	Classes, modules, and libraries	346
	6.10.3.2	Methods and commands	346
	6.10.3.3	Programming languages	346
	6.10.3.4	Scripts	347
	6.10.3.5	Variables	348
6.11		Source Code Management	348
6.12		Documentation Practices	348
	6.12.1	Introduction to Editing the Sphinx Documentation	348
	6.12.2	Uploading to ReadTheDocs	348
	6.12.3	In Cases of Major Revisions in Code Base	348
6.13		WESTPA Modules API	349

6.13.1	Binning	349
6.13.2	YAMLCFG	349
6.13.3	RC	349
6.14	WESTPA Tools	350
6.15	WEST	350
6.15.1	Setup	350
6.15.1.1	Defining and Calculating Progress Coordinates	350
6.15.1.2	Binning	350
6.15.1.2.1	RectilinearBinMapper	350
6.15.1.2.2	VoronoiBinMapper	351
6.15.1.2.3	FuncBinMapper	351
6.15.1.2.4	VectorizingFuncBinMapper	352
6.15.1.2.5	PiecewiseBinMapper	352
6.15.1.2.6	RecursiveBinMapper	352
6.15.1.3	Initial/Basis States	353
6.15.1.4	Target States	354
6.15.1.5	Propagators	355
6.15.1.5.1	The Executable Propagator	355
6.15.1.5.2	Writing custom propagators	355
6.15.1.6	Configuration File	355
6.15.1.7	Environmental Variables	357
6.15.1.7.1	Programs executed for an iteration	358
6.15.1.7.2	Programs executed for a segment	358
6.15.1.7.3	Programs executed for a single point	359
6.15.1.8	Plugins	359
6.15.1.9	Weighted Ensemble Algorithm (Resampling)	359
6.15.2	Running	359
6.15.2.1	Overview	359
6.15.2.2	Setting simulation limits	360
6.15.2.3	Running a simulation	360
6.15.2.3.1	Running on a single node	360
6.15.2.3.2	Running on multiple nodes with MPI	360
6.15.2.3.3	Running on multiple nodes with ZeroMQ	360
6.15.2.4	Managing data	360
6.15.2.5	Recovering from errors	360
6.15.3	Analysis	360
6.15.3.1	Gauging simulation progress and convergence	360
6.15.3.1.1	Progress coordinate distribution (w_pcpdist)	360
6.15.3.1.2	Kinetics for source/sink simulations	360
6.15.3.1.3	Kinetics for arbitrary state definitions	361
6.16	WEST Tools	362
6.16.1	Overview	363
6.16.1.1	Tools for setting up and running a simulation	363
6.16.1.2	Tools for analyzing simulation results	363
6.16.2	General Command Line Options	364
6.16.2.1	A note on specifying a configuration file	364
6.16.3	Work Manager Options	364
6.16.3.1	Overview	364
6.16.3.2	General work manager options	365
6.16.3.3	ZeroMQ ('zmq') work manager	365
6.16.4	Initializing/Running Simulations	366
6.17	WEST Work Manager	366
6.17.1	Introduction	366
6.17.2	Environment variables	367

6.17.2.1	For controlling task distribution	367
6.17.2.2	For passing information to workers	367
6.17.3	The ZeroMQ work manager for clusters	368
6.18	WEST Extensions	368
6.18.1	Post-Analysis Reweighting	368
6.18.2	String Method	368
6.18.3	Weighted Ensemble Equilibrium Dynamics	368
6.18.4	Weighted Ensemble Steady State	368
6.19	Command Line Tool Index	368
6.19.1	w_init	368
6.19.2	w_bins	371
6.19.2.1	Overview	371
6.19.2.2	Command-Line Options	371
6.19.2.2.1	Options Under ‘info’	371
6.19.2.2.2	Options Under ‘rebin’	372
6.19.2.3	Input Options	373
6.19.2.4	Examples	373
6.19.3	w_run	373
6.19.4	w_truncate	375
6.19.5	w_fork	376
6.19.6	w_assign	377
6.19.6.1	Source data	377
6.19.6.2	Specifying macrostates	378
6.19.6.3	Output format	378
6.19.6.4	Parallelization	379
6.19.6.5	Command-line options	379
6.19.7	w_trace	383
6.19.7.1	positional arguments	384
6.19.7.2	optional arguments	384
6.19.7.3	general options	384
6.19.7.4	WEST input data options	384
6.19.7.5	output options	384
6.19.8	w_fluxanl	385
6.19.8.1	Overview	385
6.19.8.2	Command-Line Options	385
6.19.8.2.1	Input/output options	385
6.19.8.2.2	Iteration range options	386
6.19.8.2.3	Confidence interval and bootstrapping options	386
6.19.8.3	Examples	386
6.19.9	w_ipa	387
6.19.10	w_pdist	389
6.19.10.1	Source data	390
6.19.10.2	Histogram binning	390
6.19.10.3	Output format	391
6.19.10.4	Subsequent processing	391
6.19.10.5	Parallelization	391
6.19.10.6	Command-line options	391
6.19.11	w_succ	394
6.19.12	w_crawl	395
6.19.12.1	Command-line options	395
6.19.13	w_direct	398
6.19.14	w_select	400
6.19.14.1	Predicate function	401
6.19.14.2	Output format	401

6.19.14.3	Command-line arguments	401
6.19.15	w_states	404
6.19.16	w_eddist	407
6.19.16.1	Source data	408
6.19.16.2	Histogram binning	408
6.19.16.3	Output format	408
6.19.16.4	Subsequent processing	409
6.19.16.5	Parallelization	409
6.19.16.6	Command-line options	409
6.19.17	w_ntop	412
6.19.17.1	Output format	412
6.19.17.2	Command-line arguments	412
6.19.18	plohist	413
6.19.18.1	plohist_instant	413
6.19.18.2	plohist_average	415
6.19.18.3	plohist_evolution	416
6.19.18.4	Command-line options	418
6.19.19	ploterr	419
6.19.19.1	Command-line options	419
6.19.20	w_kinavg	420
6.19.20.1	Output format	420
6.19.20.2	Command-line options	421
6.19.21	w_kinetics	423
6.19.21.1	Output format	423
6.19.21.2	Command-line options	424
6.19.22	w_stateprobs	425
6.19.22.1	Output format	425
6.19.22.2	Command-line options	426
6.20	HDF5 File Schema	428
6.20.1	Overall structure	428
6.20.2	The root group (/)	429
6.20.2.1	The iteration summary table (/summary)	429
6.20.3	Per iteration data (/iterations/iter_XXXXXXXX)	429
6.20.3.1	The segment summary table (/iterations/iter_XXXXXXXX/seg_index)	430
6.20.4	Bin Topologies group (/bin_topologies)	430
6.21	Checklist	430
6.21.1	Configuring a WESTPA Simulation	430
6.21.2	Running a WESTPA simulation	431
6.21.3	Analyzing a WESTPA simulation	432
6.22	Frequently Asked Questions (FAQ)	432
6.22.1	Simulation	432
6.22.2	GROMACS	433
6.22.3	Development	434

Python Module Index	435
----------------------------	------------

Index	437
--------------	------------

OVERVIEW

WESTPA is a package for constructing and running stochastic simulations using the “weighted ensemble” approach of Huber and Kim (1996). For use of WESTPA please cite the following:

Zwier, M.C., Adelman, J.L., Kaus, J.W., Pratt, A.J., Wong, K.F., Rego, N.B., Suarez, E., Lettieri, S., Wang, D.W., Grabe, M., Zuckerman, D.M., and Chong, L.T. “WESTPA: An Interoperable, Highly Scalable Software Package For Weighted Ensemble Simulation and Analysis,” *J. Chem. Theory Comput.*, 11: 800809 (2015).

Russo, J. D., Zhang, S., Leung, J.M.G., Bogetti, A.T., Thompson, J.P., DeGrave, A.J., Torrillo, P.A., Pratt, A.J., Wong, K.F., Xia, J., Copperman, J., Adelman, J.L., Zwier, M.C., LeBard, D.N., Zuckerman, D.M., Chong, L.T. WESTPA 2.0: High-Performance Upgrades for Weighted Ensemble Simulations and Analysis of Longer-Timescale Applications. *J. Chem. Theory Comput.*, 18 (2): 638–649 (2022).

See this [page](#) and this [powerpoint](#) for an overview of weighted ensemble simulation.

To help us fund development and improve WESTPA please fill out a one-minute [survey](#) and consider contributing documentation or code to the WESTPA community.

WESTPA is free software, licensed under the terms of the MIT License. See the file LICENSE for more information.

REQUIREMENTS

WESTPA is written in Python and requires version 3.7 or later. WESTPA also requires a number of Python scientific software packages. The simplest way to meet these requirements is to download the Anaconda Python distribution from www.anaconda.com (free for all users).

WESTPA currently runs on Unix-like operating systems, including Linux and Mac OS X. It is developed and tested on x86_64 machines running Linux.

OBTAINING AND INSTALLING WESTPA

WESTPA is developed and tested on Unix-like operating systems, including Linux and Mac OS X.

Regardless of the chosen method of installation, before installing WESTPA, we recommend you to first install the Python 3 version provided by the latest free [Anaconda Python distribution](#). After installing Anaconda, create a new python environment for the WESTPA install with the following:

```
conda create -n westpa-2.0 python=3.9
conda activate westpa-2.0
```

Then, we recommend installing WESTPA through conda or pip. Execute either of the following:

```
conda install -c conda-forge westpa
```

or:

```
python -m pip install westpa
```

See the install instructions on our [wiki](#) for more detailed information.

To install from source (**not recommended**), start by downloading the corresponding tar.gz file from the [releases page](#). After downloading the file, unpack the file and install WESTPA by executing the following:

```
tar xvzf westpa-main.tar.gz
cd westpa
python -m pip install -e .
```


GETTING STARTED

High-level tutorials of how to use the WESTPA software can be found [here](#). Further, all WESTPA command-line tools provide detailed help when given the `-h/--help` option.

Finally, while WESTPA is a powerful tool that enables expert simulators to access much longer timescales than is practical with standard simulations, there can be a steep learning curve to figuring out how to effectively run the simulations on your computing resource of choice. For serious users who have completed the online tutorials and are ready for production simulations of their system, we invite you to contact Lillian Chong (ltchong AT pitt DOT edu) about spending a few days with her lab and/or setting up video conferencing sessions to help you get your simulations off the ground.

GETTING HELP

WESTPA [FAQ](#)

A mailing list for WESTPA is available, at which one can ask questions (or see if a question one has was previously addressed). This is the preferred means for obtaining help and support. See <http://groups.google.com/group/westpa-users> to sign up or search archived messages.

DEVELOPERS

Search archived messages or post to the westpa-devel Google group: <https://groups.google.com/group/westpa-devel>.

6.1 westpa.cli package

6.1.1 w_init

`w_init` initializes the weighted ensemble simulation, creates the main HDF5 file and prepares the first iteration.

6.1.1.1 Overview

Usage:

```
w_init [-h] [-r RCFILE] [--quiet | --verbose | --debug] [--version]
      [--force] [--bstate-file BSTATE_FILE] [--bstate BSTATES]
      [--tstate-file TSTATE_FILE] [--tstate TSTATES]
      [--segs-per-state N] [--no-we] [--wm-work-manager WORK_MANAGER]
      [--wm-n-workers N_WORKERS] [--wm-zmq-mode MODE]
      [--wm-zmq-info INFO_FILE] [--wm-zmq-task-endpoint TASK_ENDPOINT]
      [--wm-zmq-result-endpoint RESULT_ENDPOINT]
      [--wm-zmq-announce-endpoint ANNOUNCE_ENDPOINT]
      [--wm-zmq-heartbeat-interval INTERVAL]
      [--wm-zmq-task-timeout TIMEOUT] [--wm-zmq-client-comm-mode MODE]
```

Initialize a new WEST simulation, creating the WEST HDF5 file and preparing the first iteration’s segments. Initial states are generated from one or more “basis states” which are specified either in a file specified with `--bstates-from`, or by one or more `--bstate` arguments. If neither `--bstates-from` nor at least one `--bstate` argument is provided, then a default basis state of probability one identified by the state ID zero and label “basis” will be created (a warning will be printed in this case, to remind you of this behavior, in case it is not what you wanted). Target states for (non- equilibrium) steady-state simulations are specified either in a file specified with `--tstates-from`, or by one or more `--tstate` arguments. If neither `--tstates-from` nor at least one `--tstate` argument is provided, then an equilibrium simulation (without any sinks) will be performed.

6.1.1.2 Command-Line Options

See the [general command-line tool reference](#) for more information on the general options.

6.1.1.2.1 State Options

```
--force
  Overwrites any existing simulation data

--bstate BSTATES
  Add the given basis state (specified as a string
  'label,probability[,auxref]') to the list of basis states (after
  those specified in --bstates-from, if any). This argument may be
  specified more than once, in which case the given states are
  appended in the order they are given on the command line.

--bstate-file BSTATE_FILE, --bstates-from BSTATE_FILE
  Read basis state names, probabilities, and (optionally) data
  references from BSTATE_FILE.

--tstate TSTATES
  Add the given target state (specified as a string
  'label,pcoord0[,pcoord1[,...]]') to the list of target states (after
  those specified in the file given by --tstates-from, if any). This
  argument may be specified more than once, in which case the given
  states are appended in the order they appear on the command line.

--tstate-file TSTATE_FILE, --tstates-from TSTATE_FILE
  Read target state names and representative progress coordinates from
  TSTATE_FILE. WESTPA uses the representative progress coordinate of a target state and
  converts the **entire** bin containing that progress coordinate into a
  recycling sink.

--segs-per-state N
  Initialize N segments from each basis state (default: 1).

--no-we, --shotgun
  Do not run the weighted ensemble bin/split/merge algorithm on
  newly-created segments.
```

6.1.1.3 Examples

(TODO: write 3 examples; Setting up the basis states, explanation of bstates and istates. Setting up an equilibrium simulation, w/o target(s) for recycling. Setting up a simulation with one/multiple target states.)

6.1.1.4 westpa.cli.core.w_init module

class westpa.cli.core.w_init.BasisState(*label, probability, pcoord=None, auxref=None, state_id=None*)

Bases: object

Describes an basis (micro)state. These basis states are used to generate initial states for new trajectories, either at the beginning of the simulation (i.e. at w_init) or due to recycling.

Variables

- **state_id** – Integer identifier of this state, usually set by the data manager.
- **label** – A descriptive label for this microstate (may be empty)
- **probability** – Probability of this state to be selected when creating a new trajectory.
- **pcoord** – The representative progress coordinate of this state.
- **auxref** – A user-provided (string) reference for locating data associated with this state (usually a filesystem path).

classmethod states_to_file(*states, fileobj*)

Write a file defining basis states, which may then be read by *states_from_file()*.

classmethod states_from_file(*statefile*)

Read a file defining basis states. Each line defines a state, and contains a label, the probability, and optionally a data reference, separated by whitespace, as in:

```
unbound    1.0
```

or:

```
unbound_0  0.6    state0.pdb
unbound_1  0.4    state1.pdb
```

as_numpy_record()

Return the data for this state as a numpy record array.

class westpa.cli.core.w_init.TargetState(*label, pcoord, state_id=None*)

Bases: object

Describes a target state.

Variables

- **state_id** – Integer identifier of this state, usually set by the data manager.
- **label** – A descriptive label for this microstate (may be empty)
- **pcoord** – The representative progress coordinate of this state.

classmethod states_to_file(*states, fileobj*)

Write a file defining basis states, which may then be read by *states_from_file()*.

classmethod `states_from_file(statefile, dtype)`

Read a file defining target states. Each line defines a state, and contains a label followed by a representative progress coordinate value, separated by whitespace, as in:

```
bound      0.02
```

for a single target and one-dimensional progress coordinates or:

```
bound      2.7      0.0
drift      100      50.0
```

for two targets and a two-dimensional progress coordinate.

`westpa.cli.core.w_init.make_work_manager()`

Using cues from the environment, instantiate a pre-configured work manager.

`westpa.cli.core.w_init.entry_point()`

`westpa.cli.core.w_init.initialize(tstates, tstate_file, bstates, bstate_file, sstates=None, sstate_file=None, segs_per_state=1, shotgun=False)`

Initialize a WESTPA simulation.

`tstates` : list of str

`tstate_file` : str

`bstates` : list of str

`bstate_file` : str

`sstates` : list of str

`sstate_file` : str

`segs_per_state` : int

`shotgun` : bool

6.1.2 w_bins

`w_bins` deals with binning modification and statistics

6.1.2.1 Overview

Usage:

```
w_bins [-h] [-r RCFILE] [--quiet | --verbose | --debug] [--version]
        [-W WEST_H5FILE]
        {info, rebin} ...
```

Display information and statistics about binning in a WEST simulation, or modify the binning for the current iteration of a WEST simulation.

6.1.2.2 Command-Line Options

See the [general command-line tool reference](#) for more information on the general options.

6.1.2.2.1 Options Under ‘info’

Usage:

```
w_bins info [-h] [-n N_ITER] [--detail]
            [--bins-from-system | --bins-from-expr BINS_FROM_EXPR | --bins-from-
            ↪function BINS_FROM_FUNCTION | --bins-from-file]
```

Positional options:

```
info
  Display information about binning.
```

Options for ‘info’:

```
-n N_ITER, --n-iter N_ITER
  Consider initial points of segment N_ITER (default: current
  iteration).

--detail
  Display detailed per-bin information in addition to summary
  information.
```

Binning options for ‘info’:

```
--bins-from-system
  Bins are constructed by the system driver specified in the WEST
  configuration file (default where stored bin definitions not
  available).

--bins-from-expr BINS_FROM_EXPR, --binbounds BINS_FROM_EXPR
  Construct bins on a rectilinear grid according to the given BINEXPR.
  This must be a list of lists of bin boundaries (one list of bin
  boundaries for each dimension of the progress coordinate), formatted
  as a Python expression. E.g. "[[0,1,2,4,inf],[-inf,0,inf]]". The
  numpy module and the special symbol "inf" (for floating-point
  infinity) are available for use within BINEXPR.

--bins-from-function BINS_FROM_FUNCTION, --binfunc BINS_FROM_FUNCTION
  Supply an external function which, when called, returns a properly
  constructed bin mapper which will then be used for bin assignments.
  This should be formatted as "[PATH:]MODULE.FUNC", where the function
  FUNC in module MODULE will be used; the optional PATH will be
  prepended to the module search path when loading MODULE.

--bins-from-file
  Load bin specification from the data file being examined (default
  where stored bin definitions available).
```

6.1.2.2.2 Options Under ‘rebin’

Usage:

```
w_bins rebin [-h] [--confirm] [--detail]
              [--bins-from-system | --bins-from-expr BINS_FROM_EXPR | --bins-from-
↪function BINS_FROM_FUNCTION]
              [--target-counts TARGET_COUNTS | --target-counts-from FILENAME]
```

Positional option:

```
rebin
  Rebuild current iteration with new binning.
```

Options for ‘rebin’:

```
--confirm
  Commit the revised iteration to HDF5; without this option, the
  effects of the new binning are only calculated and printed.

--detail
  Display detailed per-bin information in addition to summary
  information.
```

Binning options for ‘rebin’;

Same as the binning options for ‘info’.

Bin target count options for ‘rebin’;

```
--target-counts TARGET_COUNTS
  Use TARGET_COUNTS instead of stored or system driver target counts.
  TARGET_COUNTS is a comma-separated list of integers. As a special
  case, a single integer is acceptable, in which case the same target
  count is used for all bins.

--target-counts-from FILENAME
  Read target counts from the text file FILENAME instead of using
  stored or system driver target counts. FILENAME must contain a list
  of integers, separated by arbitrary whitespace (including newlines).
```

6.1.2.3 Input Options

```
-W WEST_H5FILE, --west_data WEST_H5FILE
  Take WEST data from WEST_H5FILE (default: read from the HDF5 file
  specified in west.cfg).
```

6.1.2.4 Examples

(TODO: Write up an example)

6.1.2.5 westpa.cli.tools.w_bins module

class westpa.cli.tools.w_bins.WESTTool

Bases: [WESTToolComponent](#)

Base class for WEST command line tools

prog = None

usage = None

description = None

epilog = None

add_args(*parser*)

Add arguments specific to this tool to the given argparse parser.

process_args(*args*)

Take argparse-processed arguments associated with this tool and deal with them appropriately (setting instance variables, etc)

make_parser(*prog=None, usage=None, description=None, epilog=None, args=None*)

make_parser_and_process(*prog=None, usage=None, description=None, epilog=None, args=None*)

A convenience function to create a parser, call `add_all_args()`, and then call `process_all_args()`. The argument namespace is returned.

go()

Perform the analysis associated with this tool.

main()

A convenience function to make a parser, parse and process arguments, then call `self.go()`

class westpa.cli.tools.w_bins.WESTDataReader

Bases: [WESTToolComponent](#)

Tool for reading data from WEST-related HDF5 files. Coordinates finding the main HDF5 file from `west.cfg` or command line arguments, caching of certain kinds of data (eventually), and retrieving auxiliary data sets from various places.

add_args(*parser*)

Add arguments specific to this component to the given argparse parser.

process_args(*args*)

Take argparse-processed arguments associated with this component and deal with them appropriately (setting instance variables, etc)

open(*mode='r'*)

close()

property `weight_dsspec`

property `parent_id_dsspec`

class `westpa.cli.tools.w_bins.BinMappingComponent`

Bases: [*WESTToolComponent*](#)

Component for obtaining a bin mapper from one of several places based on command-line arguments. Such locations include an HDF5 file that contains pickled mappers (including the primary WEST HDF5 file), the system object, an external function, or (in the common case of rectilinear bins) a list of lists of bin boundaries.

Some configuration is necessary prior to calling `process_args()` if loading a mapper from HDF5. Specifically, either `set_we_h5file_info()` or `set_other_h5file_info()` must be called to describe where to find the appropriate mapper. In the case of `set_we_h5file_info()`, the mapper used for WE at the end of a given iteration will be loaded. In the case of `set_other_h5file_info()`, an arbitrary group and hash value are specified; the mapper corresponding to that hash in the given group will be returned.

In the absence of arguments, the mapper contained in an existing HDF5 file is preferred; if that is not available, the mapper from the system driver is used.

This component adds the following arguments to argument parsers:

--bins-from-system Obtain bins from the system driver

—bins-from-expr=EXPR Construct rectilinear bins by parsing EXPR and calling `RectilinearBinMapper()` with the result. EXPR must therefore be a list of lists.

--bins-from-function=[PATH:]MODULE.FUNC

Call an external function FUNC in module MODULE (optionally adding PATH to the search path when loading MODULE) which, when called, returns a fully-constructed bin mapper.

—bins-from-file Load bin definitions from a YAML configuration file.

--bins-from-h5file Load bins from the file being considered; this is intended to mean the master WEST HDF5 file or results of other binning calculations, as appropriate.

add_args(*parser*, *description*='binning options', *suppress*=[])

Add arguments specific to this component to the given argparse parser.

add_target_count_args(*parser*, *description*='bin target count options')

Add options to the given parser corresponding to target counts.

process_args(*args*)

Take argparse-processed arguments associated with this component and deal with them appropriately (setting instance variables, etc)

set_we_h5file_info(*n_iter*=None, *data_manager*=None, *required*=False)

Set up to load a bin mapper from the master WEST HDF5 file. The mapper is actually loaded from the file when `self.load_bin_mapper()` is called, if and only if command line arguments direct this. If `required` is true, then a mapper must be available at iteration `n_iter`, or else an exception will be raised.

set_other_h5file_info(*topology_group*, *hashval*)

Set up to load a bin mapper from (any) open HDF5 file, where bin topologies are stored in `topology_group` (an `h5py Group` object) and the desired mapper has hash value `hashval`. The mapper itself is loaded when `self.load_bin_mapper()` is called.


```
westpa.cli.tools.w_bins.write_bin_info(mapper, assignments, weights, n_target_states,
                                       outfile=<_io.TextIOWrapper name='<stdout>' mode='w'
                                       encoding='utf-8'>, detailed=False)
```

Write information about binning to outfile, given a mapper (*mapper*) and the weights (*weights*) and bin assignments (*assignments*) of a set of segments, along with a target state count (*n_target_states*). If *detailed* is true, then per-bin information is written as well as summary information about all bins.

```
class westpa.cli.tools.w_bins.WBinTool
```

Bases: [WESTTool](#)

```
prog = 'w_bins'
```

```
description = 'Display information and statistics about binning in a WEST
simulation, or\nmodify the binning for the current iteration of a WEST simulation.\n
n-----\n'
```

```
add_args(parser)
```

Add arguments specific to this tool to the given argparse parser.

```
process_args(args)
```

Take argparse-processed arguments associated with this tool and deal with them appropriately (setting instance variables, etc)

```
go()
```

Perform the analysis associated with this tool.

```
cmd_info()
```

```
cmd_rebin()
```

```
westpa.cli.tools.w_bins.entry_point()
```

6.1.3 w_run

w_run starts or continues a weighted ensemble simulation.

6.1.3.1 Overview

Usage:

```
w_run [-h] [-r RCFILE] [--quiet | --verbose | --debug] [--version]
      [--oneseg ] [--wm-work-manager WORK_MANAGER]
      [--wm-n-workers N_WORKERS] [--wm-zmq-mode MODE]
      [--wm-zmq-info INFO_FILE] [--wm-zmq-task-endpoint TASK_ENDPOINT]
      [--wm-zmq-result-endpoint RESULT_ENDPOINT]
      [--wm-zmq-announce-endpoint ANNOUNCE_ENDPOINT]
      [--wm-zmq-heartbeat-interval INTERVAL]
      [--wm-zmq-task-timeout TIMEOUT] [--wm-zmq-client-comm-mode MODE]
```

6.1.3.2 Command-Line Options

See the *command-line tool index* for more information on the general options.

6.1.3.2.1 Segment Options

::

--oneseg Only propagate one segment (useful for debugging propagators)

6.1.3.3 Example

A simple example for using `w_run` (mostly taken from `odld` example that is available in the main WESTPA distribution):

```
w_run &> west.log
```

This commands starts up a serial weighted ensemble run and pipes the results into the `west.log` file. As a side note `--debug` option is very useful for debugging the code if something goes wrong.

6.1.3.4 westpa.cli.core.w_run module

`westpa.cli.core.w_run.make_work_manager()`

Using cues from the environment, instantiate a pre-configured work manager.

`westpa.cli.core.w_run.entry_point()`

`westpa.cli.core.w_run.run_simulation()`

6.1.4 w_truncate

`w_truncate` removes all iterations after a certain point

6.1.4.1 Overview

Usage:

```
w_truncate [-h] [-r RCFILE] [--quiet | --verbose | --debug] [--version]
           [-n N_ITER] [-W WEST_H5FILE]
```

Remove all iterations after a certain point in a

6.1.4.2 Command-Line Options

See the *command-line tool index* <*command_line_tool_index*> for more information on the general options.

6.1.4.2.1 Iteration Options

```
-n N_ITER, --iter N_ITER
    Truncate this iteration and those following.

-W WEST_H5FILE, --west-data WEST_H5FILE
    PATH of H5 file to truncate. By default, it will read from the RCFILE (e.g., west.cfg).
    This option will have override whatever's provided in the RCFILE.
```

6.1.4.3 Examples

Running the following will remove iteration 50 and all iterations after 50 from multi.h5.

```
w_truncate -n 50 -W multi.h5
```

6.1.4.4 westpa.cli.core.w_truncate module

```
westpa.cli.core.w_truncate.entry_point()
```

6.1.5 w_fork

usage:

```
w_fork [-h] [-r RCFILE] [--quiet | --verbose | --debug] [--version] [-i INPUT_H5FILE]
        [-I N_ITER] [-o OUTPUT_H5FILE] [--istate-map ISTATE_MAP] [--no-headers]
```

Prepare a new weighted ensemble simulation from an existing one at a particular point. A new HDF5 file is generated. In the case of executable propagation, it is the user's responsibility to prepare the new simulation directory appropriately, particularly making the old simulation's restart data from the appropriate iteration available as the new simulations initial state data; a mapping of old simulation segment to new simulation initial states is created, both in the new HDF5 file and as a flat text file, to aid in this. Target states and basis states for the new simulation are taken from those in the original simulation.

optional arguments:

```
-h, --help            show this help message and exit
-i INPUT_H5FILE, --input INPUT_H5FILE
                      Create simulation from the given INPUT_H5FILE (default: read from
↳ configuration
                      file.
-I N_ITER, --iteration N_ITER
                      Take initial distribution for new simulation from iteration N_ITER.
↳ (default:
                      last complete iteration).
-o OUTPUT_H5FILE, --output OUTPUT_H5FILE
                      Save new simulation HDF5 file as OUTPUT (default: forked.h5).
--istate-map ISTATE_MAP
                      Write text file describing mapping of existing segments to new
↳ initial states
                      in ISTATE_MAP (default: istate_map.txt).
--no-headers          Do not write header to ISTATE_MAP
```

general options:

```
-r RCFILE, --rcfile RCFILE          use RCFILE as the WEST run-time configuration file (default: west.
→cfg)
--quiet                             emit only essential information
--verbose                           emit extra information
--debug                             enable extra checks and emit copious information
--version                           show program's version number and exit
```

6.1.5.1 westpa.cli.tools.w_fork module

```
class westpa.cli.core.w_fork.Segment(n_iter=None, seg_id=None, weight=None, endpoint_type=None,
parent_id=None, wtg_parent_ids=None, pcoord=None,
status=None, walltime=None, cputime=None, data=None)
```

Bases: object

A class wrapping segment data that must be passed through the work manager or data manager. Most fields are self-explanatory. One item worth noting is that a negative parent ID means that the segment starts from the initial state with ID $-(\text{segment.parent_id}+1)$

SEG_STATUS_UNSET = 0

SEG_STATUS_PREPARED = 1

SEG_STATUS_COMPLETE = 2

SEG_STATUS_FAILED = 3

SEG_INITPOINT_UNSET = 0

SEG_INITPOINT_CONTINUES = 1

SEG_INITPOINT_NEWTRAJ = 2

SEG_ENDPOINT_UNSET = 0

SEG_ENDPOINT_CONTINUES = 1

SEG_ENDPOINT_MERGED = 2

SEG_ENDPOINT_RECYCLED = 3

```
statuses = {'SEG_STATUS_COMPLETE': 2, 'SEG_STATUS_FAILED': 3, 'SEG_STATUS_PREPARED':
1, 'SEG_STATUS_UNSET': 0}
```

```
initpoint_types = {'SEG_INITPOINT_CONTINUES': 1, 'SEG_INITPOINT_NEWTRAJ': 2,
'SEG_INITPOINT_UNSET': 0}
```

```
endpoint_types = {'SEG_ENDPOINT_CONTINUES': 1, 'SEG_ENDPOINT_MERGED': 2,
'SEG_ENDPOINT_RECYCLED': 3, 'SEG_ENDPOINT_UNSET': 0}
```

```
status_names = {0: 'SEG_STATUS_UNSET', 1: 'SEG_STATUS_PREPARED', 2:
'SEG_STATUS_COMPLETE', 3: 'SEG_STATUS_FAILED'}
```

```
initpoint_type_names = {0: 'SEG_INITPOINT_UNSET', 1: 'SEG_INITPOINT_CONTINUES', 2:
'SEG_INITPOINT_NEWTRAJ'}
```

```
endpoint_type_names = {0: 'SEG_ENDPOINT_UNSET', 1: 'SEG_ENDPOINT_CONTINUES', 2:
'SEG_ENDPOINT_MERGED', 3: 'SEG_ENDPOINT_RECYCLED'}
```

```
static initial_pcoord(segment)
```

Return the initial progress coordinate point of this segment.

```
static final_pcoord(segment)
```

Return the final progress coordinate point of this segment.

```
property initpoint_type
```

```
property initial_state_id
```

```
property status_text
```

```
property endpoint_type_text
```

```
class westpa.cli.core.w_fork.InitialState(state_id, basis_state_id, iter_created, iter_used=None,
                                         istate_type=None, istate_status=None, pcoord=None,
                                         basis_state=None, basis_auxref=None)
```

Bases: object

Describes an initial state for a new trajectory. These are generally constructed by appropriate modification of a basis state.

Variables

- **state_id** – Integer identifier of this state, usually set by the data manager.
- **basis_state_id** – Identifier of the basis state from which this state was generated, or None.
- **basis_state** – The *BasisState* from which this state was generated, or None.
- **iter_created** – Iteration in which this state was generated (0 for simulation initialization).
- **iter_used** – Iteration in which this state was used to initiate a trajectory (None for unused).
- **istate_type** – Integer describing the type of this initial state (ISTATE_TYPE_BASIS for direct use of a basis state, ISTATE_TYPE_GENERATED for a state generated from a basis state, ISTATE_TYPE_RESTART for a state corresponding to the endpoint of a segment in another simulation, or ISTATE_TYPE_START for a state generated from a start state).
- **istate_status** – Integer describing whether this initial state has been properly prepared.
- **pcoord** – The representative progress coordinate of this state.

```
ISTATE_TYPE_UNSET = 0
```

```
ISTATE_TYPE_BASIS = 1
```

```
ISTATE_TYPE_GENERATED = 2
```

```
ISTATE_TYPE_RESTART = 3
```

```
ISTATE_TYPE_START = 4
```

```
ISTATE_UNUSED = 0
```

```
ISTATE_STATUS_PENDING = 0
```

```
ISTATE_STATUS_PREPARED = 1

ISTATE_STATUS_FAILED = 2

istate_types = {'ISTATE_TYPE_BASIS': 1, 'ISTATE_TYPE_GENERATED': 2,
                'ISTATE_TYPE_RESTART': 3, 'ISTATE_TYPE_START': 4, 'ISTATE_TYPE_UNSET': 0}

istate_type_names = {0: 'ISTATE_TYPE_UNSET', 1: 'ISTATE_TYPE_BASIS', 2:
                    'ISTATE_TYPE_GENERATED', 3: 'ISTATE_TYPE_RESTART', 4: 'ISTATE_TYPE_START'}

istate_statuses = {'ISTATE_STATUS_FAILED': 2, 'ISTATE_STATUS_PENDING': 0,
                  'ISTATE_STATUS_PREPARED': 1}

istate_status_names = {0: 'ISTATE_STATUS_PENDING', 1: 'ISTATE_STATUS_PREPARED', 2:
                      'ISTATE_STATUS_FAILED'}

as_numpy_record()

westpa.cli.core.w_fork.n_iter_dtype
    alias of uint32

westpa.cli.core.w_fork.seg_id_dtype
    alias of int64

westpa.cli.core.w_fork.entry_point()
```

6.1.6 w_assign

`w_assign` uses simulation output to assign walkers to user-specified bins and macrostates. These assignments are required for some other simulation tools, namely `w_kinetics` and `w_kinavg`.

`w_assign` supports parallelization (see [general work manager options](#) for more on command line options to specify a work manager).

6.1.6.1 Overview

Usage:

```
w_assign [-h] [-r RCFILE] [--quiet | --verbose | --debug] [--version]
          [-W WEST_H5FILE] [-o OUTPUT]
          [--bins-from-system | --bins-from-expr BINS_FROM_EXPR | --bins-from-
↪function BINS_FROM_FUNCTION]
          [-p MODULE.FUNCTION]
          [--states STATEDEF [STATEDEF ...] | --states-from-file STATEFILE | --
↪states-from-function STATEFUNC]
          [--wm-work-manager WORK_MANAGER] [--wm-n-workers N_WORKERS]
          [--wm-zmq-mode MODE] [--wm-zmq-info INFO_FILE]
          [--wm-zmq-task-endpoint TASK_ENDPOINT]
          [--wm-zmq-result-endpoint RESULT_ENDPOINT]
          [--wm-zmq-announce-endpoint ANNOUNCE_ENDPOINT]
          [--wm-zmq-listen-endpoint ANNOUNCE_ENDPOINT]
          [--wm-zmq-heartbeat-interval INTERVAL]
          [--wm-zmq-task-timeout TIMEOUT]
          [--wm-zmq-client-comm-mode MODE]
```

6.1.6.2 Command-Line Options

See the [general command-line tool reference](#) for more information on the general options.

6.1.6.3 Input/output Options

`-W, --west-data /path/to/file`

Read simulation result data **from file** `*file*`. (**Default:** The `*hdf5*` file specified **in** the configuration file, by default `**west.h5**`)

`-o, --output /path/to/file`

Write assignment results to file `*outfile*`. (**Default:** `*hdf5*` file `**assign.h5**`)

6.1.6.4 Binning Options

Specify how binning is to be assigned to the dataset.:

`--bins-from-system`

Use binning scheme specified by the system driver; system driver can be found in the west configuration file, by default named `**west.cfg**` (**Default binning**)

`--bins-from-expr bin_expr`

Use binning scheme specified in `*``bin_expr``*`, which takes the form a Python list of lists, where each inner list corresponds to the binning a given dimension. (for example, `"[[0,1,2,4,inf],[-inf,0,inf]]"` specifies bin boundaries for two dimensional progress coordinate. Note that this option accepts the special symbol 'inf' for floating point infinity

`--bins-from-function bin_func`

Bins specified by calling an external function `*``bin_func``*`. `*``bin_func``*` should be formatted as `'[PATH:]module.function'`, where the function 'function' in module 'module' will be used

6.1.6.5 Macrostate Options

You can optionally specify how to assign user-defined macrostates. Note that macrostates must be assigned for subsequent analysis tools, namely `w_kinetics` and `w_kinavg`.:

`--states statedef [statedef ...]`

Specify a macrostate for a single bin as `*``statedef``*`, formatted as a coordinate tuple where each coordinate specifies the bin to which it belongs, for instance:

`'[1.0, 2.0]'` assigns a macrostate corresponding to the bin that contains the (two-dimensional) progress coordinates 1.0 and 2.0. Note that a macrostate label can optionally be specified, for instance: `'bound:[1.0, 2.0]'` assigns the corresponding bin

(continues on next page)

(continued from previous page)

```

containing the given coordinates the macrostate named 'bound'. Note
that multiple assignments can be specified with this command, but
only one macrostate per bin is possible - if you wish to specify
multiple bins in a single macrostate, use the
*`--states-from-file`* option.

--states-from-file statefile
Read macrostate assignments from *yaml* file *`statefile`*. This
option allows you to assign multiple bins to a single macrostate.
The following example shows the contents of *`statefile`* that
specify two macrostates, bound and unbound, over multiple bins with
a two-dimensional progress coordinate:

---
states:
- label: unbound
  coords:
    - [9.0, 1.0]
    - [9.0, 2.0]
- label: bound
  coords:
    - [0.1, 0.0]

```

6.1.6.6 Specifying Progress Coordinate

By default, progress coordinate information for each iteration is taken from *pcoord* dataset in the specified input file (which, by default is *west.h5*). Optionally, you can specify a function to construct the progress coordinate for each iteration - this may be useful to consolidate data from several sources or otherwise preprocess the progress coordinate data.:

```

--construct-pcoord module.function, -p module.function
Use the function *module.function* to construct the progress
coordinate for each iteration. This will be called once per
iteration as *function(n_iter, iter_group)* and should return an
array indexable as [seg_id][timepoint][dimension]. The
**default** function returns the 'pcoord' dataset for that iteration
(i.e. the function executes return iter_group['pcoord'][...])

```

6.1.6.7 Examples

6.1.6.8 westpa.cli.tools.w_assign module

```

westpa.cli.tools.w_assign.seg_id_dtype
    alias of int64
westpa.cli.tools.w_assign.weight_dtype
    alias of float64
westpa.cli.tools.w_assign.index_dtype
    alias of uint16

```



```
westpa.cli.tools.w_assign.assign_and_label(nsegs_lb, nsegs_ub, parent_ids, assign, nstates, state_map,
                                             last_labels, pcoords, subsample)
```

Assign trajectories to bins and last-visited macrostates for each timepoint.

```
westpa.cli.tools.w_assign.accumulate_labeled_populations(weights, bin_assignments,
                                                         label_assignments, labeled_bin_pops)
```

For a set of segments in one iteration, calculate the average population in each bin, with separation by last-visited macrostate.

```
class westpa.cli.tools.w_assign.WESTParallelTool(wm_env=None)
```

Bases: [WESTTool](#)

Base class for command-line tools parallelized with wwmgr. This automatically adds and processes wwmgr command-line arguments and creates a work manager at self.work_manager.

```
make_parser_and_process(prog=None, usage=None, description=None, epilog=None, args=None)
```

A convenience function to create a parser, call add_all_args(), and then call process_all_args(). The argument namespace is returned.

```
add_args(parser)
```

Add arguments specific to this tool to the given argparse parser.

```
process_args(args)
```

Take argparse-processed arguments associated with this tool and deal with them appropriately (setting instance variables, etc)

```
go()
```

Perform the analysis associated with this tool.

```
main()
```

A convenience function to make a parser, parse and process arguments, then run self.go() in the master process.

```
class westpa.cli.tools.w_assign.WESTDataReader
```

Bases: [WESTToolComponent](#)

Tool for reading data from WEST-related HDF5 files. Coordinates finding the main HDF5 file from west.cfg or command line arguments, caching of certain kinds of data (eventually), and retrieving auxiliary data sets from various places.

```
add_args(parser)
```

Add arguments specific to this component to the given argparse parser.

```
process_args(args)
```

Take argparse-processed arguments associated with this component and deal with them appropriately (setting instance variables, etc)

```
open(mode='r')
```

```
close()
```

```
property weight_dsspec
```

```
property parent_id_dsspec
```

```
class westpa.cli.tools.w_assign.WESTDSSynthesizer(default_dsname=None, h5filename=None)
```

Bases: [WESTToolComponent](#)

Tool for synthesizing a dataset for analysis from other datasets. This may be done using a custom function, or a list of “data set specifications”. It is anticipated that if several source datasets are required, then a tool will have multiple instances of this class.

```
group_name = 'input dataset options'
```

```
add_args(parser)
```

Add arguments specific to this component to the given argparse parser.

```
process_args(args)
```

Take argparse-processed arguments associated with this component and deal with them appropriately (setting instance variables, etc)

```
class westpa.cli.tools.w_assign.BinMappingComponent
```

Bases: [WESTToolComponent](#)

Component for obtaining a bin mapper from one of several places based on command-line arguments. Such locations include an HDF5 file that contains pickled mappers (including the primary WEST HDF5 file), the system object, an external function, or (in the common case of rectilinear bins) a list of lists of bin boundaries.

Some configuration is necessary prior to calling `process_args()` if loading a mapper from HDF5. Specifically, either `set_we_h5file_info()` or `set_other_h5file_info()` must be called to describe where to find the appropriate mapper. In the case of `set_we_h5file_info()`, the mapper used for WE at the end of a given iteration will be loaded. In the case of `set_other_h5file_info()`, an arbitrary group and hash value are specified; the mapper corresponding to that hash in the given group will be returned.

In the absence of arguments, the mapper contained in an existing HDF5 file is preferred; if that is not available, the mapper from the system driver is used.

This component adds the following arguments to argument parsers:

--bins-from-system Obtain bins from the system driver

—bins-from-expr=EXPR Construct rectilinear bins by parsing EXPR and calling RectilinearBinMapper() with the result. EXPR must therefore be a list of lists.

--bins-from-function=[PATH:]MODULE.FUNC

Call an external function FUNC in module MODULE (optionally adding PATH to the search path when loading MODULE) which, when called, returns a fully-constructed bin mapper.

—bins-from-file Load bin definitions from a YAML configuration file.

--bins-from-h5file Load bins from the file being considered; this is intended to mean the master WEST HDF5 file or results of other binning calculations, as appropriate.

```
add_args(parser, description='binning options', suppress=[])
```

Add arguments specific to this component to the given argparse parser.

```
add_target_count_args(parser, description='bin target count options')
```

Add options to the given parser corresponding to target counts.

```
process_args(args)
```

Take argparse-processed arguments associated with this component and deal with them appropriately (setting instance variables, etc)

set_we_h5file_info(*n_iter=None, data_manager=None, required=False*)

Set up to load a bin mapper from the master WEST HDF5 file. The mapper is actually loaded from the file when `self.load_bin_mapper()` is called, if and only if command line arguments direct this. If `required` is true, then a mapper must be available at iteration `n_iter`, or else an exception will be raised.

set_other_h5file_info(*topology_group, hashval*)

Set up to load a bin mapper from (any) open HDF5 file, where bin topologies are stored in `topology_group` (an `h5py Group` object) and the desired mapper has hash value `hashval`. The mapper itself is loaded when `self.load_bin_mapper()` is called.

class `westpa.cli.tools.w_assign.ProgressIndicatorComponent`

Bases: [`WESTToolComponent`](#)

add_args(*parser*)

Add arguments specific to this component to the given argparse parser.

process_args(*args*)

Take argparse-processed arguments associated with this component and deal with them appropriately (setting instance variables, etc)

class `westpa.cli.tools.w_assign.WESTPAH5File(*args, **kwargs)`

Bases: `File`

Generalized input/output for WESTPA simulation (or analysis) data.

Create a new file object.

See the `h5py` user guide for a detailed explanation of the options.

name

Name of the file on disk, or file-like object. Note: for files created with the ‘core’ driver, HDF5 still requires this be non-empty.

mode

r Readonly, file must exist (default) r+ Read/write, file must exist w Create file, truncate if exists w- or x Create file, fail if exists a Read/write if exists, create otherwise

driver

Name of the driver to use. Legal values are `None` (default, recommended), ‘core’, ‘sec2’, ‘direct’, ‘stdio’, ‘mpio’, ‘ros3’.

libver

Library version bounds. Supported values: ‘earliest’, ‘v108’, ‘v110’, ‘v112’ and ‘latest’. The ‘v108’, ‘v110’ and ‘v112’ options can only be specified with the HDF5 1.10.2 library or later.

userblock_size

Desired size of user block. Only allowed when creating a new file (mode w, w- or x).

swmr

Open the file in SWMR read mode. Only used when mode = ‘r’.

rdcc_nbytes

Total size of the dataset chunk cache in bytes. The default size is 1024×2 (1 MiB) per dataset. Applies to all datasets unless individually changed.

rdcc_w0

The chunk preemption policy for all datasets. This must be between 0 and 1 inclusive and indicates the weighting according to which chunks which have been fully read or written are penalized when determining which chunks to flush from cache. A value of 0 means fully read or written chunks are treated no differently than other chunks (the preemption is strictly LRU) while a value of 1 means fully read or written chunks are always preempted before other chunks. If your application only reads or writes data once, this can be

safely set to 1. Otherwise, this should be set lower depending on how often you re-read or re-write the same data. The default value is 0.75. Applies to all datasets unless individually changed.

rdcc_nslots

The number of chunk slots in the raw data chunk cache for this file. Increasing this value reduces the number of cache collisions, but slightly increases the memory used. Due to the hashing strategy, this value should ideally be a prime number. As a rule of thumb, this value should be at least 10 times the number of chunks that can fit in rdcc_nbytes bytes. For maximum performance, this value should be set approximately 100 times that number of chunks. The default value is 521. Applies to all datasets unless individually changed.

track_order

Track dataset/group/attribute creation order under root group if True. If None use global default h5.get_config().track_order.

fs_strategy

The file space handling strategy to be used. Only allowed when creating a new file (mode w, w- or x). Defined as: “fsm” FSM, Aggregators, VFD “page” Paged FSM, VFD “aggregate” Aggregators, VFD “none” VFD If None use HDF5 defaults.

fs_page_size

File space page size in bytes. Only used when fs_strategy=“page”. If None use the HDF5 default (4096 bytes).

fs_persist

A boolean value to indicate whether free space should be persistent or not. Only allowed when creating a new file. The default value is False.

fs_threshold

The smallest free-space section size that the free space manager will track. Only allowed when creating a new file. The default value is 1.

page_buf_size

Page buffer size in bytes. Only allowed for HDF5 files created with fs_strategy=“page”. Must be a power of two value and greater or equal than the file space page size when creating the file. It is not used by default.

min_meta_keep

Minimum percentage of metadata to keep in the page buffer before allowing pages containing metadata to be evicted. Applicable only if page_buf_size is set. Default value is zero.

min_raw_keep

Minimum percentage of raw data to keep in the page buffer before allowing pages containing raw data to be evicted. Applicable only if page_buf_size is set. Default value is zero.

locking

The file locking behavior. Defined as:

- False (or “false”) – Disable file locking
- True (or “true”) – Enable file locking
- “best-effort” – Enable file locking but ignore some errors
- None – Use HDF5 defaults

Warning: The HDF5_USE_FILE_LOCKING environment variable can override this parameter.

Only available with HDF5 \geq 1.12.1 or 1.10.x \geq 1.10.7.

alignment_threshold

Together with alignment_interval, this property ensures that any file object greater than or equal in

size to the alignment threshold (in bytes) will be aligned on an address which is a multiple of alignment interval.

alignment_interval

This property should be used in conjunction with `alignment_threshold`. See the description above. For more details, see https://portal.hdfgroup.org/display/HDF5/H5P_SET_ALIGNMENT

meta_block_size

Set the current minimum size, in bytes, of new metadata block allocations. See https://portal.hdfgroup.org/display/HDF5/H5P_SET_META_BLOCK_SIZE

Additional keywords

Passed on to the selected file driver.

default_iter_prec = 8

replace_dataset(*args, **kwargs)

iter_object_name(n_iter, prefix="", suffix="")

Return a properly-formatted per-iteration name for iteration `n_iter`. (This is used in `create/require/get_iter_group`, but may also be useful for naming datasets on a per-iteration basis.)

create_iter_group(n_iter, group=None)

Create a per-iteration data storage group for iteration number `n_iter` in the group `group` (which is `'iterations'` by default).

require_iter_group(n_iter, group=None)

Ensure that a per-iteration data storage group for iteration number `n_iter` is available in the group `group` (which is `'iterations'` by default).

get_iter_group(n_iter, group=None)

Get the per-iteration data group for iteration number `n_iter` from within the group `group` (`'iterations'` by default).

`westpa.cli.tools.w_assign.get_object(object_name, path=None)`

Attempt to load the given object, using additional path information if given.

`westpa.cli.tools.w_assign.parse_pcoord_value(pc_str)`

class `westpa.cli.tools.w_assign.WAssign`

Bases: `WESTParallelTool`

`prog = 'w_assign'`

```

description = 'Assign walkers to bins, producing a file (by default named
"assign.h5")\nwhich can be used in subsequent analysis.\n\nFor consistency in
subsequent analysis operations, the entire dataset\nmust be assigned, even if only a
subset of the data will be used. This\nensures that analyses that rely on tracing
trajectories always know the\noriginating bin of each trajectory.\n\n\
n-----\
nSource
data\n-----\
\n\nSource data is provided either by a user-specified
function\n(--construct-dataset) or a list of "data set specifications"
(--dsspecs).\nIf neither is provided, the progress coordinate dataset \\'pcoord\''
is used.\n\nTo use a custom function to extract or calculate data whose
probability\ndistribution will be calculated, specify the function in standard
Python\nMODULE.FUNCTION syntax as the argument to --construct-dataset. This
function\nwill be called as function(n_iter,iter_group), where n_iter is the
iteration\nwhose data are being considered and iter_group is the corresponding
group\nin the main WEST HDF5 file (west.h5). The function must return data which
can\nbe indexed as [segment][timepoint][dimension].\n\nTo use a list of data set
specifications, specify --dsspecs and then list the\ndesired datasets one-by-one
(space-separated in most shells). These data set\nspecifications are formatted as
NAME[,file=FILENAME,slice=SLICE], which will\nuse the dataset called NAME in the
HDF5 file FILENAME (defaulting to the main\nWEST HDF5 file west.h5), and slice it
with the Python slice expression SLICE\n(as in [0:2] to select the first two
elements of the first axis of the\ndataset). The ``slice`` option is most useful for
selecting one column (or\nmore) from a multi-column dataset, such as arises when
using a progress\ncoordinate of multiple dimensions.\n\n\
n-----\
nSpecifying
macrostates\
n-----\n
\nOptionally, kinetic macrostates may be defined in terms of sets of bins.\nEach
trajectory will be labeled with the kinetic macrostate it was most\nrecently in at
each timepoint, for use in subsequent kinetic analysis.\nThis is required for all
kinetics analysis (w_kintrace and w_kinmat).\n\nThere are three ways to specify
macrostates:\n\n1. States corresponding to single bins may be identified on the
command\nline using the --states option, which takes multiple arguments, one for\n
each state (separated by spaces in most shells). Each state is specified\n
as a coordinate tuple, with an optional label prepended, as in\n ``bound:1.0`` or
``unbound:(2.5,2.5)``. Unlabeled states are named\n ``stateN``, where N is the
(zero-based) position in the list of states\n supplied to --states.\n\n2. States
corresponding to multiple bins may use a YAML input file specified\n
with --states-from-file. This file defines a list of states, each with a\n
name and a list of coordinate tuples; bins containing these coordinates\n
will be mapped to the containing state. For instance, the following\n
file::\n\n ---\n states:\n - label:
unbound\n coords:\n - [9.0, 1.0]\n - [9.0, 2.0]\n - label: bound\n coords:\n -
[0.1, 0.0]\n\n produces two macrostates: the first state is called "unbound" and\n
consists of bins containing the (2-dimensional) progress coordinate\n
values (9.0, 1.0) and (9.0, 2.0); the second state is called "bound"\n
and consists of the single bin containing the point (0.1, 0.0).\n\n3. Arbitrary state definitions may be
supplied by a user-defined function,\n
specified as --states-from-function=MODULE.FUNCTION. This function is\n
called with the bin mapper as an argument (``function(mapper)``) and must\n
return a list of dictionaries, one per state. Each dictionary must contain\n
a vector of coordinate tuples with key "coords"; the bins into which each\n
of these tuples falls define the state. An optional name for the state\n
(with key "label") may also be provided.
\n\n\
n-----\
nOutput
format\
n-----\n
\nThe output file (-o/--output, by default "assign.h5") contains the

```

add_args(*parser*)

Add arguments specific to this tool to the given argparse parser.

process_args(*args*)

Take argparse-processed arguments associated with this tool and deal with them appropriately (setting instance variables, etc)

parse_cmdline_states(*state_strings*)

load_config_from_west(*scheme*)

load_state_file(*state_filename*)

states_from_dict(*ystates*)

load_states_from_function(*statefunc*)

assign_iteration(*n_iter, nstates, nbins, state_map, last_labels*)

Method to encapsulate the segment slicing (into *n_worker* slices) and parallel job submission Submits job(s), waits on completion, splices them back together Returns: assignments, trajlabels, pops for this iteration

go()

Perform the analysis associated with this tool.

`westpa.cli.tools.w_assign.entry_point()`

6.1.7 w_trace

usage:

```
w_trace [-h] [-r RCFILE] [--quiet | --verbose | --debug] [--version] [-W WEST_H5FILE]
        [-d DSNAME] [--output-pattern OUTPUT_PATTERN] [-o OUTPUT]
        N_ITER:SEG_ID [N_ITER:SEG_ID ...]
```

Trace individual WEST trajectories and emit (or calculate) quantities along the trajectory.

Trajectories are specified as *N_ITER:SEG_ID* pairs. Each segment is traced back to its initial point, and then various quantities (notably *n_iter* and *seg_id*) are printed in order from initial point up until the given segment in the given iteration.

Output is stored in several files, all named according to the pattern given by the *-o/--output-pattern* parameter. The default output pattern is “traj_*%d_%d*”, where the printf-style format codes are replaced by the iteration number and segment ID of the terminal segment of the trajectory being traced.

Individual datasets can be selected for writing using the *-d/--dataset* option (which may be specified more than once). The simplest form is *-d dsname*, which causes data from dataset *dsname* along the trace to be stored to HDF5. The dataset is assumed to be stored on a per-iteration basis, with the first dimension corresponding to *seg_id* and the second dimension corresponding to time within the segment. Further options are specified as comma-separated key=value pairs after the data set name, as in:

```
-d dsname,alias=newname,index=idsname,file=otherfile.h5,slice=[100,...]
```

The following options for datasets are supported:

`alias=newname`

When writing this data to HDF5 or text files, use `newname` instead of `dsname` to identify the dataset. This is mostly of use in conjunction with the `slice` option in order, e.g., to retrieve two different slices of a dataset and store them with different names for future use.

`index=idsname`

The dataset is not stored on a per-iteration basis for all segments, but instead is stored as a single dataset whose first dimension indexes `n_iter/seg_id` pairs. The index to these `n_iter/seg_id` pairs is `idsname`.

`file=otherfile.h5`

Instead of reading data from the main WEST HDF5 file (usually `west.h5`), read data from `otherfile.h5`.

`slice=[100,...]`

Retrieve only the given slice from the dataset. This can be used to pick a subset of interest to minimize I/O.

6.1.7.1 positional arguments

<code>N_ITER:SEG_ID</code>	Trace trajectory ending (or at least alive at) <code>N_ITER:SEG_ID</code> .
----------------------------	---

6.1.7.2 optional arguments

<code>-h, --help</code>	show this help message and exit
<code>-d DSNAME, --dataset DSNAME</code>	Include the dataset named <code>DSNAME</code> in trace output. An extended form
<code>↳ like</code>	<code>DSNAME[,alias=ALIAS][,index=INDEX][,file=FILE][,slice=SLICE]</code> will
<code>↳ obtain the</code>	dataset from the given <code>FILE</code> instead of the main WEST HDF5 file,
<code>↳ slice it by</code>	<code>SLICE</code> , call it <code>ALIAS</code> in output, and/or access per-segment data by a
<code>↳ group for</code>	<code>n_iter,seg_id INDEX</code> instead of a <code>seg_id</code> indexed dataset in the
	<code>n_iter</code> .

6.1.7.3 general options

```
-r RCFILE, --rcfile RCFILE
                        use RCFILE as the WEST run-time configuration file (default: west.
↳cfg)
--quiet                emit only essential information
--verbose              emit extra information
--debug                enable extra checks and emit copious information
--version              show program's version number and exit
```

6.1.7.4 WEST input data options

```
-W WEST_H5FILE, --west-data WEST_H5FILE
                        Take WEST data from WEST_H5FILE (default: read from the HDF5 file,
↳specified in
                        west.cfg).
```

6.1.7.5 output options

```
--output-pattern OUTPUT_PATTERN
                        Write per-trajectory data to output files/HDF5 groups whose names,
↳begin with
                        OUTPUT_PATTERN, which must contain two printf-style format flags,
↳which will be
                        replaced with the iteration number and segment ID of the terminal,
↳segment of
                        the trajectory being traced. (Default: traj_%d_%d.)
-o OUTPUT, --output OUTPUT
                        Store intermediate data and analysis results to OUTPUT (default:
↳trajs.h5).
```

6.1.7.6 westpa.cli.tools.w_trace module

class westpa.cli.tools.w_trace.WESTTool

Bases: *WESTToolComponent*

Base class for WEST command line tools

prog = None

usage = None

description = None

epilog = None

add_args(*parser*)

Add arguments specific to this tool to the given argparse parser.

process_args(args)

Take argparse-processed arguments associated with this tool and deal with them appropriately (setting instance variables, etc)

make_parser(prog=None, usage=None, description=None, epilog=None, args=None)

make_parser_and_process(prog=None, usage=None, description=None, epilog=None, args=None)

A convenience function to create a parser, call `add_all_args()`, and then call `process_all_args()`. The argument namespace is returned.

go()

Perform the analysis associated with this tool.

main()

A convenience function to make a parser, parse and process arguments, then call `self.go()`

class westpa.cli.tools.w_trace.**WESTDataReader**

Bases: [*WESTToolComponent*](#)

Tool for reading data from WEST-related HDF5 files. Coordinates finding the main HDF5 file from `west.cfg` or command line arguments, caching of certain kinds of data (eventually), and retrieving auxiliary data sets from various places.

add_args(parser)

Add arguments specific to this component to the given argparse parser.

process_args(args)

Take argparse-processed arguments associated with this component and deal with them appropriately (setting instance variables, etc)

open(mode='r')

close()

property weight_dsspec

property parent_id_dsspec

class westpa.cli.tools.w_trace.**Segment**(n_iter=None, seg_id=None, weight=None, endpoint_type=None, parent_id=None, wtg_parent_ids=None, pcoord=None, status=None, walltime=None, cputime=None, data=None)

Bases: object

A class wrapping segment data that must be passed through the work manager or data manager. Most fields are self-explanatory. One item worth noting is that a negative parent ID means that the segment starts from the initial state with ID `-(segment.parent_id+1)`

SEG_STATUS_UNSET = 0

SEG_STATUS_PREPARED = 1

SEG_STATUS_COMPLETE = 2

SEG_STATUS_FAILED = 3

SEG_INITPOINT_UNSET = 0

SEG_INITPOINT_CONTINUES = 1

```

SEG_INITPOINT_NEWTRAJ = 2

SEG_ENDPOINT_UNSET = 0

SEG_ENDPOINT_CONTINUES = 1

SEG_ENDPOINT_MERGED = 2

SEG_ENDPOINT_RECYCLED = 3

statuses = {'SEG_STATUS_COMPLETE': 2, 'SEG_STATUS_FAILED': 3, 'SEG_STATUS_PREPARED':
1, 'SEG_STATUS_UNSET': 0}

initpoint_types = {'SEG_INITPOINT_CONTINUES': 1, 'SEG_INITPOINT_NEWTRAJ': 2,
'SEG_INITPOINT_UNSET': 0}

endpoint_types = {'SEG_ENDPOINT_CONTINUES': 1, 'SEG_ENDPOINT_MERGED': 2,
'SEG_ENDPOINT_RECYCLED': 3, 'SEG_ENDPOINT_UNSET': 0}

status_names = {0: 'SEG_STATUS_UNSET', 1: 'SEG_STATUS_PREPARED', 2:
'SEG_STATUS_COMPLETE', 3: 'SEG_STATUS_FAILED'}

initpoint_type_names = {0: 'SEG_INITPOINT_UNSET', 1: 'SEG_INITPOINT_CONTINUES', 2:
'SEG_INITPOINT_NEWTRAJ'}

endpoint_type_names = {0: 'SEG_ENDPOINT_UNSET', 1: 'SEG_ENDPOINT_CONTINUES', 2:
'SEG_ENDPOINT_MERGED', 3: 'SEG_ENDPOINT_RECYCLED'}

static initial_pcoord(segment)
    Return the initial progress coordinate point of this segment.

static final_pcoord(segment)
    Return the final progress coordinate point of this segment.

property initpoint_type

property initial_state_id

property status_text

property endpoint_type_text

class westpa.cli.tools.w_trace.InitialState(state_id, basis_state_id, iter_created, iter_used=None,
istate_type=None, istate_status=None, pcoord=None,
basis_state=None, basis_auxref=None)

```

Bases: object

Describes an initial state for a new trajectory. These are generally constructed by appropriate modification of a basis state.

Variables

- **state_id** – Integer identifier of this state, usually set by the data manager.
- **basis_state_id** – Identifier of the basis state from which this state was generated, or None.
- **basis_state** – The *BasisState* from which this state was generated, or None.
- **iter_created** – Iteration in which this state was generated (0 for simulation initialization).
- **iter_used** – Iteration in which this state was used to initiate a trajectory (None for unused).

- **istate_type** – Integer describing the type of this initial state (ISTATE_TYPE_BASIS for direct use of a basis state, ISTATE_TYPE_GENERATED for a state generated from a basis state, ISTATE_TYPE_RESTART for a state corresponding to the endpoint of a segment in another simulation, or ISTATE_TYPE_START for a state generated from a start state).
- **istate_status** – Integer describing whether this initial state has been properly prepared.
- **pcoord** – The representative progress coordinate of this state.

```
ISTATE_TYPE_UNSET = 0
```

```
ISTATE_TYPE_BASIS = 1
```

```
ISTATE_TYPE_GENERATED = 2
```

```
ISTATE_TYPE_RESTART = 3
```

```
ISTATE_TYPE_START = 4
```

```
ISTATE_UNUSED = 0
```

```
ISTATE_STATUS_PENDING = 0
```

```
ISTATE_STATUS_PREPARED = 1
```

```
ISTATE_STATUS_FAILED = 2
```

```
istate_types = {'ISTATE_TYPE_BASIS': 1, 'ISTATE_TYPE_GENERATED': 2,  
'ISTATE_TYPE_RESTART': 3, 'ISTATE_TYPE_START': 4, 'ISTATE_TYPE_UNSET': 0}
```

```
istate_type_names = {0: 'ISTATE_TYPE_UNSET', 1: 'ISTATE_TYPE_BASIS', 2:  
'ISTATE_TYPE_GENERATED', 3: 'ISTATE_TYPE_RESTART', 4: 'ISTATE_TYPE_START'}
```

```
istate_statuses = {'ISTATE_STATUS_FAILED': 2, 'ISTATE_STATUS_PENDING': 0,  
'ISTATE_STATUS_PREPARED': 1}
```

```
istate_status_names = {0: 'ISTATE_STATUS_PENDING', 1: 'ISTATE_STATUS_PREPARED', 2:  
'ISTATE_STATUS_FAILED'}
```

```
as_numpy_record()
```

```
westpa.cli.tools.w_trace.weight_dtype
```

```
alias of float64
```

```
westpa.cli.tools.w_trace.n_iter_dtype
```

```
alias of uint32
```

```
westpa.cli.tools.w_trace.seg_id_dtype
```

```
alias of int64
```

```
westpa.cli.tools.w_trace.ptime_dtype
```

```
alias of float64
```

```
class westpa.cli.tools.w_trace.Trace(summary, endpoint_type, basis_state, initial_state,  
data_manager=None)
```

```
Bases: object
```

A class representing a trace of a certain trajectory segment back to its origin.

classmethod `from_data_manager(n_iter, seg_id, data_manager=None)`

Construct and return a trajectory trace whose last segment is identified by `seg_id` in the iteration number `n_iter`.

get_segment_data_slice(*datafile, dsname, n_iter, seg_id, slice_=None, index_data=None, iter_prec=None*)

Return the data from the dataset named `dsname` within the given `datafile` (an open `h5py.File` object) for the given iteration and segment. By default, it is assumed that the dataset is stored in the iteration group for iteration `n_iter`, but if `index_data` is provided, it must be an iterable (preferably a simple array) of (`n_iter`,`seg_id`) pairs, and the index in the `index_data` iterable of the matching `n_iter/seg_id` pair is used as the index of the data to retrieve.

If an optional `slice_` is provided, then the given slicing tuple is appended to that used to retrieve the segment-specific data (i.e. it can be used to pluck a subset of the data that would otherwise be returned).

trace_timepoint_dataset(*dsname, slice_=None, auxfile=None, index_ds=None*)

Return a trace along this trajectory over a dataset which is layed out as [`seg_id`][`timepoint`][...]. Overlapping values at segment boundaries are accounted for. Returns (`data_trace`, `weight`), where `data_trace` is a time series of the dataset along this trajectory, and `weight` is the corresponding trajectory weight at each time point.

If `auxfile` is given, then load the dataset from the given HDF5 file, which must be layed out the same way as the main HDF5 file (e.g. iterations arranged as `iterations/iter_*`).

If `index_ds` is given, instead of reading data per-iteration from `iter_*` groups, then the given `index_ds` is used as an index of `n_iter,seg_id` pairs into `dsname`. In this case, the target data set need not exist on a per-iteration basis inside `iter_*` groups.

If `slice_` is given, then *further* slice the data returned from the HDF5 dataset. This can minimize I/O if it is known (and specified) that only a subset of the data along the trajectory is needed.

class `westpa.cli.tools.w_trace.WTraceTool`

Bases: [`WESTTool`](#)

`prog = 'w_trace'`

description = 'Trace individual WEST trajectories and emit (or calculate) quantities along the trajectory. Trajectories are specified as N_ITER:SEG_ID pairs. Each segment is traced back to its initial point, and then various quantities (notably n_iter and seg_id) are printed in order from initial point up until the given segment in the given iteration. Output is stored in several files, all named according to the pattern given by the -o/--output-pattern parameter. The default output pattern is "traj_%d_%d", where the printf-style format codes are replaced by the iteration number and segment ID of the terminal segment of the trajectory being traced. Individual datasets can be selected for writing using the -d/--dataset option (which may be specified more than once). The simplest form is ``-d dsname``, which causes data from dataset ``dsname`` along the trace to be stored to HDF5. The dataset is assumed to be stored on a per-iteration basis, with the first dimension corresponding to seg_id and the second dimension corresponding to time within the segment. Further options are specified as comma-separated key=value pairs after the data set name, as in\n\n -d dsname,alias=newname,index=idsname,file=otherfile.h5,slice=[100,...]\n\nThe following options for datasets are supported:\n\n alias=newname\n When writing this data to HDF5 or text files, use ``newname`` instead of ``dsname`` to identify the dataset. This is mostly of use in conjunction with the ``slice`` option in order, e.g., to retrieve two different slices of a dataset and store them with different names for future use.\n\n index=idsname\n The dataset is not stored on a per-iteration basis for all segments, but instead is stored as a single dataset whose first dimension indexes n_iter/seg_id pairs. The index to these n_iter/seg_id pairs is ``idsname``.\n\n file=otherfile.h5\n Instead of reading data from the main WEST HDF5 file (usually ``west.h5``), read data from ``otherfile.h5``.\n\n slice=[100,...]\n Retrieve only the given slice from the dataset. This can be used to pick a subset of interest to minimize I/O.\n\n-----\n'

```
pcoord_formats = {'f4': '%14.7g', 'f8': '%023.15g', 'i2': '%6d', 'i4': '%11d', 'i8': '%20d', 'u2': '%5d', 'u4': '%10d', 'u8': '%20d'}
```

add_args(*parser*)

Add arguments specific to this tool to the given argparse parser.

process_args(*args*)

Take argparse-processed arguments associated with this tool and deal with them appropriately (setting instance variables, etc)

parse_dataset_string(*dsstr*)

go()

Perform the analysis associated with this tool.

emit_trace_h5(*trace*, *output_group*)

emit_trace_text(*trace*, *output_file*)

Dump summary information about each segment in the given trace to the given output_file, which must be opened for writing in text mode. Output columns are separated by at least one space.

westpa.cli.tools.w_trace.entry_point()

6.1.8 w_ipa

The `w_ipa` is a (beta) WESTPA tool that automates analysis using analysis schemes and enables interactive analysis of WESTPA simulation data. The tool can do a variety of different types of analysis, including the following: * Calculate fluxes and rate constants * Adjust and use alternate state definitions * Trace trajectory segments, including statistical weights, position along the progress coordinate, and other auxiliary data * Plot all of the above in the terminal!

If you are using `w_ipa` for kinetics automated kinetics analysis, keep in mind that `w_ipa` is running `w_assign` and `w_direct` using the scheme designated in your `west.cfg` file. For more diverse kinetics analysis options, consider using `w_assign` and `w_direct` manually. This can be useful if you'd like to use auxiliary coordinates that aren't your progress coordinate, in one or two dimension options.

usage:

```
w_ipa [-h] [-r RCFILE] [--quiet] [--verbose] [--version] [--max-queue-length MAX_QUEUE_
↳LENGTH]
        [-W WEST_H5FILE] [--analysis-only] [--reanalyze] [--ignore-hash] [--debug] [-
↳-terminal]
        [--serial | --parallel | --work-manager WORK_MANAGER] [--n-workers N_WORKERS]
        [--zmq-mode MODE] [--zmq-comm-mode COMM_MODE] [--zmq-write-host-info INFO_
↳FILE]
        [--zmq-read-host-info INFO_FILE] [--zmq-upstream-rr-endpoint ENDPOINT]
        [--zmq-upstream-ann-endpoint ENDPOINT] [--zmq-downstream-rr-endpoint_
↳ENDPOINT]
        [--zmq-downstream-ann-endpoint ENDPOINT] [--zmq-master-heartbeat MASTER_
↳HEARTBEAT]
        [--zmq-worker-heartbeat WORKER_HEARTBEAT] [--zmq-timeout-factor FACTOR]
        [--zmq-startup-timeout STARTUP_TIMEOUT] [--zmq-shutdown-timeout SHUTDOWN_
↳TIMEOUT]
```

optional arguments:

```
-h, --help          show this help message and exit
```

general options:

-r RCFILE, --rcfile RCFILE	use RCFILE as the WEST run-time configuration file (default: west.cfg)
--quiet	emit only essential information
--verbose	emit extra information
--version	show program's version number and exit

parallelization options:

```
--max-queue-length MAX_QUEUE_LENGTH
        Maximum number of tasks that can be queued. Useful to limit RAM_
↳use for tasks that
        have very large requests/response. Default: no limit.
```

WEST input data options:

-W WEST_H5FILE, --west-data WEST_H5FILE	Take WEST data from WEST_H5FILE (default: read from the HDF5 file specified in west.cfg).
--	---

runtime options:

```
--analysis-only, -ao Use this flag to run the analysis and return to the terminal.
--reanalyze, -ra Use this flag to delete the existing files and reanalyze.
--ignore-hash, -ih Ignore hash and don't regenerate files.
--debug, -d Debug output largely intended for development.
--terminal, -t Plot output in terminal.
```

parallelization options:

```
--serial run in serial mode
--parallel run in parallel mode (using processes)
--work-manager WORK_MANAGER use the given work manager for parallel task distribution.
↳ Available work managers are ('serial', 'threads', 'processes', 'zmq'); default is
↳ 'processes'
--n-workers N_WORKERS Use up to N_WORKERS on this host, for work managers which support
↳ this option. Use 0 for a dedicated server. (Ignored by work managers which do not
↳ support this option.)
```

options for ZeroMQ (“zmq”) work manager (master or node):

```
--zmq-mode MODE Operate as a master (server) or a node (workers/client). "server"
↳ is a deprecated synonym for "master" and "client" is a deprecated synonym for "node"
↳ ".
--zmq-comm-mode COMM_MODE Use the given communication mode -- TCP or IPC (Unix-domain) --
↳ sockets for communication within a node. IPC (the default) may be more
↳ efficient but is not available on (exceptionally rare) systems without node-local
↳ storage (e.g. /tmp); on such systems, TCP may be used instead.
--zmq-write-host-info INFO_FILE Store hostname and port information needed to connect to this
↳ instance in INFO_FILE. This allows the master and nodes assisting in
↳ coordinating the communication of other nodes to choose ports randomly. Downstream
↳ nodes read this file with --zmq-read-host-info and know where how to connect.
--zmq-read-host-info INFO_FILE Read hostname and port information needed to connect to the master
↳ (or other coordinating node) from INFO_FILE. This allows the master and
↳ nodes assisting in coordinating the communication of other nodes to choose ports
↳ randomly, writing that information with --zmq-write-host-info for this instance to
↳ read.
```

(continues on next page)

(continued from previous page)

```

--zmq-upstream-rr-endpoint ENDPOINT
    ZeroMQ endpoint to which to send request/response (task and
↳result) traffic toward
        the master.
--zmq-upstream-ann-endpoint ENDPOINT
    ZeroMQ endpoint on which to receive announcement (heartbeat and
↳shutdown
        notification) traffic from the master.
--zmq-downstream-rr-endpoint ENDPOINT
    ZeroMQ endpoint on which to listen for request/response (task and
↳result) traffic
        from subsidiary workers.
--zmq-downstream-ann-endpoint ENDPOINT
    ZeroMQ endpoint on which to send announcement (heartbeat and
↳shutdown
        notification) traffic toward workers.
--zmq-master-heartbeat MASTER_HEARTBEAT
    Every MASTER_HEARTBEAT seconds, the master announces its presence
↳to workers.
--zmq-worker-heartbeat WORKER_HEARTBEAT
    Every WORKER_HEARTBEAT seconds, workers announce their presence to
↳the master.
--zmq-timeout-factor FACTOR
    Scaling factor for heartbeat timeouts. If the master doesn't hear
↳from a worker in
        WORKER_HEARTBEAT*FACTOR, the worker is assumed to have crashed. If
↳a worker
        doesn't hear from the master in MASTER_HEARTBEAT*FACTOR seconds,
↳the master is
        assumed to have crashed. Both cases result in shutdown.
--zmq-startup-timeout STARTUP_TIMEOUT
    Amount of time (in seconds) to wait for communication between the
↳master and at
        least one worker. This may need to be changed on very large,
↳heavily-loaded
        computer systems that start all processes simultaneously.
--zmq-shutdown-timeout SHUTDOWN_TIMEOUT
    Amount of time (in seconds) to wait for workers to shut down.

```

6.1.8.1 westpa.cli.tools.w_ipa module

class westpa.cli.tools.w_ipa.WESTParallelTool(wm_env=None)

Bases: *WESTTool*

Base class for command-line tools parallelized with wwmgr. This automatically adds and processes wwmgr command-line arguments and creates a work manager at self.work_manager.

make_parser_and_process(prog=None, usage=None, description=None, epilog=None, args=None)

A convenience function to create a parser, call add_all_args(), and then call process_all_args(). The argument namespace is returned.

add_args(parser)

Add arguments specific to this tool to the given argparse parser.

process_args(args)

Take argparse-processed arguments associated with this tool and deal with them appropriately (setting instance variables, etc)

go()

Perform the analysis associated with this tool.

main()

A convenience function to make a parser, parse and process arguments, then run self.go() in the master process.

class westpa.cli.tools.w_ipa.WESTDataReader

Bases: [WESTToolComponent](#)

Tool for reading data from WEST-related HDF5 files. Coordinates finding the main HDF5 file from west.cfg or command line arguments, caching of certain kinds of data (eventually), and retrieving auxiliary data sets from various places.

add_args(parser)

Add arguments specific to this component to the given argparse parser.

process_args(args)

Take argparse-processed arguments associated with this component and deal with them appropriately (setting instance variables, etc)

open(mode='r')

close()

property weight_dsspec

property parent_id_dsspec

class westpa.cli.tools.w_ipa.ProgressIndicatorComponent

Bases: [WESTToolComponent](#)

add_args(parser)

Add arguments specific to this component to the given argparse parser.

process_args(args)

Take argparse-processed arguments associated with this component and deal with them appropriately (setting instance variables, etc)

class westpa.cli.tools.w_ipa.Plotter(h5file, h5key, iteration=-1, interface='matplotlib')

Bases: object

This is a semi-generic plotting interface that has a built in curses based terminal plotter. It's fairly specific to what we're using it for here, but we could (and maybe should) build it out into a little library that we can use via the command line to plot things. Might be useful for looking at data later. That would also cut the size of this tool down by a good bit.

plot(i=0, j=1, tau=1, iteration=None, dim=0, interface=None)

class westpa.cli.tools.w_ipa.WIPIDataset(raw, key)

Bases: object

keys()

class westpa.cli.tools.w_ipa.WIPIScheme(*scheme, name, parent, settings*)

Bases: object

property scheme

property list_schemes

Lists what schemes are configured in west.cfg file. Schemes should be structured as follows, in west.cfg:

west:

system:

analysis:

directory: analysis analysis_schemes:

scheme.1:

enabled: True states:

- label: unbound coords: [[7.0]]
- label: bound coords: [[2.7]]

bins:

- type: RectilinearBinMapper boundaries: [[0.0, 2.80, 7, 10000]]

property iteration

property assign

property direct

The output from w_direct.py from the current scheme.

property state_labels

property bin_labels

property west

property reweight

property current

The current iteration. See help for __get_data_for_iteration__

property past

The previous iteration. See help for __get_data_for_iteration__

class westpa.cli.tools.w_ipa.WIPI

Bases: [WESTParallelTool](#)

Welcome to w_ipa (WESTPA Interactive Python Analysis)! From here, you can run traces, look at weights, progress coordinates, etc. This is considered a ‘stateful’ tool; that is, the data you are pulling is always pulled from the current analysis scheme and iteration. By default, the first analysis scheme in west.cfg is used, and you are set at iteration 1.

ALL PROPERTIES ARE ACCESSED VIA w or west To see the current iteration, try:

w.iteration OR west.iteration

to set it, simply plug in a new value.

w.iteration = 100

To change/list the current analysis schemes:

w.list_schemes w.scheme = OUTPUT FROM w.list_schemes

To see the states and bins defined in the current analysis scheme:

```
w.states w.bin_labels
```

All information about the current iteration is available in an object called ‘current’:

```
w.current walkers, summary, states, seg_id, weights, parents, kinavg, pcoord, bins, populations, and  
auxdata, if it exists.
```

In addition, the function `w.trace(seg_id)` will run a trace over a `seg_id` in the current iteration and return a dictionary containing all pertinent information about that `seg_id`’s history. It’s best to store this, as the trace can be expensive.

Run `help` on any function or property for more information!

Happy analyzing!

add_args(*parser*)

Add arguments specific to this tool to the given argparse parser.

process_args(*args*)

Take argparse-processed arguments associated with this tool and deal with them appropriately (setting instance variables, etc)

hash_args(*args*, *extra=None*, *path=None*)

Create unique hash stamp to determine if arguments/file is different from before.

stamp_hash(*h5file_name*, *new_hash*)

Loads a file, stamps it, and returns the opened file in read only

analysis_structure()

Run automatically on startup. Parses through the configuration file, and loads up all the data files from the different analysis schematics. If they don’t exist, it creates them automatically by hooking in to existing analysis routines and going from there.

It does this by calling in the `make_parser_and_process` function for `w_{assign,reweight,direct}` using a custom built list of args. The user can specify everything in the configuration file that would have been specified on the command line.

For instance, were one to call `w_direct` as follows:

```
w_direct -evolution cumulative -step-iter 1 -disable-correl
```

the `west.cfg` would look as follows:

west:

analysis:

w_direct:

```
evolution: cumulative step_iter: 1 extra: ['disable-correl']
```

Alternatively, if one wishes to use the same options for both `w_direct` and `w_reweight`, the key ‘`w_direct`’ can be replaced with ‘`kinetics`’.

property assign

property direct

The output from `w_kinavg.py` from the current scheme.

property state_labels

property bin_labels

property west

property reweight

property scheme

Returns and sets what scheme is currently in use. To see what schemes are available, run:
`w.list_schemes`

property list_schemes

Lists what schemes are configured in west.cfg file. Schemes should be structured as follows, in west.cfg:
west:

system:**analysis:**

directory: analysis analysis_schemes:

scheme.1:

enabled: True states:

- label: unbound coords: [[7.0]]
- label: bound coords: [[2.7]]

bins:

- type: RectilinearBinMapper boundaries: [[0.0, 2.80, 7, 10000]]

property iteration

Returns/sets the current iteration.

property current

The current iteration. See help for `__get_data_for_iteration__`

property past

The previous iteration. See help for `__get_data_for_iteration__`

trace(seg_id)

Runs a trace on a seg_id within the current iteration, all the way back to the beginning, returning a dictionary containing all interesting information:

seg_id, pcoord, states, bins, weights, iteration, auxdata (optional)
sorted in chronological order.

Call with a seg_id.

property future

Similar to current/past, but keyed differently and returns different datasets. See help for Future.

class Future(raw, key)

Bases: [WIPIDataset](#)

go()

Function automatically called by main() when launched via the command line interface. Generally, call main, not this function.

property introduction

Just spits out an introduction, in case someone doesn't call help.

property help

Just a minor function to call help on itself. Only in here to really help someone get help.

`westpa.cli.tools.w_ipa.entry_point()`

6.1.9 w_pdist

w_pdist constructs and calculates the progress coordinate probability distribution's evolution over a user-specified number of simulation iterations. w_pdist supports progress coordinates with dimensionality 1.

The resulting distribution can be viewed with the *plothist* tool.

6.1.9.1 Overview

Usage:

```
w_pdist [-h] [-r RCFILE] [--quiet | --verbose | --debug] [--version]
        [-W WEST_H5FILE] [--first-iter N_ITER] [--last-iter N_ITER]
        [-b BINEXPR] [-o OUTPUT]
        [--construct-dataset CONSTRUCT_DATASET | --
↪dsspecs DSSPEC [DSSPEC ...]]
        [--serial | --parallel | --work-manager WORK_MANAGER]
        [--n-workers N_WORKERS] [--zmq-mode MODE]
        [--zmq-info INFO_FILE] [--zmq-task-endpoint TASK_ENDPOINT]
        [--zmq-result-endpoint RESULT_ENDPOINT]
        [--zmq-announce-endpoint ANNOUNCE_ENDPOINT]
        [--zmq-listen-endpoint ANNOUNCE_ENDPOINT]
        [--zmq-heartbeat-interval INTERVAL]
        [--zmq-task-timeout TIMEOUT] [--zmq-client-comm-mode MODE]
```

Note: This tool supports parallelization, which may be more efficient for especially large datasets.

6.1.9.2 Command-Line Options

See the [general command-line tool reference](#) for more information on the general options.

6.1.9.2.1 Input/output options

These arguments allow the user to specify where to read input simulation result data and where to output calculated progress coordinate probability distribution data.

Both input and output files are *hdf5* format:

```
-W, --WEST_H5FILE file
  Read simulation result data from file *file*. (**Default:** The
  *hdf5* file specified in the configuration file (default config file
  is *west.h5*))

-o, --output file
  Store this tool's output in *file*. (**Default:** The *hdf5* file
  **pcpdist.h5**)
```

6.1.9.2.2 Iteration range options

Specify the range of iterations over which to construct the progress coordinate probability distribution.:

```
--first-iter n_iter
Construct probability distribution starting with iteration *n_iter*
(**Default:** 1)

--last-iter n_iter
Construct probability distribution's time evolution up to (and
including) iteration *n_iter* (**Default:** Last completed
iteration)
```

6.1.9.2.3 Probability distribution binning options

Specify the number of bins to use when constructing the progress coordinate probability distribution. If using a multidimensional progress coordinate, different binning schemes can be used for the probability distribution for each progress coordinate.:

```
-b binexpr
*binexpr* specifies the number and formatting of the bins. Its
format can be as follows:

    1. an integer, in which case all distributions have that many
    equal sized bins
    2. a python-style list of integers, of length corresponding to
    the number of dimensions of the progress coordinate, in which
    case each progress coordinate's probability distribution has the
    corresponding number of bins
    3. a python-style list of lists of scalars, where the list at
    each index corresponds to each dimension of the progress
    coordinate and specifies specific bin boundaries for that
    progress coordinate's probability distribution.

(**Default:** 100 bins for all progress coordinates)
```

6.1.9.3 Examples

Assuming simulation results are stored in *west.h5* (which is specified in the configuration file named *west.cfg*), for a simulation with a 1-dimensional progress coordinate:

Calculate a probability distribution histogram using all default options (output file: *pdist.h5*; histogram binning: 100 equal sized bins; probability distribution over the lowest reached progress coordinate to the largest; work is parallelized over all available local cores using the ‘processes’ work manager):

```
w_pdist
```

Same as above, except using the serial work manager (which may be more efficient for smaller datasets):

```
w_pdist --serial
```

6.1.9.4 westpa.cli.tools.w_pdist module

class westpa.cli.tools.w_pdist.WESTParallelTool(*w_m_env=None*)

Bases: [WESTTool](#)

Base class for command-line tools parallelized with wwmgr. This automatically adds and processes wwmgr command-line arguments and creates a work manager at self.work_manager.

make_parser_and_process(*prog=None, usage=None, description=None, epilog=None, args=None*)

A convenience function to create a parser, call add_all_args(), and then call process_all_args(). The argument namespace is returned.

add_args(*parser*)

Add arguments specific to this tool to the given argparse parser.

process_args(*args*)

Take argparse-processed arguments associated with this tool and deal with them appropriately (setting instance variables, etc)

go()

Perform the analysis associated with this tool.

main()

A convenience function to make a parser, parse and process arguments, then run self.go() in the master process.

class westpa.cli.tools.w_pdist.WESTDataReader

Bases: [WESTToolComponent](#)

Tool for reading data from WEST-related HDF5 files. Coordinates finding the main HDF5 file from west.cfg or command line arguments, caching of certain kinds of data (eventually), and retrieving auxiliary data sets from various places.

add_args(*parser*)

Add arguments specific to this component to the given argparse parser.

process_args(*args*)

Take argparse-processed arguments associated with this component and deal with them appropriately (setting instance variables, etc)

open(*mode='r'*)

close()

property weight_dsspec

property parent_id_dsspec

class westpa.cli.tools.w_pdist.WESTDSSynthesizer(*default_dsname=None, h5filename=None*)

Bases: [WESTToolComponent](#)

Tool for synthesizing a dataset for analysis from other datasets. This may be done using a custom function, or a list of “data set specifications”. It is anticipated that if several source datasets are required, then a tool will have multiple instances of this class.

group_name = 'input dataset options'

add_args(*parser*)

Add arguments specific to this component to the given argparse parser.

process_args(args)

Take argparse-processed arguments associated with this component and deal with them appropriately (setting instance variables, etc)

class westpa.cli.tools.w_pdist.WESTWDSynthesizer(default_dsname=None, h5filename=None)

Bases: [WESTToolComponent](#)

group_name = 'weight dataset options'

add_args(parser)

Add arguments specific to this component to the given argparse parser.

process_args(args)

Take argparse-processed arguments associated with this component and deal with them appropriately (setting instance variables, etc)

class westpa.cli.tools.w_pdist.IterRangeSelection(data_manager=None)

Bases: [WESTToolComponent](#)

Select and record limits on iterations used in analysis and/or reporting. This class provides both the user-facing command-line options and parsing, and the application-side API for recording limits in HDF5.

HDF5 datasets calculated based on a restricted set of iterations should be tagged with the following attributes:

first_iter

The first iteration included in the calculation.

last_iter

One past the last iteration included in the calculation.

iter_step

Blocking or sampling period for iterations included in the calculation.

add_args(parser)

Add arguments specific to this component to the given argparse parser.

process_args(args, override_iter_start=None, override_iter_stop=None, default_iter_step=1)

Take argparse-processed arguments associated with this component and deal with them appropriately (setting instance variables, etc)

iter_block_iter()

Return an iterable of (block_start,block_end) over the blocks of iterations selected by --first-iter/--last-iter/--step-iter.

n_iter_blocks()

Return the number of blocks of iterations (as returned by iter_block_iter) selected by --first-iter/--last-iter/--step-iter.

record_data_iter_range(h5object, iter_start=None, iter_stop=None)

Store attributes iter_start and iter_stop on the given HDF5 object (group/dataset)

record_data_iter_step(h5object, iter_step=None)

Store attribute iter_step on the given HDF5 object (group/dataset).

check_data_iter_range_least(h5object, iter_start=None, iter_stop=None)

Check that the given HDF5 object contains (as denoted by its iter_start/iter_stop attributes) data at least for the iteration range specified.

check_data_iter_range_equal(h5object, iter_start=None, iter_stop=None)

Check that the given HDF5 object contains (as denoted by its iter_start/iter_stop attributes) data exactly for the iteration range specified.

check_data_iter_step_conformant(*h5object*, *iter_step=None*)

Check that the given HDF5 object contains per-iteration data at an iteration stride suitable for extracting data with the given stride (in other words, the given *iter_step* is a multiple of the stride with which data was recorded).

check_data_iter_step_equal(*h5object*, *iter_step=None*)

Check that the given HDF5 object contains per-iteration data at an iteration stride the same as that specified.

slice_per_iter_data(*dataset*, *iter_start=None*, *iter_stop=None*, *iter_step=None*, *axis=0*)

Return the subset of the given dataset corresponding to the given iteration range and stride. Unless otherwise specified, the first dimension of the dataset is the one sliced.

iter_range(*iter_start=None*, *iter_stop=None*, *iter_step=None*, *dtype=None*)

Return a sequence for the given iteration numbers and stride, filling in missing values from those stored on self. The smallest data type capable of holding *iter_stop* is returned unless otherwise specified using the *dtype* argument.

class westpa.cli.tools.w_pdist.**ProgressIndicatorComponent**

Bases: [*WESTToolComponent*](#)

add_args(*parser*)

Add arguments specific to this component to the given argparse parser.

process_args(*args*)

Take argparse-processed arguments associated with this component and deal with them appropriately (setting instance variables, etc)

westpa.cli.tools.w_pdist.histnd(*values*, *binbounds*, *weights=1.0*, *out=None*, *binbound_check=True*, *ignore_out_of_range=False*)

Generate an N-dimensional PDF (or contribution to a PDF) from the given values. *binbounds* is a list of arrays of boundary values, with one entry for each dimension (*values* must have as many columns as there are entries in *binbounds*) *weight*, if provided, specifies the weight each value contributes to the histogram; this may be a scalar (for equal weights for all values) or a vector of the same length as *values* (for unequal weights). If *binbound_check* is True, then the boundaries are checked for strict positive monotonicity; set to False to shave a few microseconds if you know your bin boundaries to be monotonically increasing.

westpa.cli.tools.w_pdist.normhistnd(*hist*, *binbounds*)

Normalize the N-dimensional histogram *hist* with corresponding bin boundaries *binbounds*. Modifies *hist* in place and returns the normalization factor used.

westpa.cli.tools.w_pdist.isiterable(*x*)

class westpa.cli.tools.w_pdist.**WPDist**

Bases: [*WESTParallelTool*](#)

prog = 'w_pdist'

```

description = 'Calculate time-resolved, multi-dimensional probability distributions
of WE\ndatasets.\n\n
n-----\n
nSource
data\n-----\n
n\nSource data is provided either by a user-specified
function\n(--construct-dataset) or a list of "data set specifications"
(--dsspecs).\nIf neither is provided, the progress coordinate dataset '\pcoord\'
is used.\n\nTo use a custom function to extract or calculate data whose
probability\ndistribution will be calculated, specify the function in standard
Python\nMODULE.FUNCTION syntax as the argument to --construct-dataset. This
function\nwill be called as function(n_iter,iter_group), where n_iter is the
iteration\nwhose data are being considered and iter_group is the corresponding
group\nin the main WEST HDF5 file (west.h5). The function must return data which
can\nbe indexed as [segment][timepoint][dimension].\n\nTo use a list of data set
specifications, specify --dsspecs and then list the\ndesired datasets one-by-one
(space-separated in most shells). These data set\nspecifications are formatted as
NAME[,file=FILENAME,slice=SLICE], which will\nuse the dataset called NAME in the
HDF5 file FILENAME (defaulting to the main\nWEST HDF5 file west.h5), and slice it
with the Python slice expression SLICE\n(as in [0:2] to select the first two
elements of the first axis of the\ndataset). The ``slice`` option is most useful for
selecting one column (or\nmore) from a multi-column dataset, such as arises when
using a progress\ncoordinate of multiple dimensions.\n\n
n-----\n
nHistogram
binning\n
n-----\n\nBy
default, histograms are constructed with 100 bins in each dimension. This\ncan be
overridden by specifying -b/--bins, which accepts a number of different\nkinds of
arguments:\n\n
a single integer N\n
N uniformly spaced bins will be used in each
dimension.\n\n
a sequence of integers N1,N2,... (comma-separated)\n
N1 uniformly
spaced bins will be used for the first dimension, N2 for the\n
second, and so
on.\n\n
a list of lists [[B11, B12, B13, ...], [B21, B22, B23, ...], ...]\n
The bin
boundaries B11, B12, B13, ... will be used for the first dimension,\n
B21, B22, B23,
... for the second dimension, and so on. These bin\n
boundaries need not be
uniformly spaced. These expressions will be\n
evaluated with Python\'s ``eval``
construct, with ``np`` available for\n
use [e.g. to specify bins using
np.arange()].\n\n
The first two forms (integer, list of integers) will trigger a scan
of all\ndata in each dimension in order to determine the minimum and maximum
values,\nwhich may be very expensive for large datasets. This can be avoided
by\nexplicitly providing bin boundaries using the list-of-lists form.\n\n
Note that
these bins are *NOT* at all related to the bins used to drive WE\nsampling.\n\n
n-----\n
nOutput
format\n
n-----\n\n
n\nThe output file produced (specified by -o/--output, defaulting to "pdist.h5")\nmay
be fed to plohist to generate plots (or appropriately processed text or\nHDF5
files) from this data. In short, the following datasets are created:\n\n
``histograms``\n
Normalized histograms. The first axis corresponds to iteration,
and\n
remaining axes correspond to dimensions of the input dataset.\n\n
``/binbounds_0``\n
Vector of bin boundaries for the first (index 0) dimension.
Additional\n
datasets similarly named (/binbounds_1, /binbounds_2, ...) are
created\n
for additional dimensions.\n\n
``/midpoints_0``\n
Vector of bin midpoints
for the first (index 0) dimension. Additional\n
datasets similarly named are created
for additional dimensions.\n\n
``n_iter``\n
Vector of iteration numbers
for additional dimensions.\n\n
6.1. westpacli package
The stored histograms (i.e.\n
the first axis of the ``histograms``55
dataset).\n\n
n-----\n
nSubsequent

```

add_args(*parser*)

Add arguments specific to this tool to the given argparse parser.

process_args(*args*)

Take argparse-processed arguments associated with this tool and deal with them appropriately (setting instance variables, etc)

go()

Perform the analysis associated with this tool.

static parse_binspec(*binspec*)**construct_bins(*bins*)**

Construct bins according to *bins*, which may be:

- 1) A scalar integer (for that number of bins in each dimension)
- 2) A sequence of integers (specifying number of bins for each dimension)
- 3) A sequence of sequences of bin boundaries (specifying boundaries for each dimension)

Sets `self.binbounds` to a list of arrays of bin boundaries appropriate for passing to `fasthist.histnd`, along with `self.midpoints` to the midpoints of the bins.

scan_data_shape()**scan_data_range()**

Scan input data for range in each dimension. The number of dimensions is determined from the shape of the progress coordinate as of `self.iter_start`.

construct_histogram()

Construct a histogram using bins previously constructed with `construct_bins()`. The time series of histogram values is stored in `histograms`. Each histogram in the time series is normalized.

`westpa.cli.tools.w_pdist.entry_point()`

6.1.10 w_succ

usage:

```
w_succ [-h] [-r RCFILE] [--quiet | --verbose | --debug] [--version] [-A H5FILE] [-W WEST_
↪H5FILE]
      [-o OUTPUT_FILE]
```

List segments which successfully reach a target state.

optional arguments:

```
-h, --help            show this help message and exit
-o OUTPUT_FILE, --output OUTPUT_FILE
                        Store output in OUTPUT_FILE (default: write to standard output).
```

general options:

```
-r RCFILE, --rcfile RCFILE
                        use RCFILE as the WEST run-time configuration file (default: west.
↪cfg)
--quiet                emit only essential information
```

(continues on next page)

(continued from previous page)

```
--verbose      emit extra information
--debug        enable extra checks and emit copious information
--version      show program's version number and exit
```

general analysis options:

```
-A H5FILE, --analysis-file H5FILE
                        Store intermediate and final results in H5FILE (default: analysis.
↳ h5).
```

WEST input data options:

```
-W WEST_H5FILE, --west-data WEST_H5FILE
                        Take WEST data from WEST_H5FILE (default: read from the HDF5 file.
↳ specified in
                        west.cfg).
```

6.1.10.1 westpa.cli.core.w_succ module

```
class westpa.cli.core.w_succ.Segment(n_iter=None, seg_id=None, weight=None, endpoint_type=None,
                                     parent_id=None, wtg_parent_ids=None, pcoord=None,
                                     status=None, walltime=None, cputime=None, data=None)
```

Bases: object

A class wrapping segment data that must be passed through the work manager or data manager. Most fields are self-explanatory. One item worth noting is that a negative parent ID means that the segment starts from the initial state with ID -(segment.parent_id+1)

```
SEG_STATUS_UNSET = 0
```

```
SEG_STATUS_PREPARED = 1
```

```
SEG_STATUS_COMPLETE = 2
```

```
SEG_STATUS_FAILED = 3
```

```
SEG_INITPOINT_UNSET = 0
```

```
SEG_INITPOINT_CONTINUES = 1
```

```
SEG_INITPOINT_NEWTRAJ = 2
```

```
SEG_ENDPOINT_UNSET = 0
```

```
SEG_ENDPOINT_CONTINUES = 1
```

```
SEG_ENDPOINT_MERGED = 2
```

```
SEG_ENDPOINT_RECYCLED = 3
```

```
statuses = {'SEG_STATUS_COMPLETE': 2, 'SEG_STATUS_FAILED': 3, 'SEG_STATUS_PREPARED':
1, 'SEG_STATUS_UNSET': 0}
```

```
initpoint_types = {'SEG_INITPOINT_CONTINUES': 1, 'SEG_INITPOINT_NEWTRAJ': 2,
'SEG_INITPOINT_UNSET': 0}
```

```
endpoint_types = {'SEG_ENDPOINT_CONTINUES': 1, 'SEG_ENDPOINT_MERGED': 2,
                  'SEG_ENDPOINT_RECYCLED': 3, 'SEG_ENDPOINT_UNSET': 0}

status_names = {0: 'SEG_STATUS_UNSET', 1: 'SEG_STATUS_PREPARED', 2:
                'SEG_STATUS_COMPLETE', 3: 'SEG_STATUS_FAILED'}

initpoint_type_names = {0: 'SEG_INITPOINT_UNSET', 1: 'SEG_INITPOINT_CONTINUES', 2:
                        'SEG_INITPOINT_NEWTRAJ'}

endpoint_type_names = {0: 'SEG_ENDPOINT_UNSET', 1: 'SEG_ENDPOINT_CONTINUES', 2:
                       'SEG_ENDPOINT_MERGED', 3: 'SEG_ENDPOINT_RECYCLED'}

static initial_pcoord(segment)
    Return the initial progress coordinate point of this segment.

static final_pcoord(segment)
    Return the final progress coordinate point of this segment.

property initpoint_type

property initial_state_id

property status_text

property endpoint_type_text

class westpa.cli.core.w_succ.WESTAnalysisTool
    Bases: object
    add_args(parser, upcall=True)
        Add arguments to a parser common to all analyses of this type.
    process_args(args, upcall=True)
    open_analysis_backing()
    close_analysis_backing()
    require_analysis_group(groupname, replace=False)

class westpa.cli.core.w_succ.WESTDataReaderMixin
    Bases: AnalysisMixin
    A mixin for analysis requiring access to the HDF5 files generated during a WEST run.
    add_args(parser, upcall=True)
    process_args(args, upcall=True)
    clear_run_cache()

    property cache_pcoords
        Whether or not to cache progress coordinate data. While caching this data can significantly speed up some analysis operations, this requires copious RAM.

        Setting this to False when it was formerly True will release any cached data.

    get_summary_table()
```

get_iter_group(*n_iter*)
Return the HDF5 group corresponding to *n_iter*

get_segments(*n_iter*, *include_pcoords=True*)
Return all segments present in iteration *n_iter*

get_segments_by_id(*n_iter*, *seg_ids*, *include_pcoords=True*)
Get segments from the data manager, employing caching where possible

get_children(*segment*, *include_pcoords=True*)

get_seg_index(*n_iter*)

get_wtg_parent_array(*n_iter*)

get_parent_array(*n_iter*)

get_pcoord_array(*n_iter*)

get_pcoord_dataset(*n_iter*)

get_pcoords(*n_iter*, *seg_ids*)

get_seg_ids(*n_iter*, *bool_array=None*)

get_created_seg_ids(*n_iter*)
Return a list of *seg_ids* corresponding to segments which were created for the given iteration (are not continuations).

max_iter_segs_in_range(*first_iter*, *last_iter*)
Return the maximum number of segments present in any iteration in the range selected

total_segs_in_range(*first_iter*, *last_iter*)
Return the total number of segments present in all iterations in the range selected

get_pcoord_len(*n_iter*)
Get the length of the progress coordinate array for the given iteration.

get_total_time(*first_iter=None*, *last_iter=None*, *dt=None*)
Return the total amount of simulation time spanned between *first_iter* and *last_iter* (inclusive).

class westpa.cli.core.w_succ.**CommonOutputMixin**
Bases: *AnalysisMixin*

add_common_output_args(*parser_or_group*)

process_common_output_args(*args*)

class westpa.cli.core.w_succ.**WSucc**
Bases: *CommonOutputMixin*, *WESTDataReaderMixin*, *WESTAnalysisTool*

find_successful_trajs()

westpa.cli.core.w_succ.**entry_point**()

6.1.11 w_crawl

usage:

```
w_crawl [-h] [-r RCFILE] [--quiet | --verbose | --debug] [--version]
        [--max-queue-length MAX_QUEUE_LENGTH] [-W WEST_H5FILE] [--first-iter N_
↪ITER]
        [--last-iter N_ITER] [-c CRAWLER_INSTANCE]
        [--serial | --parallel | --work-manager WORK_MANAGER] [--n-workers N_
↪WORKERS]
        [--zmq-mode MODE] [--zmq-comm-mode COMM_MODE] [--zmq-write-host-info INFO_
↪FILE]
        [--zmq-read-host-info INFO_FILE] [--zmq-upstream-rr-endpoint ENDPOINT]
        [--zmq-upstream-ann-endpoint ENDPOINT] [--zmq-downstream-rr-endpoint_
↪ENDPOINT]
        [--zmq-downstream-ann-endpoint ENDPOINT] [--zmq-master-heartbeat MASTER_
↪HEARTBEAT]
        [--zmq-worker-heartbeat WORKER_HEARTBEAT] [--zmq-timeout-factor FACTOR]
        [--zmq-startup-timeout STARTUP_TIMEOUT] [--zmq-shutdown-timeout SHUTDOWN_
↪TIMEOUT]
        task_callable
```

Crawl a weighted ensemble dataset, executing a function for each iteration. This can be used for postprocessing of trajectories, cleanup of datasets, or anything else that can be expressed as “do X for iteration N, then do something with the result”. Tasks are parallelized by iteration, and no guarantees are made about evaluation order.

6.1.11.1 Command-line options

optional arguments:

```
-h, --help          show this help message and exit
```

general options:

```
-r RCFILE, --rcfile RCFILE
        use RCFILE as the WEST run-time configuration file (default: west.
↪cfg)
--quiet          emit only essential information
--verbose        emit extra information
--debug          enable extra checks and emit copious information
--version        show program's version number and exit
```

parallelization options:

```
--max-queue-length MAX_QUEUE_LENGTH
        Maximum number of tasks that can be queued. Useful to limit RAM_
↪use for tasks
        that have very large requests/response. Default: no limit.
```

WEST input data options:

```
-W WEST_H5FILE, --west-data WEST_H5FILE
        Take WEST data from WEST_H5FILE (default: read from the HDF5 file_
```

(continues on next page)

(continued from previous page)

```
↪specified in
west.cfg).
```

iteration range:

```
--first-iter N_ITER   Begin analysis at iteration N_ITER (default: 1).
--last-iter N_ITER    Conclude analysis with N_ITER, inclusive (default: last completed
↪iteration).
```

task options:

```
-c CRAWLER_INSTANCE, --crawler-instance CRAWLER_INSTANCE
                        Use CRAWLER_INSTANCE (specified as module.instance) as an instance
↪of
                        WESTPACrawler to coordinate the calculation. Required only if
↪initialization,
                        finalization, or task result processing is required.
task_callable          Run TASK_CALLABLE (specified as module.function) on each iteration.
↪ Required.
```

parallelization options:

```
--serial              run in serial mode
--parallel            run in parallel mode (using processes)
--work-manager WORK_MANAGER
                        use the given work manager for parallel task distribution.
↪Available work
                        managers are ('serial', 'threads', 'processes', 'zmq'); default is
↪'serial'
--n-workers N_WORKERS Use up to N_WORKERS on this host, for work managers which support
↪this option.
                        Use 0 for a dedicated server. (Ignored by work managers which do
↪not support
                        this option.)
```

options for ZeroMQ (“zmq”) work manager (master or node):

```
--zmq-mode MODE       Operate as a master (server) or a node (workers/client). "server"
↪is a
                        deprecated synonym for "master" and "client" is a deprecated
↪synonym for
                        "node".
--zmq-comm-mode COMM_MODE
                        Use the given communication mode -- TCP or IPC (Unix-domain) --
↪sockets for
                        communication within a node. IPC (the default) may be more
↪efficient but is not
                        available on (exceptionally rare) systems without node-local
↪storage (e.g.
                        /tmp); on such systems, TCP may be used instead.
--zmq-write-host-info INFO_FILE
                        Store hostname and port information needed to connect to this
```

(continues on next page)

(continued from previous page)

```

↪instance in
INFO_FILE. This allows the master and nodes assisting in
↪coordinating the
communication of other nodes to choose ports randomly. Downstream
↪nodes read
this file with --zmq-read-host-info and know where how to connect.
--zmq-read-host-info INFO_FILE
Read hostname and port information needed to connect to the master
↪(or other
coordinating node) from INFO_FILE. This allows the master and
↪nodes assisting
in coordinating the communication of other nodes to choose ports
↪randomly,
writing that information with --zmq-write-host-info for this
↪instance to read.
--zmq-upstream-rr-endpoint ENDPOINT
ZeroMQ endpoint to which to send request/response (task and
↪result) traffic
toward the master.
--zmq-upstream-ann-endpoint ENDPOINT
ZeroMQ endpoint on which to receive announcement (heartbeat and
↪shutdown
notification) traffic from the master.
--zmq-downstream-rr-endpoint ENDPOINT
ZeroMQ endpoint on which to listen for request/response (task and
↪result)
traffic from subsidiary workers.
--zmq-downstream-ann-endpoint ENDPOINT
ZeroMQ endpoint on which to send announcement (heartbeat and
↪shutdown
notification) traffic toward workers.
--zmq-master-heartbeat MASTER_HEARTBEAT
Every MASTER_HEARTBEAT seconds, the master announces its presence
↪to workers.
--zmq-worker-heartbeat WORKER_HEARTBEAT
Every WORKER_HEARTBEAT seconds, workers announce their presence to
↪the master.
--zmq-timeout-factor FACTOR
Scaling factor for heartbeat timeouts. If the master doesn't hear
↪from a worker
in WORKER_HEARTBEAT*FACTOR, the worker is assumed to have crashed.
↪If a worker
doesn't hear from the master in MASTER_HEARTBEAT*FACTOR seconds,
↪the master is
assumed to have crashed. Both cases result in shutdown.
--zmq-startup-timeout STARTUP_TIMEOUT
Amount of time (in seconds) to wait for communication between the
↪master and at
least one worker. This may need to be changed on very large,
↪heavily-loaded
computer systems that start all processes simultaneously.
--zmq-shutdown-timeout SHUTDOWN_TIMEOUT

```

(continues on next page)

(continued from previous page)

Amount of time (in seconds) to wait for workers to shut down.

6.1.11.2 westpa.cli.tools.w_crawl module

class westpa.cli.tools.w_crawl.WESTParallelTool(*wm_env=None*)

Bases: [WESTTool](#)

Base class for command-line tools parallelized with wwmgr. This automatically adds and processes wwmgr command-line arguments and creates a work manager at self.work_manager.

make_parser_and_process(*prog=None, usage=None, description=None, epilog=None, args=None*)

A convenience function to create a parser, call add_all_args(), and then call process_all_args(). The argument namespace is returned.

add_args(*parser*)

Add arguments specific to this tool to the given argparse parser.

process_args(*args*)

Take argparse-processed arguments associated with this tool and deal with them appropriately (setting instance variables, etc)

go()

Perform the analysis associated with this tool.

main()

A convenience function to make a parser, parse and process arguments, then run self.go() in the master process.

class westpa.cli.tools.w_crawl.WESTDataReader

Bases: [WESTToolComponent](#)

Tool for reading data from WEST-related HDF5 files. Coordinates finding the main HDF5 file from west.cfg or command line arguments, caching of certain kinds of data (eventually), and retrieving auxiliary data sets from various places.

add_args(*parser*)

Add arguments specific to this component to the given argparse parser.

process_args(*args*)

Take argparse-processed arguments associated with this component and deal with them appropriately (setting instance variables, etc)

open(*mode='r'*)

close()

property weight_dsspec

property parent_id_dsspec

class westpa.cli.tools.w_crawl.IterRangeSelection(*data_manager=None*)

Bases: [WESTToolComponent](#)

Select and record limits on iterations used in analysis and/or reporting. This class provides both the user-facing command-line options and parsing, and the application-side API for recording limits in HDF5.

HDF5 datasets calculated based on a restricted set of iterations should be tagged with the following attributes:

first_iter

The first iteration included in the calculation.

last_iter

One past the last iteration included in the calculation.

iter_step

Blocking or sampling period for iterations included in the calculation.

add_args(*parser*)

Add arguments specific to this component to the given argparse parser.

process_args(*args*, *override_iter_start=None*, *override_iter_stop=None*, *default_iter_step=1*)

Take argparse-processed arguments associated with this component and deal with them appropriately (setting instance variables, etc)

iter_block_iter()

Return an iterable of (block_start,block_end) over the blocks of iterations selected by `--first-iter/--last-iter/--step-iter`.

n_iter_blocks()

Return the number of blocks of iterations (as returned by `iter_block_iter`) selected by `--first-iter/--last-iter/--step-iter`.

record_data_iter_range(*h5object*, *iter_start=None*, *iter_stop=None*)

Store attributes `iter_start` and `iter_stop` on the given HDF5 object (group/dataset)

record_data_iter_step(*h5object*, *iter_step=None*)

Store attribute `iter_step` on the given HDF5 object (group/dataset).

check_data_iter_range_least(*h5object*, *iter_start=None*, *iter_stop=None*)

Check that the given HDF5 object contains (as denoted by its `iter_start/iter_stop` attributes) data at least for the iteration range specified.

check_data_iter_range_equal(*h5object*, *iter_start=None*, *iter_stop=None*)

Check that the given HDF5 object contains (as denoted by its `iter_start/iter_stop` attributes) data exactly for the iteration range specified.

check_data_iter_step_conformant(*h5object*, *iter_step=None*)

Check that the given HDF5 object contains per-iteration data at an iteration stride suitable for extracting data with the given stride (in other words, the given `iter_step` is a multiple of the stride with which data was recorded).

check_data_iter_step_equal(*h5object*, *iter_step=None*)

Check that the given HDF5 object contains per-iteration data at an iteration stride the same as that specified.

slice_per_iter_data(*dataset*, *iter_start=None*, *iter_stop=None*, *iter_step=None*, *axis=0*)

Return the subset of the given dataset corresponding to the given iteration range and stride. Unless otherwise specified, the first dimension of the dataset is the one sliced.

iter_range(*iter_start=None*, *iter_stop=None*, *iter_step=None*, *dtype=None*)

Return a sequence for the given iteration numbers and stride, filling in missing values from those stored on self. The smallest data type capable of holding `iter_stop` is returned unless otherwise specified using the `dtype` argument.

class westpa.cli.tools.w_crawl.**ProgressIndicatorComponent**

Bases: [WESTToolComponent](#)

add_args(*parser*)

Add arguments specific to this component to the given argparse parser.

process_args(args)

Take argparse-processed arguments associated with this component and deal with them appropriately (setting instance variables, etc)

`westpa.cli.tools.w_crawl.get_object(object_name, path=None)`

Attempt to load the given object, using additional path information if given.

class `westpa.cli.tools.w_crawl.WESTPACrawler`

Bases: `object`

Base class for general crawling execution. This class only exists on the master.

initialize(iter_start, iter_stop)

Initialize this crawling process.

finalize()

Finalize this crawling process.

process_iter_result(n_iter, result)

Process the result of a per-iteration task.

class `westpa.cli.tools.w_crawl.WCrawl`

Bases: `WESTParallelTool`

prog = 'w_crawl'

description = 'Crawl a weighted ensemble dataset, executing a function for each iteration.\nThis can be used for postprocessing of trajectories, cleanup of datasets,\nor anything else that can be expressed as "do X for iteration N, then do\nsomething with the result". Tasks are parallelized by iteration, and\nno guarantees are made about evaluation order.\n\n\n-----\nCommand-line options\n-----\n\n'

add_args(parser)

Add arguments specific to this tool to the given argparse parser.

process_args(args)

Take argparse-processed arguments associated with this tool and deal with them appropriately (setting instance variables, etc)

go()

Perform the analysis associated with this tool.

`westpa.cli.tools.w_crawl.entry_point()`

6.1.12 w_direct

usage:

```
w_direct [-h] [-r RCFILE] [--quiet | --verbose | --debug] [--version]
          [--max-queue-length MAX_QUEUE_LENGTH]
          [--serial | --parallel | --work-manager WORK_MANAGER] [--n-workers N_
↪WORKERS]
          [--zmq-mode MODE] [--zmq-comm-mode COMM_MODE] [--zmq-write-host-info INFO_
↪FILE]
          [--zmq-read-host-info INFO_FILE] [--zmq-upstream-rr-endpoint ENDPOINT]
          [--zmq-upstream-ann-endpoint ENDPOINT] [--zmq-downstream-rr-endpoint_
↪ENDPOINT]
          [--zmq-downstream-ann-endpoint ENDPOINT] [--zmq-master-heartbeat MASTER_
↪HEARTBEAT]
          [--zmq-worker-heartbeat WORKER_HEARTBEAT] [--zmq-timeout-factor FACTOR]
          [--zmq-startup-timeout STARTUP_TIMEOUT] [--zmq-shutdown-timeout SHUTDOWN_
↪TIMEOUT]
          {help,init,average,kinetics,probs,all} ...
```

optional arguments:

```
-h, --help          show this help message and exit
```

general options:

```
-r RCFILE, --rcfile RCFILE
                        use RCFILE as the WEST run-time configuration file (default: west.
↪cfg)
--quiet                emit only essential information
--verbose              emit extra information
--debug                enable extra checks and emit copious information
--version              show program's version number and exit
```

parallelization options:

```
--max-queue-length MAX_QUEUE_LENGTH
                        Maximum number of tasks that can be queued. Useful to limit RAM_
↪use for tasks that
                        have very large requests/response. Default: no limit.
```

direct kinetics analysis schemes:

```
{help,init,average,kinetics,probs,all}
  help          print help for this command or individual subcommands
  init          calculate state-to-state kinetics by tracing trajectories
  average       Averages and returns fluxes, rates, and color/state populations.
  kinetics      Generates rate and flux values from a WESTPA simulation via_
↪tracing.
  probs         Calculates color and state probabilities via tracing.
  all           Runs the full suite, including the tracing of events.
```

parallelization options:

```
--serial          run in serial mode
--parallel        run in parallel mode (using processes)
--work-manager WORK_MANAGER
                  use the given work manager for parallel task distribution.
↳ Available work managers
                  are ('serial', 'threads', 'processes', 'zmq'); default is 'serial'
--n-workers N_WORKERS
                  Use up to N_WORKERS on this host, for work managers which support
↳ this option. Use
                  0 for a dedicated server. (Ignored by work managers which do not
↳ support this
                  option.)
```

options for ZeroMQ (“zmq”) work manager (master or node):

```
--zmq-mode MODE    Operate as a master (server) or a node (workers/client). "server"
↳ is a deprecated
                  synonym for "master" and "client" is a deprecated synonym for "node"
↳ ".
--zmq-comm-mode COMM_MODE
                  Use the given communication mode -- TCP or IPC (Unix-domain) --
↳ sockets for
                  communication within a node. IPC (the default) may be more
↳ efficient but is not
                  available on (exceptionally rare) systems without node-local
↳ storage (e.g. /tmp);
                  on such systems, TCP may be used instead.
--zmq-write-host-info INFO_FILE
                  Store hostname and port information needed to connect to this
↳ instance in
                  INFO_FILE. This allows the master and nodes assisting in
↳ coordinating the
                  communication of other nodes to choose ports randomly. Downstream
↳ nodes read this
                  file with --zmq-read-host-info and know where how to connect.
--zmq-read-host-info INFO_FILE
                  Read hostname and port information needed to connect to the master
↳ (or other
                  coordinating node) from INFO_FILE. This allows the master and
↳ nodes assisting in
                  coordinating the communication of other nodes to choose ports
↳ randomly, writing
                  that information with --zmq-write-host-info for this instance to
↳ read.
--zmq-upstream-rr-endpoint ENDPOINT
                  ZeroMQ endpoint to which to send request/response (task and
↳ result) traffic toward
                  the master.
--zmq-upstream-ann-endpoint ENDPOINT
                  ZeroMQ endpoint on which to receive announcement (heartbeat and
↳ shutdown
                  notification) traffic from the master.
--zmq-downstream-rr-endpoint ENDPOINT
```

(continues on next page)

(continued from previous page)

```

ZeroMQ endpoint on which to listen for request/response (task and
↪result) traffic
    from subsidiary workers.
--zmq-downstream-ann-endpoint ENDPOINT
    ZeroMQ endpoint on which to send announcement (heartbeat and
↪shutdown
    notification) traffic toward workers.
--zmq-master-heartbeat MASTER_HEARTBEAT
    Every MASTER_HEARTBEAT seconds, the master announces its presence,
↪to workers.
--zmq-worker-heartbeat WORKER_HEARTBEAT
    Every WORKER_HEARTBEAT seconds, workers announce their presence to,
↪the master.
--zmq-timeout-factor FACTOR
    Scaling factor for heartbeat timeouts. If the master doesn't hear,
↪from a worker in
    WORKER_HEARTBEAT*FACTOR, the worker is assumed to have crashed. If,
↪a worker
    doesn't hear from the master in MASTER_HEARTBEAT*FACTOR seconds,
↪the master is
    assumed to have crashed. Both cases result in shutdown.
--zmq-startup-timeout STARTUP_TIMEOUT
    Amount of time (in seconds) to wait for communication between the,
↪master and at
    least one worker. This may need to be changed on very large,
↪heavily-loaded
    computer systems that start all processes simultaneously.
--zmq-shutdown-timeout SHUTDOWN_TIMEOUT
    Amount of time (in seconds) to wait for workers to shut down.

```

6.1.12.1 westpa.cli.tools.w_direct module

westpa.cli.tools.w_direct.weight_dtype

alias of float64

class westpa.cli.tools.w_direct.WESTMasterCommand

Bases: *WESTTool*

Base class for command-line tools that employ subcommands

subparsers_title = None

subcommands = None

include_help_command = True

add_args(*parser*)

Add arguments specific to this tool to the given argparse parser.

process_args(*args*)

Take argparse-processed arguments associated with this tool and deal with them appropriately (setting instance variables, etc)

go()

Perform the analysis associated with this tool.

class westpa.cli.tools.w_direct.**WESTParallelTool**(*wm_env=None*)

Bases: [WESTTool](#)

Base class for command-line tools parallelized with wwmgr. This automatically adds and processes wwmgr command-line arguments and creates a work manager at self.work_manager.

make_parser_and_process(*prog=None, usage=None, description=None, epilog=None, args=None*)

A convenience function to create a parser, call `add_all_args()`, and then call `process_all_args()`. The argument namespace is returned.

add_args(*parser*)

Add arguments specific to this tool to the given argparse parser.

process_args(*args*)

Take argparse-processed arguments associated with this tool and deal with them appropriately (setting instance variables, etc)

go()

Perform the analysis associated with this tool.

main()

A convenience function to make a parser, parse and process arguments, then run `self.go()` in the master process.

westpa.cli.tools.w_direct.sequence_macro_flux_to_rate(*dataset, pops, istate, jstate, pairwise=True, stride=None*)

Convert a sequence of macrostate fluxes and corresponding list of trajectory ensemble populations to a sequence of rate matrices.

If the optional `pairwise` is true (the default), then rates are normalized according to the relative probability of the initial state among the pair of states (initial, final); this is probably what you want, as these rates will then depend only on the definitions of the states involved (and never the remaining states). Otherwise (`pairwise` is false), the rates are normalized according the probability of the initial state among *all* other states.

class westpa.cli.tools.w_direct.**WKinetics**

Bases: object

w_kinetics()

class westpa.cli.tools.w_direct.**WESTKineticsBase**(*parent*)

Bases: [WESTSubcommand](#)

Common argument processing for w_direct/w_reweight subcommands. Mostly limited to handling input and output from w_assign.

add_args(*parser*)

Add arguments specific to this component to the given argparse parser.

process_args(*args*)

Take argparse-processed arguments associated with this component and deal with them appropriately (setting instance variables, etc)

class westpa.cli.tools.w_direct.**AverageCommands**(*parent*)

Bases: [WESTKineticsBase](#)

```
default_output_file = 'direct.h5'
```

```
add_args(parser)
```

Add arguments specific to this component to the given argparse parser.

```
process_args(args)
```

Take argparse-processed arguments associated with this component and deal with them appropriately (setting instance variables, etc)

```
stamp_mcbs_info(dataset)
```

```
open_files()
```

```
open_assignments()
```

```
print_averages(dataset, header, dim=1)
```

```
run_calculation(pi, nstates, start_iter, stop_iter, step_iter, dataset, eval_block, name, dim,
                do_averages=False, **extra)
```

```
westpa.cli.tools.w_direct.mcbs_ci_correl(estimator_datasets, estimator, alpha, n_sets=None,
                                         args=None, autocorrel_alpha=None, autocorrel_n_sets=None,
                                         subsample=None, do_correl=True, mcbs_enable=None,
                                         estimator_kwargs={})
```

Perform a Monte Carlo bootstrap estimate for the $(1-\alpha)$ confidence interval on the given dataset with the given estimator. This routine is appropriate for time-correlated data, using the method described in Huber & Kim, “Weighted-ensemble Brownian dynamics simulations for protein association reactions” (1996), doi:10.1016/S0006-3495(96)79552-8 to determine a statistically-significant correlation time and then reducing the dataset by a factor of that correlation time before running a “classic” Monte Carlo bootstrap.

Returns (estimate, ci_lb, ci_ub, correl_time) where estimate is the application of the given estimator to the input dataset, ci_lb and ci_ub are the lower and upper limits, respectively, of the $(1-\alpha)$ confidence interval on estimate, and correl_time is the correlation time of the dataset, significant to $(1-\text{autocorrel_alpha})$.

estimator is called as `estimator(dataset, *args, **kwargs)`. Common estimators include:

- np.mean – calculate the confidence interval on the mean of dataset
- np.median – calculate a confidence interval on the median of dataset
- np.std – calculate a confidence interval on the standard deviation of dataset.

n_sets is the number of synthetic data sets to generate using the given estimator, which will be chosen using ``get_bssize()`` if n_sets is not given.

autocorrel_alpha (which defaults to alpha) can be used to adjust the significance level of the autocorrelation calculation. Note that too high a significance level (too low an alpha) for evaluating the significance of autocorrelation values can result in a failure to detect correlation if the autocorrelation function is noisy.

The given subsample function is used, if provided, to subsample the dataset prior to running the full Monte Carlo bootstrap. If none is provided, then a random entry from each correlated block is used as the value for that block. Other reasonable choices include np.mean, np.median, (lambda x: x[0]) or (lambda x: x[-1]). In particular, using subsample=np.mean will converge to the block averaged mean and standard error, while accounting for any non-normality in the distribution of the mean.

```
westpa.cli.tools.w_direct.accumulate_state_populations_from_labeled(labeled_bin_pops,
                                                                    state_map, state_pops,
                                                                    check_state_map=True)
```

```
class westpa.cli.tools.w_direct.DKinetics(parent)
```

Bases: [WESTKineticsBase](#), [WKinetics](#)

```

subcommand = 'init'

default_kinetics_file = 'direct.h5'

default_output_file = 'direct.h5'

help_text = 'calculate state-to-state kinetics by tracing trajectories'

description = 'Calculate state-to-state rates and transition event durations by
tracing\ntrajectories.\n\nA bin assignment file (usually "assign.h5") including
trajectory labeling\nis required (see "w_assign --help" for information on
generating this file).\n\nThis subcommand for w_direct is used as input for all
other w_direct\nsubcommands, which will convert the flux data in the output file
into\naverage rates/fluxes/populations with confidence intervals.\n\
n-----\
nOutput
format\
n-----\n\
nThe output file (-o/--output, by default "direct.h5") contains the\nfollowing
datasets:\n\n ``/conditional_fluxes`` [iteration][state][state]\n *(Floating-point)*
Macrostate-to-macrostate fluxes. These are **not**\n normalized by the population of
the initial macrostate.\n\n ``/conditional_arrivals`` [iteration][stateA][stateB]\n
*(Integer)* Number of trajectories arriving at state *stateB* in a given\n
iteration, given that they departed from *stateA*.\n\n ``/total_fluxes``
[iteration][state]\n *(Floating-point)* Total flux into a given macrostate.\n\n
``/arrivals`` [iteration][state]\n *(Integer)* Number of trajectories arriving at a
given state in a given\n iteration, regardless of where they originated.\n\n
``/duration_count`` [iteration]\n *(Integer)* The number of event durations recorded
in each iteration.\n\n ``/durations`` [iteration][event duration]\n *(Structured --
see below)* Event durations for transition events ending\n during a given iteration.
These are stored as follows:\n\n istate\n *(Integer)* Initial state of transition
event.\n\n fstate\n *(Integer)* Final state of transition event.\n\n duration\n
*(Floating-point)* Duration of transition, in units of tau.\n\n weight\n
*(Floating-point)* Weight of trajectory at end of transition, **not**\n normalized
by initial state population.\n\nBecause state-to-state fluxes stored in this file
are not normalized by\ninitial macrostate population, they cannot be used as rates
without further\nprocessing. The ``w_direct kinetics`` command is used to perform
this normalization\nwhile taking statistical fluctuation and correlation into
account. See\n``w_direct kinetics --help`` for more information. Target fluxes
(total flux\ninto a given state) require no such normalization.\n\
n-----\
nCommand-line
options\
n-----\n'

open_files()

go()

class westpa.cli.tools.w_direct.DKinAvg(parent)
    Bases: AverageCommands
    subcommand = 'kinetics'

    help_text = 'Generates rate and flux values from a WESTPA simulation via tracing.'
```

```

default_kinetics_file = 'direct.h5'

description = 'Calculate average rates/fluxes and associated errors from weighted
ensemble\ndata. Bin assignments (usually "assign.h5") and kinetics data
(usually\n"direct.h5") data files must have been previously generated
(see\n"w_assign --help" and "w_direct init --help" for information on\ngenerating
these files).\n\nThe evolution of all datasets may be calculated, with or without
confidence\nintervals.\n\
n-----\
nOutput
format\
n-----\n\
nThe output file (-o/--output, usually "direct.h5") contains the
following\ndataset:\n\n /avg_rates [state,state]\n (Structured -- see below)
State-to-state rates based on entire window of\n iterations selected.\n\n
/avg_total_fluxes [state]\n (Structured -- see below) Total fluxes into each state
based on entire\n window of iterations selected.\n\n /avg_conditional_fluxes
[state,state]\n (Structured -- see below) State-to-state fluxes based on entire
window of\n iterations selected.\n\nIf --evolution-mode is specified, then the
following additional datasets are\navailable:\n\n /rate_evolution
[window][state][state]\n (Structured -- see below). State-to-state rates based on
windows of\n iterations of varying width. If --evolution-mode=cumulative, then\n
these windows all begin at the iteration specified with\n --start-iter and grow in
length by --step-iter for each successive\n element. If --evolution-mode=blocked,
then these windows are all of\n width --step-iter (excluding the last, which may be
shorter), the first\n of which begins at iteration --start-iter.\n\n
/target_flux_evolution [window,state]\n (Structured -- see below). Total flux into a
given macro state based on\n windows of iterations of varying width, as in
/rate_evolution.\n\n /conditional_flux_evolution [window,state,state]\n (Structured
-- see below). State-to-state fluxes based on windows of\n varying width, as in
/rate_evolution.\n\nThe structure of these datasets is as follows:\n\n iter_start\n
(Integer) Iteration at which the averaging window begins (inclusive).\n\n
iter_stop\n (Integer) Iteration at which the averaging window ends (exclusive).\n\n
expected\n (Floating-point) Expected (mean) value of the observable as evaluated
within\n this window, in units of inverse tau.\n\n ci_lbound\n (Floating-point)
Lower bound of the confidence interval of the observable\n within this window, in
units of inverse tau.\n\n ci_ubound\n (Floating-point) Upper bound of the confidence
interval of the observable\n within this window, in units of inverse tau.\n\n
stderr\n (Floating-point) The standard error of the mean of the observable\n within
this window, in units of inverse tau.\n\n corr_len\n (Integer) Correlation length of
the observable within this window, in units\n of tau.\n\nEach of these datasets is
also stamped with a number of attributes:\n\n mcbs_alpha\n (Floating-point) Alpha
value of confidence intervals. (For example,\n *alpha=0.05* corresponds to a 95%
confidence interval.)\n\n mcbs_nsets\n (Integer) Number of bootstrap data sets used
in generating confidence\n intervals.\n\n mcbs_acalpha\n (Floating-point) Alpha
value for determining correlation lengths.\n\n\
n-----\
nCommand-line
options\
n-----\n'

w_kinavg()

go()

```

```

class westpa.cli.tools.w_direct.DStateProbs(parent)
    Bases: AverageCommands

    subcommand = 'probs'

    help_text = 'Calculates color and state probabilities via tracing.'

    default_kinetics_file = 'direct.h5'

    description = 'Calculate average populations and associated errors in state
populations from nweighted ensemble data. Bin assignments, including macrostate
definitions, are required. (See "w_assign --help" for more information).\n\
n-----\
nOutput
format\
n-----\n\
nThe output file (-o/--output, usually "direct.h5") contains the
following dataset:\n\n /avg_state_probs [state]\n (Structured -- see below)
Population of each state across entire\n range specified.\n\n /avg_color_probs
[state]\n (Structured -- see below) Population of each ensemble across entire\n
range specified.\n\n If --evolution-mode is specified, then the following additional
datasets are available:\n\n /state_pop_evolution [window][state]\n (Structured --
see below). State populations based on windows of\n iterations of varying width. If
--evolution-mode=cumulative, then\n these windows all begin at the iteration
specified with\n --start-iter and grow in length by --step-iter for each
successive\n element. If --evolution-mode=blocked, then these windows are all of\n
width --step-iter (excluding the last, which may be shorter), the first\n of which
begins at iteration --start-iter.\n\n /color_prob_evolution [window][state]\n
(Structured -- see below). Ensemble populations based on windows of\n iterations of
varying width. If --evolution-mode=cumulative, then\n these windows all begin at the
iteration specified with\n --start-iter and grow in length by --step-iter for each
successive\n element. If --evolution-mode=blocked, then these windows are all of\n
width --step-iter (excluding the last, which may be shorter), the first\n of which
begins at iteration --start-iter.\n\n The structure of these datasets is as
follows:\n\n iter_start\n (Integer) Iteration at which the averaging window begins
(inclusive).\n\n iter_stop\n (Integer) Iteration at which the averaging window ends
(exclusive).\n\n expected\n (Floating-point) Expected (mean) value of the observable
as evaluated within\n this window, in units of inverse tau.\n\n ci_lbound\n
(Floating-point) Lower bound of the confidence interval of the observable\n within
this window, in units of inverse tau.\n\n ci_ubound\n (Floating-point) Upper bound
of the confidence interval of the observable\n within this window, in units of
inverse tau.\n\n stderr\n (Floating-point) The standard error of the mean of the
observable\n within this window, in units of inverse tau.\n\n corr_len\n (Integer)
Correlation length of the observable within this window, in units\n of tau.\n\n Each
of these datasets is also stamped with a number of attributes:\n\n mcbs_alpha\n
(Floating-point) Alpha value of confidence intervals. (For example,\n *alpha=0.05*
corresponds to a 95% confidence interval.)\n\n mcbs_nsets\n (Integer) Number of
bootstrap data sets used in generating confidence\n intervals.\n\n mcbs_acalpha\n
(Floating-point) Alpha value for determining correlation lengths.\n\n\
n-----\
nCommand-line
options\
n-----\n'

    calculate_state_populations(pops)

```

```
w_stateprobs()

go()

class westpa.cli.tools.w_direct.DAll(parent)
    Bases: DStateProbs, DKinAvg, DKinetics

    subcommand = 'all'

    help_text = 'Runs the full suite, including the tracing of events.'

    default_kinetics_file = 'direct.h5'

    description = 'A convenience function to run init/kinetics/probs. Bin
    assignments,\nincluding macrostate definitions, are required. (See\n"w_assign
    --help" for more information).\n\nFor more information on the individual subcommands
    this subs in for, run\nw_direct {init/kinetics/probs} --help.\n\
    n-----\
    nCommand-line
    options\
    n-----\n'

    go()

class westpa.cli.tools.w_direct.DAverage(parent)
    Bases: DStateProbs, DKinAvg

    subcommand = 'average'

    help_text = 'Averages and returns fluxes, rates, and color/state populations.'

    default_kinetics_file = 'direct.h5'

    description = 'A convenience function to run kinetics/probs. Bin
    assignments,\nincluding macrostate definitions, are required. (See\n"w_assign
    --help" for more information).\n\nFor more information on the individual subcommands
    this subs in for, run\nw_direct {kinetics/probs} --help.\n\
    n-----\
    nCommand-line
    options\
    n-----\n'

    go()

class westpa.cli.tools.w_direct.WDirect
    Bases: WESTMasterCommand, WESTParallelTool

    prog = 'w_direct'

    subcommands = [<class 'westpa.cli.tools.w_direct.DKinetics'>, <class
    'westpa.cli.tools.w_direct.DAverage'>, <class 'westpa.cli.tools.w_direct.DKinAvg'>,
    <class 'westpa.cli.tools.w_direct.DStateProbs'>, <class
    'westpa.cli.tools.w_direct.DAll'>]

    subparsers_title = 'direct kinetics analysis schemes'

westpa.cli.tools.w_direct.entry_point()
```

6.1.13 w_select

usage:

```
w_select [-h] [-r RCFILE] [--quiet | --verbose | --debug] [--version]
          [--max-queue-length MAX_QUEUE_LENGTH] [-W WEST_H5FILE] [--first-iter N_
↪ITER]
          [--last-iter N_ITER] [-p MODULE.FUNCTION] [-v] [-a] [-o OUTPUT]
          [--serial | --parallel | --work-manager WORK_MANAGER] [--n-workers N_
↪WORKERS]
          [--zmq-mode MODE] [--zmq-comm-mode COMM_MODE] [--zmq-write-host-info INFO_
↪FILE]
          [--zmq-read-host-info INFO_FILE] [--zmq-upstream-rr-endpoint ENDPOINT]
          [--zmq-upstream-ann-endpoint ENDPOINT] [--zmq-downstream-rr-endpoint_
↪ENDPOINT]
          [--zmq-downstream-ann-endpoint ENDPOINT] [--zmq-master-heartbeat MASTER_
↪HEARTBEAT]
          [--zmq-worker-heartbeat WORKER_HEARTBEAT] [--zmq-timeout-factor FACTOR]
          [--zmq-startup-timeout STARTUP_TIMEOUT] [--zmq-shutdown-timeout SHUTDOWN_
↪TIMEOUT]
```

Select dynamics segments matching various criteria. This requires a user-provided predicate function. By default, only matching segments are stored. If the `-a/--include-ancestors` option is given, then matching segments and their ancestors will be stored.

6.1.13.1 Predicate function

Segments are selected based on a predicate function, which must be callable as `predicate(n_iter, iter_group)` and return a collection of segment IDs matching the predicate in that iteration.

The predicate may be inverted by specifying the `-v/--invert` command-line argument.

6.1.13.2 Output format

The output file (`-o/--output`, by default “select.h5”) contains the following datasets:

```
``/n_iter`` [iteration]
  *(Integer)* Iteration numbers for each entry in other datasets.

``/n_segs`` [iteration]
  *(Integer)* Number of segment IDs matching the predicate (or inverted
  predicate, if -v/--invert is specified) in the given iteration.

``/seg_ids`` [iteration][segment]
  *(Integer)* Matching segments in each iteration. For an iteration
  ``n_iter``, only the first ``n_iter`` entries are valid. For example,
  the full list of matching seg_ids in the first stored iteration is
  ``seg_ids[0][:n_segs[0]]``.

``/weights`` [iteration][segment]
  *(Floating-point)* Weights for each matching segment in ``/seg_ids``.
```

6.1.13.3 Command-line arguments

optional arguments:

<code>-h, --help</code>	show this help message and exit
-------------------------	--

general options:

<code>-r RCFILE, --rcfile RCFILE</code>	use RCFILE as the WEST run-time configuration file (default: west. → <code>cfg</code>)
<code>--quiet</code>	emit only essential information
<code>--verbose</code>	emit extra information
<code>--debug</code>	enable extra checks and emit copious information
<code>--version</code>	show program's version number and exit

parallelization options:

<code>--max-queue-length MAX_QUEUE_LENGTH</code>	Maximum number of tasks that can be queued. Useful to limit RAM. →use for tasks that
	have very large requests/response. Default: no limit.

WEST input data options:

<code>-W WEST_H5FILE, --west-data WEST_H5FILE</code>	Take WEST data from WEST_H5FILE (default: read from the HDF5 file. →specified in
	west.cfg).

iteration range:

<code>--first-iter N_ITER</code>	Begin analysis at iteration N_ITER (default: 1).
<code>--last-iter N_ITER</code>	Conclude analysis with N_ITER, inclusive (default: last completed. →iteration).

selection options:

<code>-p MODULE.FUNCTION, --predicate-function MODULE.FUNCTION</code>	Use the given predicate function to match segments. This function. →should take an
	iteration number and the HDF5 group corresponding to that. →iteration and return a
	sequence of seg_ids matching the predicate, as in ``match_ →predicate(n_iter,
	iter_group)``.
<code>-v, --invert</code>	Invert the match predicate.
<code>-a, --include-ancestors</code>	Include ancestors of matched segments in output.

output options:

`-o OUTPUT, --output OUTPUT` Write output to OUTPUT (default: select.h5).

parallelization options:


```
--serial          run in serial mode
--parallel        run in parallel mode (using processes)
--work-manager WORK_MANAGER
                  use the given work manager for parallel task distribution.
↳ Available work managers
                  are ('serial', 'threads', 'processes', 'zmq'); default is 'serial'
--n-workers N_WORKERS
                  Use up to N_WORKERS on this host, for work managers which support
↳ this option. Use
                  0 for a dedicated server. (Ignored by work managers which do not
↳ support this
                  option.)
```

options for ZeroMQ (“zmq”) work manager (master or node):

```
--zmq-mode MODE    Operate as a master (server) or a node (workers/client). "server"
↳ is a deprecated
                  synonym for "master" and "client" is a deprecated synonym for "node
↳ ".
--zmq-comm-mode COMM_MODE
                  Use the given communication mode -- TCP or IPC (Unix-domain) --
↳ sockets for
                  communication within a node. IPC (the default) may be more
↳ efficient but is not
                  available on (exceptionally rare) systems without node-local
↳ storage (e.g. /tmp);
                  on such systems, TCP may be used instead.
--zmq-write-host-info INFO_FILE
                  Store hostname and port information needed to connect to this
↳ instance in
                  INFO_FILE. This allows the master and nodes assisting in
↳ coordinating the
                  communication of other nodes to choose ports randomly. Downstream
↳ nodes read this
                  file with --zmq-read-host-info and know where how to connect.
--zmq-read-host-info INFO_FILE
                  Read hostname and port information needed to connect to the master
↳ (or other
                  coordinating node) from INFO_FILE. This allows the master and
↳ nodes assisting in
                  coordinating the communication of other nodes to choose ports
↳ randomly, writing
                  that information with --zmq-write-host-info for this instance to
↳ read.
--zmq-upstream-rr-endpoint ENDPOINT
                  ZeroMQ endpoint to which to send request/response (task and
↳ result) traffic toward
                  the master.
--zmq-upstream-ann-endpoint ENDPOINT
                  ZeroMQ endpoint on which to receive announcement (heartbeat and
↳ shutdown
                  notification) traffic from the master.
--zmq-downstream-rr-endpoint ENDPOINT
```

(continues on next page)

(continued from previous page)

```

ZeroMQ endpoint on which to listen for request/response (task and
↪result) traffic
    from subsidiary workers.
--zmq-downstream-ann-endpoint ENDPOINT
    ZeroMQ endpoint on which to send announcement (heartbeat and
↪shutdown
    notification) traffic toward workers.
--zmq-master-heartbeat MASTER_HEARTBEAT
    Every MASTER_HEARTBEAT seconds, the master announces its presence,
↪to workers.
--zmq-worker-heartbeat WORKER_HEARTBEAT
    Every WORKER_HEARTBEAT seconds, workers announce their presence to
↪the master.
--zmq-timeout-factor FACTOR
    Scaling factor for heartbeat timeouts. If the master doesn't hear
↪from a worker in
    WORKER_HEARTBEAT*FACTOR, the worker is assumed to have crashed. If
↪a worker
    doesn't hear from the master in MASTER_HEARTBEAT*FACTOR seconds,
↪the master is
    assumed to have crashed. Both cases result in shutdown.
--zmq-startup-timeout STARTUP_TIMEOUT
    Amount of time (in seconds) to wait for communication between the
↪master and at
    least one worker. This may need to be changed on very large,
↪heavily-loaded
    computer systems that start all processes simultaneously.
--zmq-shutdown-timeout SHUTDOWN_TIMEOUT
    Amount of time (in seconds) to wait for workers to shut down.

```

6.1.13.4 westpa.cli.tools.w_select module

class westpa.cli.tools.w_select.WESTParallelTool(wm_env=None)

Bases: [WESTTool](#)

Base class for command-line tools parallelized with wwmgr. This automatically adds and processes wwmgr command-line arguments and creates a work manager at self.work_manager.

make_parser_and_process(prog=None, usage=None, description=None, epilog=None, args=None)

A convenience function to create a parser, call add_all_args(), and then call process_all_args(). The argument namespace is returned.

add_args(parser)

Add arguments specific to this tool to the given argparse parser.

process_args(args)

Take argparse-processed arguments associated with this tool and deal with them appropriately (setting instance variables, etc)

go()

Perform the analysis associated with this tool.

main()

A convenience function to make a parser, parse and process arguments, then run `self.go()` in the master process.

class westpa.cli.tools.w_select.WESTDataReader

Bases: [*WESTToolComponent*](#)

Tool for reading data from WEST-related HDF5 files. Coordinates finding the main HDF5 file from `west.cfg` or command line arguments, caching of certain kinds of data (eventually), and retrieving auxiliary data sets from various places.

add_args(parser)

Add arguments specific to this component to the given argparse parser.

process_args(args)

Take argparse-processed arguments associated with this component and deal with them appropriately (setting instance variables, etc)

open(mode='r')**close()****property weight_dsspec****property parent_id_dsspec****class westpa.cli.tools.w_select.IterRangeSelection(data_manager=None)**

Bases: [*WESTToolComponent*](#)

Select and record limits on iterations used in analysis and/or reporting. This class provides both the user-facing command-line options and parsing, and the application-side API for recording limits in HDF5.

HDF5 datasets calculated based on a restricted set of iterations should be tagged with the following attributes:

first_iter

The first iteration included in the calculation.

last_iter

One past the last iteration included in the calculation.

iter_step

Blocking or sampling period for iterations included in the calculation.

add_args(parser)

Add arguments specific to this component to the given argparse parser.

process_args(args, override_iter_start=None, override_iter_stop=None, default_iter_step=1)

Take argparse-processed arguments associated with this component and deal with them appropriately (setting instance variables, etc)

iter_block_iter()

Return an iterable of (block_start,block_end) over the blocks of iterations selected by `–first-iter/–last-iter/–step-iter`.

n_iter_blocks()

Return the number of blocks of iterations (as returned by `iter_block_iter`) selected by `–first-iter/–last-iter/–step-iter`.

record_data_iter_range(h5object, iter_start=None, iter_stop=None)

Store attributes `iter_start` and `iter_stop` on the given HDF5 object (group/dataset)

record_data_iter_step(*h5object*, *iter_step=None*)

Store attribute `iter_step` on the given HDF5 object (group/dataset).

check_data_iter_range_least(*h5object*, *iter_start=None*, *iter_stop=None*)

Check that the given HDF5 object contains (as denoted by its `iter_start/iter_stop` attributes) data at least for the iteration range specified.

check_data_iter_range_equal(*h5object*, *iter_start=None*, *iter_stop=None*)

Check that the given HDF5 object contains (as denoted by its `iter_start/iter_stop` attributes) data exactly for the iteration range specified.

check_data_iter_step_conformant(*h5object*, *iter_step=None*)

Check that the given HDF5 object contains per-iteration data at an iteration stride suitable for extracting data with the given stride (in other words, the given `iter_step` is a multiple of the stride with which data was recorded).

check_data_iter_step_equal(*h5object*, *iter_step=None*)

Check that the given HDF5 object contains per-iteration data at an iteration stride the same as that specified.

slice_per_iter_data(*dataset*, *iter_start=None*, *iter_stop=None*, *iter_step=None*, *axis=0*)

Return the subset of the given dataset corresponding to the given iteration range and stride. Unless otherwise specified, the first dimension of the dataset is the one sliced.

iter_range(*iter_start=None*, *iter_stop=None*, *iter_step=None*, *dtype=None*)

Return a sequence for the given iteration numbers and stride, filling in missing values from those stored on `self`. The smallest data type capable of holding `iter_stop` is returned unless otherwise specified using the `dtype` argument.

class `westpa.cli.tools.w_select.ProgressIndicatorComponent`

Bases: [`WESTToolComponent`](#)

add_args(*parser*)

Add arguments specific to this component to the given argparse parser.

process_args(*args*)

Take argparse-processed arguments associated with this component and deal with them appropriately (setting instance variables, etc)

`westpa.cli.tools.w_select.seg_id_dtype`

alias of `int64`

`westpa.cli.tools.w_select.n_iter_dtype`

alias of `uint32`

`westpa.cli.tools.w_select.weight_dtype`

alias of `float64`

`westpa.cli.tools.w_select.get_object`(*object_name*, *path=None*)

Attempt to load the given object, using additional path information if given.

class `westpa.cli.tools.w_select.WSelectTool`

Bases: [`WESTParallelTool`](#)

prog = `'w_select'`

```

description = 'Select dynamics segments matching various criteria. This requires
a\nuser-provided predicate function. By default, only matching segments are\nstored.
If the -a/--include-ancestors option is given, then matching segments\nand their
ancestors will be stored.\n\n\
n-----\
nPredicate
function\
n-----\n\
nSegments are selected based on a predicate function, which must be callable\nas
``predicate(n_iter, iter_group)`` and return a collection of segment IDs\nmatching
the predicate in that iteration.\n\nThe predicate may be inverted by specifying the
-v/--invert command-line\nargument.\n\n\
n-----\
nOutput
format\
n-----\n\
nThe output file (-o/--output, by default "select.h5") contains the
following\ndatasets:\n\n ``/n_iter`` [iteration]\n *(Integer)* Iteration numbers for
each entry in other datasets.\n\n ``/n_segs`` [iteration]\n *(Integer)* Number of
segment IDs matching the predicate (or inverted\n predicate, if -v/--invert is
specified) in the given iteration.\n\n ``/seg_ids`` [iteration][segment]\n
*(Integer)* Matching segments in each iteration. For an iteration\n ``n_iter``, only
the first ``n_iter`` entries are valid. For example,\n the full list of matching
seg_ids in the first stored iteration is\n ``seg_ids[0][:n_segs[0]]``.\n\n
``/weights`` [iteration][segment]\n *(Floating-point)* Weights for each matching
segment in ``/seg_ids``.\n\n\
n-----\
nCommand-line
arguments\
n-----\n'

```

add_args(parser)

Add arguments specific to this tool to the given argparse parser.

process_args(args)

Take argparse-processed arguments associated with this tool and deal with them appropriately (setting instance variables, etc)

go()

Perform the analysis associated with this tool.

`westpa.cli.tools.w_select.entry_point()`

6.1.14 w_states

usage:

```

w_states [-h] [-r RCFILE] [--quiet | --verbose | --debug] [--version]
          [--show | --append | --replace] [--bstate-file BSTATE_FILE] [--bstate-
↪BSTATES]
          [--tstate-file TSTATE_FILE] [--tstate TSTATES]
          [--serial | --parallel | --work-manager WORK_MANAGER] [--n-workers N_
↪WORKERS]

```

(continues on next page)

(continued from previous page)

```

[--zmq-mode MODE] [--zmq-comm-mode COMM_MODE] [--zmq-write-host-info INFO_
↪FILE]
[--zmq-read-host-info INFO_FILE] [--zmq-upstream-rr-endpoint ENDPOINT]
[--zmq-upstream-ann-endpoint ENDPOINT] [--zmq-downstream-rr-endpoint_
↪ENDPOINT]
[--zmq-downstream-ann-endpoint ENDPOINT] [--zmq-master-heartbeat MASTER_
↪HEARTBEAT]
[--zmq-worker-heartbeat WORKER_HEARTBEAT] [--zmq-timeout-factor FACTOR]
[--zmq-startup-timeout STARTUP_TIMEOUT] [--zmq-shutdown-timeout SHUTDOWN_
↪TIMEOUT]

```

Display or manipulate basis (initial) or target (recycling) states for a WEST simulation. By default, states are displayed (or dumped to files). If `--replace` is specified, all basis/target states are replaced for the next iteration. If `--append` is specified, the given target state(s) are appended to the list for the next iteration. Appending basis states is not permitted, as this would require renormalizing basis state probabilities in ways that may be error-prone. Instead, use `w_states --show --bstate-file=bstates.txt` and then edit the resulting `bstates.txt` file to include the new desired basis states, then use `w_states --replace --bstate-file=bstates.txt` to update the WEST HDF5 file appropriately.

optional arguments:

```

-h, --help          show this help message and exit
--bstate-file BSTATE_FILE
                    Read (--append/--replace) or write (--show) basis state names,
↪probabilities, and
                    data references from/to BSTATE_FILE.
--bstate BSTATES    Add the given basis state (specified as a string 'label,
↪probability[,auxref]') to
                    the list of basis states (after those specified in --bstate-file,
↪if any). This
                    argument may be specified more than once, in which case the given
↪states are
                    appended in the order they are given on the command line.
--tstate-file TSTATE_FILE
                    Read (--append/--replace) or write (--show) target state names and
↪representative
                    progress coordinates from/to TSTATE_FILE
--tstate TSTATES    Add the given target state (specified as a string 'label,pcoord0[,
↪pcoord1[,...]]')
                    to the list of target states (after those specified in the file
↪given by
                    --tstates-from, if any). This argument may be specified more than
↪once, in which
                    case the given states are appended in the order they appear on the
↪command line.

```

general options:

```

-r RCFILE, --rcfile RCFILE
                    use RCFILE as the WEST run-time configuration file (default: west.
↪cfg)
--quiet            emit only essential information
--verbose         emit extra information
--debug           enable extra checks and emit copious information

```

(continues on next page)

(continued from previous page)

```
--version          show program's version number and exit
```

modes of operation:

```
--show          Display current basis/target states (or dump to files).
--append        Append the given basis/target states to those currently in use.
--replace       Replace current basis/target states with those specified.
```

parallelization options:

```
--serial        run in serial mode
--parallel       run in parallel mode (using processes)
--work-manager WORK_MANAGER
                use the given work manager for parallel task distribution.
↳ Available work managers
                are ('serial', 'threads', 'processes', 'zmq'); default is 'serial'
--n-workers N_WORKERS
                Use up to N_WORKERS on this host, for work managers which support
↳ this option. Use
                0 for a dedicated server. (Ignored by work managers which do not
↳ support this
                option.)
```

options for ZeroMQ (“zmq”) work manager (master or node):

```
--zmq-mode MODE  Operate as a master (server) or a node (workers/client). "server"
↳ is a deprecated
                synonym for "master" and "client" is a deprecated synonym for "node
↳ ".
--zmq-comm-mode COMM_MODE
                Use the given communication mode -- TCP or IPC (Unix-domain) --
↳ sockets for
                communication within a node. IPC (the default) may be more
↳ efficient but is not
                available on (exceptionally rare) systems without node-local
↳ storage (e.g. /tmp);
                on such systems, TCP may be used instead.
--zmq-write-host-info INFO_FILE
                Store hostname and port information needed to connect to this
↳ instance in
                INFO_FILE. This allows the master and nodes assisting in
↳ coordinating the
                communication of other nodes to choose ports randomly. Downstream
↳ nodes read this
                file with --zmq-read-host-info and know where how to connect.
--zmq-read-host-info INFO_FILE
                Read hostname and port information needed to connect to the master
↳ (or other
                coordinating node) from INFO_FILE. This allows the master and
↳ nodes assisting in
                coordinating the communication of other nodes to choose ports
↳ randomly, writing
                that information with --zmq-write-host-info for this instance to
```

(continues on next page)

(continued from previous page)

```

↪read.
--zmq-upstream-rr-endpoint ENDPOINT
    ZeroMQ endpoint to which to send request/response (task and
↪result) traffic toward
    the master.
--zmq-upstream-ann-endpoint ENDPOINT
    ZeroMQ endpoint on which to receive announcement (heartbeat and
↪shutdown
    notification) traffic from the master.
--zmq-downstream-rr-endpoint ENDPOINT
    ZeroMQ endpoint on which to listen for request/response (task and
↪result) traffic
    from subsidiary workers.
--zmq-downstream-ann-endpoint ENDPOINT
    ZeroMQ endpoint on which to send announcement (heartbeat and
↪shutdown
    notification) traffic toward workers.
--zmq-master-heartbeat MASTER_HEARTBEAT
    Every MASTER_HEARTBEAT seconds, the master announces its presence
↪to workers.
--zmq-worker-heartbeat WORKER_HEARTBEAT
    Every WORKER_HEARTBEAT seconds, workers announce their presence to
↪the master.
--zmq-timeout-factor FACTOR
    Scaling factor for heartbeat timeouts. If the master doesn't hear
↪from a worker in
    WORKER_HEARTBEAT*FACTOR, the worker is assumed to have crashed. If
↪a worker
    doesn't hear from the master in MASTER_HEARTBEAT*FACTOR seconds,
↪the master is
    assumed to have crashed. Both cases result in shutdown.
--zmq-startup-timeout STARTUP_TIMEOUT
    Amount of time (in seconds) to wait for communication between the
↪master and at
    least one worker. This may need to be changed on very large,
↪heavily-loaded
    computer systems that start all processes simultaneously.
--zmq-shutdown-timeout SHUTDOWN_TIMEOUT
    Amount of time (in seconds) to wait for workers to shut down.

```

6.1.14.1 westpa.cli.core.w_states module

`westpa.cli.core.w_states.make_work_manager()`

Using cues from the environment, instantiate a pre-configured work manager.

class `westpa.cli.core.w_states.Segment` (*n_iter=None, seg_id=None, weight=None, endpoint_type=None, parent_id=None, wtg_parent_ids=None, pcoord=None, status=None, walltime=None, cputime=None, data=None*)

Bases: `object`

A class wrapping segment data that must be passed through the work manager or data manager. Most fields are self-explanatory. One item worth noting is that a negative parent ID means that the segment starts from the initial


```

state with ID -(segment.parent_id+1)

SEG_STATUS_UNSET = 0

SEG_STATUS_PREPARED = 1

SEG_STATUS_COMPLETE = 2

SEG_STATUS_FAILED = 3

SEG_INITPOINT_UNSET = 0

SEG_INITPOINT_CONTINUES = 1

SEG_INITPOINT_NEWTRAJ = 2

SEG_ENDPOINT_UNSET = 0

SEG_ENDPOINT_CONTINUES = 1

SEG_ENDPOINT_MERGED = 2

SEG_ENDPOINT_RECYCLED = 3

statuses = {'SEG_STATUS_COMPLETE': 2, 'SEG_STATUS_FAILED': 3, 'SEG_STATUS_PREPARED':
1, 'SEG_STATUS_UNSET': 0}

initpoint_types = {'SEG_INITPOINT_CONTINUES': 1, 'SEG_INITPOINT_NEWTRAJ': 2,
'SEG_INITPOINT_UNSET': 0}

endpoint_types = {'SEG_ENDPOINT_CONTINUES': 1, 'SEG_ENDPOINT_MERGED': 2,
'SEG_ENDPOINT_RECYCLED': 3, 'SEG_ENDPOINT_UNSET': 0}

status_names = {0: 'SEG_STATUS_UNSET', 1: 'SEG_STATUS_PREPARED', 2:
'SEG_STATUS_COMPLETE', 3: 'SEG_STATUS_FAILED'}

initpoint_type_names = {0: 'SEG_INITPOINT_UNSET', 1: 'SEG_INITPOINT_CONTINUES', 2:
'SEG_INITPOINT_NEWTRAJ'}

endpoint_type_names = {0: 'SEG_ENDPOINT_UNSET', 1: 'SEG_ENDPOINT_CONTINUES', 2:
'SEG_ENDPOINT_MERGED', 3: 'SEG_ENDPOINT_RECYCLED'}

static initial_pcoord(segment)
    Return the initial progress coordinate point of this segment.

static final_pcoord(segment)
    Return the final progress coordinate point of this segment.

property initpoint_type

property initial_state_id

property status_text

property endpoint_type_text

```

```
class westpa.cli.core.w_states.BasisState(label, probability, pcoord=None, auxref=None,
                                          state_id=None)
```

Bases: object

Describes an basis (micro)state. These basis states are used to generate initial states for new trajectories, either at the beginning of the simulation (i.e. at `w_init`) or due to recycling.

Variables

- **state_id** – Integer identifier of this state, usually set by the data manager.
- **label** – A descriptive label for this microstate (may be empty)
- **probability** – Probability of this state to be selected when creating a new trajectory.
- **pcoord** – The representative progress coordinate of this state.
- **auxref** – A user-provided (string) reference for locating data associated with this state (usually a filesystem path).

```
classmethod states_to_file(states, fileobj)
```

Write a file defining basis states, which may then be read by `states_from_file()`.

```
classmethod states_from_file(statefile)
```

Read a file defining basis states. Each line defines a state, and contains a label, the probability, and optionally a data reference, separated by whitespace, as in:

```
unbound      1.0
```

or:

```
unbound_0    0.6      state0.pdb
unbound_1    0.4      state1.pdb
```

```
as_numpy_record()
```

Return the data for this state as a numpy record array.

```
class westpa.cli.core.w_states.TargetState(label, pcoord, state_id=None)
```

Bases: object

Describes a target state.

Variables

- **state_id** – Integer identifier of this state, usually set by the data manager.
- **label** – A descriptive label for this microstate (may be empty)
- **pcoord** – The representative progress coordinate of this state.

```
classmethod states_to_file(states, fileobj)
```

Write a file defining basis states, which may then be read by `states_from_file()`.

```
classmethod states_from_file(statefile, dtype)
```

Read a file defining target states. Each line defines a state, and contains a label followed by a representative progress coordinate value, separated by whitespace, as in:

```
bound        0.02
```

for a single target and one-dimensional progress coordinates or:

```
bound        2.7      0.0
drift        100      50.0
```

for two targets and a two-dimensional progress coordinate.

```
westpa.cli.core.w_states.entry_point()
```

```
westpa.cli.core.w_states.initialize(mode, bstates, _bstate_file, tstates, _tstate_file)
```

6.1.15 w_eddist

usage:

```
w_eddist [-h] [-r RCFILE] [--quiet | --verbose | --debug] [--version]
          [--max-queue-length MAX_QUEUE_LENGTH] [-b BINEXPR] [-C] [--loose] --
↪istate ISTATE
          --fstate FSTATE [--first-iter ITER_START] [--last-iter ITER_STOP] [-k
↪KINETICS]
          [-o OUTPUT] [--serial | --parallel | --work-manager WORK_MANAGER]
          [--n-workers N_WORKERS] [--zmq-mode MODE] [--zmq-comm-mode COMM_MODE]
          [--zmq-write-host-info INFO_FILE] [--zmq-read-host-info INFO_FILE]
          [--zmq-upstream-rr-endpoint ENDPOINT] [--zmq-upstream-ann-endpoint
↪ENDPOINT]
          [--zmq-downstream-rr-endpoint ENDPOINT] [--zmq-downstream-ann-endpoint
↪ENDPOINT]
          [--zmq-master-heartbeat MASTER_HEARTBEAT] [--zmq-worker-heartbeat WORKER_
↪HEARTBEAT]
          [--zmq-timeout-factor FACTOR] [--zmq-startup-timeout STARTUP_TIMEOUT]
          [--zmq-shutdown-timeout SHUTDOWN_TIMEOUT]
```

Calculate time-resolved transition-event duration distribution from kinetics results

6.1.15.1 Source data

Source data is collected from the results of ‘w_kinetics trace’ (see w_kinetics trace –help for more information on generating this dataset).

6.1.15.2 Histogram binning

By default, histograms are constructed with 100 bins in each dimension. This can be overridden by specifying -b/-bins, which accepts a number of different kinds of arguments:

- a single integer N
N uniformly spaced bins will be used in each dimension.
- a sequence of integers N1,N2,... (comma-separated)
N1 uniformly spaced bins will be used for the first dimension, N2 for the second, and so on.
- a list of lists [[B11, B12, B13, ...], [B21, B22, B23, ...], ...]
The bin boundaries B11, B12, B13, ... will be used for the first dimension, B21, B22, B23, ... for the second dimension, and so on. These bin boundaries need not be uniformly spaced. These expressions will be evaluated with Python's `eval` construct, with `np` available for use [e.g. to specify bins using `np.arange()`].

The first two forms (integer, list of integers) will trigger a scan of all data in each dimension in order to determine the minimum and maximum values, which may be very expensive for large datasets. This can be avoided by explicitly providing bin boundaries using the list-of-lists form.

Note that these bins are *NOT* at all related to the bins used to drive WE sampling.

6.1.15.3 Output format

The output file produced (specified by `-o/--output`, defaulting to “`pdist.h5`”) may be fed to `plothist` to generate plots (or appropriately processed text or HDF5 files) from this data. In short, the following datasets are created:

```
``histograms``
  Normalized histograms. The first axis corresponds to iteration, and
  remaining axes correspond to dimensions of the input dataset.

``/binbounds_0``
  Vector of bin boundaries for the first (index 0) dimension. Additional
  datasets similarly named (/binbounds_1, /binbounds_2, ...) are created
  for additional dimensions.

``/midpoints_0``
  Vector of bin midpoints for the first (index 0) dimension. Additional
  datasets similarly named are created for additional dimensions.

``n_iter``
  Vector of iteration numbers corresponding to the stored histograms (i.e.
  the first axis of the ``histograms`` dataset).
```

6.1.15.4 Subsequent processing

The output generated by this program (`-o/--output`, default “`pdist.h5`”) may be plotted by the `plothist` program. See `plothist --help` for more information.

6.1.15.5 Parallelization

This tool supports parallelized binning, including reading of input data. Parallel processing is the default. For simple cases (reading pre-computed input data, modest numbers of segments), serial processing (`--serial`) may be more efficient.

6.1.15.6 Command-line options

optional arguments:

```
-h, --help          show this help message and exit
-b BINEXPR, --bins BINEXPR
                    Use BINEXPR for bins. This may be an integer, which will be used
    ↪ for each      dimension of the progress coordinate; a list of integers
    ↪ (formatted as [n1,n2,...]) which will use n1 bins for the first dimension, n2
    ↪ for the second dimension, and so on; or a list of lists of boundaries (formatted
```

(continues on next page)

(continued from previous page)

```

→as [[a1, a2, ..., [b1, b2, ...], ... ]], which will use [a1, a2, ...] as bin
→boundaries for the first dimension, [b1, b2, ...] as bin boundaries for the
→second dimension, and so on. (Default: 100 bins in each dimension.)
-C, --compress Compress histograms. May make storage of higher-dimensional
→histograms more tractable, at the (possible extreme) expense of increased analysis
→time. (Default: no compression.)
--loose Ignore values that do not fall within bins. (Risky, as this can
→make buggy bin boundaries appear as reasonable data. Only use if you are sure of
→your bin boundary specification.)
--istate ISTATE Initial state defining transition event
--fstate FSTATE Final state defining transition event

```

general options:

```

-r RCFILE, --rcfile RCFILE use RCFILE as the WEST run-time configuration file (default: west.
→cfg)
--quiet emit only essential information
--verbose emit extra information
--debug enable extra checks and emit copious information
--version show program's version number and exit

```

parallelization options:

--max-queue-length MAX_QUEUE_LENGTH Maximum number of tasks that can be queued.
Useful to limit RAM use for tasks that have very large requests/response.
Default: no limit.

iteration range options:

```

--first-iter ITER_START Iteration to begin analysis (default: 1)
--last-iter ITER_STOP Iteration to end analysis

```

input/output options:

```

-k KINETICS, --kinetics KINETICS Populations and transition rates (including evolution) are stored
→in KINETICS (default: kintrace.h5).
-o OUTPUT, --output OUTPUT Store results in OUTPUT (default: eddist.h5).

```

parallelization options:

--serial run in serial mode

- parallel** run in parallel mode (using processes)
- work-manager WORK_MANAGER** use the given work manager for parallel task distribution. Available work managers are ('serial', 'threads', 'processes', 'zmq'); default is 'processes'
- n-workers N_WORKERS** Use up to N_WORKERS on this host, for work managers which support this option. Use 0 for a dedicated server. (Ignored by work managers which do not support this option.)

options for ZeroMQ ("zmq") work manager (master or node):

```
--zmq-mode MODE      Operate as a master (server) or a node (workers/client). "server"
↳ is a deprecated synonym for "master" and "client" is a deprecated
↳ synonym for "node".
--zmq-comm-mode COMM_MODE Use the given communication mode -- TCP or IPC (Unix-domain) --
↳ sockets for communication within a node. IPC (the default) may be more
↳ efficient but is not available on (exceptionally rare) systems without node-local
↳ storage (e.g. /tmp); on such systems, TCP may be used instead.
--zmq-write-host-info INFO_FILE Store hostname and port information needed to connect to this
↳ instance in INFO_FILE. This allows the master and nodes assisting in
↳ coordinating the communication of other nodes to choose ports randomly. Downstream
↳ nodes read this file with --zmq-read-host-info and know where how to connect.
--zmq-read-host-info INFO_FILE Read hostname and port information needed to connect to the master
↳ (or other coordinating node) from INFO_FILE. This allows the master and
↳ nodes assisting in coordinating the communication of other nodes to choose ports
↳ randomly, writing that information with --zmq-write-host-info for this
↳ instance to read.
--zmq-upstream-rr-endpoint ENDPOINT ZeroMQ endpoint to which to send request/response (task and
↳ result) traffic toward the master.
--zmq-upstream-ann-endpoint ENDPOINT ZeroMQ endpoint on which to receive announcement (heartbeat and
↳ shutdown notification) traffic from the master.
--zmq-downstream-rr-endpoint ENDPOINT ZeroMQ endpoint on which to listen for request/response (task and
↳ result) traffic from subsidiary workers.
--zmq-downstream-ann-endpoint ENDPOINT
```

(continues on next page)

(continued from previous page)

```

↪ shutdown                ZeroMQ endpoint on which to send announcement (heartbeat and
--zmq-master-heartbeat MASTER_HEARTBEAT
                           notification) traffic toward workers.
                           Every MASTER_HEARTBEAT seconds, the master announces its presence.
↪ to workers.
--zmq-worker-heartbeat WORKER_HEARTBEAT
                           Every WORKER_HEARTBEAT seconds, workers announce their presence to
↪ the master.
--zmq-timeout-factor FACTOR
                           Scaling factor for heartbeat timeouts. If the master doesn't hear
↪ from a worker
                           in WORKER_HEARTBEAT*FACTOR, the worker is assumed to have crashed.
↪ If a worker
                           doesn't hear from the master in MASTER_HEARTBEAT*FACTOR seconds,
↪ the master is
                           assumed to have crashed. Both cases result in shutdown.
--zmq-startup-timeout STARTUP_TIMEOUT
                           Amount of time (in seconds) to wait for communication between the
↪ master and at
                           least one worker. This may need to be changed on very large,
↪ heavily-loaded
                           computer systems that start all processes simultaneously.
--zmq-shutdown-timeout SHUTDOWN_TIMEOUT
                           Amount of time (in seconds) to wait for workers to shut down.

```

6.1.15.7 westpa.cli.tools.w_eddist module

class westpa.cli.tools.w_eddist.WESTParallelTool(wm_env=None)

Bases: [WESTTool](#)

Base class for command-line tools parallelized with wwmgr. This automatically adds and processes wwmgr command-line arguments and creates a work manager at self.work_manager.

make_parser_and_process(prog=None, usage=None, description=None, epilog=None, args=None)

A convenience function to create a parser, call add_all_args(), and then call process_all_args(). The argument namespace is returned.

add_args(parser)

Add arguments specific to this tool to the given argparse parser.

process_args(args)

Take argparse-processed arguments associated with this tool and deal with them appropriately (setting instance variables, etc)

go()

Perform the analysis associated with this tool.

main()

A convenience function to make a parser, parse and process arguments, then run self.go() in the master process.

class westpa.cli.tools.w_eddist.**ProgressIndicatorComponent**

Bases: [*WESTToolComponent*](#)

add_args(*parser*)

Add arguments specific to this component to the given argparse parser.

process_args(*args*)

Take argparse-processed arguments associated with this component and deal with them appropriately (setting instance variables, etc)

westpa.cli.tools.w_eddist.**histnd**(*values, binbounds, weights=1.0, out=None, binbound_check=True, ignore_out_of_range=False*)

Generate an N-dimensional PDF (or contribution to a PDF) from the given values. *binbounds* is a list of arrays of boundary values, with one entry for each dimension (*values* must have as many columns as there are entries in *binbounds*) *weight*, if provided, specifies the weight each value contributes to the histogram; this may be a scalar (for equal weights for all values) or a vector of the same length as *values* (for unequal weights). If *binbound_check* is *True*, then the boundaries are checked for strict positive monotonicity; set to *False* to shave a few microseconds if you know your bin boundaries to be monotonically increasing.

westpa.cli.tools.w_eddist.**normhistnd**(*hist, binbounds*)

Normalize the N-dimensional histogram *hist* with corresponding bin boundaries *binbounds*. Modifies *hist* in place and returns the normalization factor used.

class westpa.cli.tools.w_eddist.**DurationDataset**(*dataset, mask, iter_start=1*)

Bases: object

A facade for the 'dsspec' dataclass that incorporates the mask into *get_iter_data* method

get_iter_data(*n_iter*)

westpa.cli.tools.w_eddist.**isiterable**(*x*)

class westpa.cli.tools.w_eddist.**WEDDist**

Bases: [*WESTParallelTool*](#)

prog = 'w_eddist'


```

description = 'Calculate time-resolved transition-event duration distribution from
kinetics results\n\n
n-----\n
nSource
data\n-----\n
n\nSource data is collected from the results of '\w_kinetics trace\' (see w_kinetics
trace --help for\nmore information on generating this dataset).\n\n
n-----\n
nHistogram
binning\n
n-----\n\nBy
default, histograms are constructed with 100 bins in each dimension. This\ncan be
overridden by specifying -b/--bins, which accepts a number of different\nkinds of
arguments:\n\n
n a single integer N\n N uniformly spaced bins will be used in each
dimension.\n\n
n a sequence of integers N1,N2,... (comma-separated)\n N1 uniformly
spaced bins will be used for the first dimension, N2 for the\n second, and so
on.\n\n
n a list of lists [[B11, B12, B13, ...], [B21, B22, B23, ...], ...]\n The bin
boundaries B11, B12, B13, ... will be used for the first dimension,\n B21, B22, B23,
... for the second dimension, and so on. These bin\n boundaries need not be
uniformly spaced. These expressions will be\n evaluated with Python\'s ``eval``
construct, with ``np`` available for\n use [e.g. to specify bins using
np.arange()].\n\n
The first two forms (integer, list of integers) will trigger a scan
of all\n data in each dimension in order to determine the minimum and maximum
values,\n which may be very expensive for large datasets. This can be avoided
by\n explicitly providing bin boundaries using the list-of-lists form.\n\n
Note that these bins are *NOT* at all related to the bins used to drive WE\n sampling.\n\n
n-----\n
nOutput
format\n
n-----\n\n
n\nThe output file produced (specified by -o/--output, defaulting to "pdist.h5")\nmay
be fed to plothist to generate plots (or appropriately processed text or\nHDF5
files) from this data. In short, the following datasets are created:\n\n
``histograms``\n Normalized histograms. The first axis corresponds to iteration,
and\n remaining axes correspond to dimensions of the input dataset.\n\n
``/binbounds_0``\n Vector of bin boundaries for the first (index 0) dimension.
Additional\n datasets similarly named (/binbounds_1, /binbounds_2, ...) are
created\n for additional dimensions.\n\n
``/midpoints_0``\n Vector of bin midpoints
for the first (index 0) dimension. Additional\n datasets similarly named are created
for additional dimensions.\n\n
``n_iter``\n Vector of iteration numbers
corresponding to the stored histograms (i.e.\n the first axis of the ``histograms``
dataset).\n\n
n-----\n
nSubsequent
processing\n
n-----\n\n
n\nThe output generated by this program (-o/--output, default "pdist.h5") may
be\n plotted by the ``plothist`` program. See ``plothist --help`` for
more\n information.\n\n
n-----\n
nParallelization\n
n-----\n\n
n\nThis tool supports parallelized binning, including reading of input data.\nParallel
processing is the default. For simple cases (reading pre-computed\ninput data,
modest numbers of segments), serial processing (--serial) may be\nmore efficient.\n\n
n\n
n-----\n
n\n'

```

add_args(*parser*)

Add arguments specific to this tool to the given argparse parser.

process_args(*args*)

Take argparse-processed arguments associated with this tool and deal with them appropriately (setting instance variables, etc)

go()

Perform the analysis associated with this tool.

static parse_binspec(*binspec*)

construct_bins(*bins*)

Construct bins according to *bins*, which may be:

- 1) A scalar integer (for that number of bins in each dimension)
- 2) A sequence of integers (specifying number of bins for each dimension)
- 3) A sequence of sequences of bin boundaries (specifying boundaries for each dimension)

Sets `self.binbounds` to a list of arrays of bin boundaries appropriate for passing to `fasthist.histnd`, along with `self.midpoints` to the midpoints of the bins.

scan_data_shape()

scan_data_range()

Scan input data for range in each dimension. The number of dimensions is determined from the shape of the progress coordinate as of `self.iter_start`.

construct_histogram()

Construct a histogram using bins previously constructed with `construct_bins()`. The time series of histogram values is stored in `histograms`. Each histogram in the time series is normalized.

`westpa.cli.tools.w_eddist.entry_point()`

6.1.16 w_ntop

usage:

```
w_ntop [-h] [-r RCFILE] [--quiet | --verbose | --debug] [--version] [-W WEST_H5FILE]
        [--first-iter N_ITER] [--last-iter N_ITER] [-a ASSIGNMENTS] [-n COUNT] [-t TIMEPOINT]
        [--highweight | --lowweight | --random] [-o OUTPUT]
```

Select walkers from bins . An assignment file mapping walkers to bins at each timepoint is required (see `w_assign -help`` for further information on generating this file). By default, high-weight walkers are selected (hence the name `w_ntop`: select the N top-weighted walkers from each bin); however, minimum weight walkers and randomly-selected walkers may be selected instead.

6.1.16.1 Output format

The output file (-o/-output, by default “ntop.h5”) contains the following datasets:

```
``/n_iter`` [iteration]
  *(Integer)* Iteration numbers for each entry in other datasets.

``/n_segs`` [iteration][bin]
  *(Integer)* Number of segments in each bin/state in the given iteration.
  This will generally be the same as the number requested with
  ``--n/--count`` but may be smaller if the requested number of walkers
  does not exist.

``/seg_ids`` [iteration][bin][segment]
  *(Integer)* Matching segments in each iteration for each bin.
  For an iteration ``n_iter``, only the first ``n_iter`` entries are
  valid. For example, the full list of matching seg_ids in bin 0 in the
  first stored iteration is ``seg_ids[0][0][:n_segs[0]]``.

``/weights`` [iteration][bin][segment]
  *(Floating-point)* Weights for each matching segment in ``/seg_ids``.
```

6.1.16.2 Command-line arguments

optional arguments:

```
-h, --help          show this help message and exit
--highweight        Select COUNT highest-weight walkers from each bin.
--lowweight         Select COUNT lowest-weight walkers from each bin.
--random            Select COUNT walkers randomly from each bin.
```

general options:

```
-r RCFILE, --rcfile RCFILE
                        use RCFILE as the WEST run-time configuration file (default: west.
↳ cfg)
--quiet               emit only essential information
--verbose             emit extra information
--debug              enable extra checks and emit copious information
--version             show program's version number and exit
```

WEST input data options:

```
-W WEST_H5FILE, --west-data WEST_H5FILE
                        Take WEST data from WEST_H5FILE (default: read from the HDF5 file,
↳ specified in
                        west.cfg).
```

iteration range:

```
--first-iter N_ITER   Begin analysis at iteration N_ITER (default: 1).
--last-iter N_ITER    Conclude analysis with N_ITER, inclusive (default: last completed,
↳ iteration).
```

input options:

```
-a ASSIGNMENTS, --assignments ASSIGNMENTS
    Use assignments from the given ASSIGNMENTS file (default: assign.
    ↳h5).
```

selection options:

```
-n COUNT, --count COUNT
    Select COUNT walkers from each iteration for each bin (default: 1).
-t TIMEPOINT, --timepoint TIMEPOINT
    Base selection on the given TIMEPOINT within each iteration.
    ↳Default (-1)
    corresponds to the last timepoint.
```

output options:

```
-o OUTPUT, --output OUTPUT
    Write output to OUTPUT (default: ntop.h5).
```

6.1.16.3 westpa.cli.tools.w_ntop module

class westpa.cli.tools.w_ntop.WESTTool

Bases: *WESTToolComponent*

Base class for WEST command line tools

prog = None

usage = None

description = None

epilog = None

add_args(parser)

Add arguments specific to this tool to the given argparse parser.

process_args(args)

Take argparse-processed arguments associated with this tool and deal with them appropriately (setting instance variables, etc)

make_parser(prog=None, usage=None, description=None, epilog=None, args=None)

make_parser_and_process(prog=None, usage=None, description=None, epilog=None, args=None)

A convenience function to create a parser, call `add_all_args()`, and then call `process_all_args()`. The argument namespace is returned.

go()

Perform the analysis associated with this tool.

main()

A convenience function to make a parser, parse and process arguments, then call `self.go()`

class westpa.cli.tools.w_ntop.WESTDataReader

Bases: [WESTToolComponent](#)

Tool for reading data from WEST-related HDF5 files. Coordinates finding the main HDF5 file from west.cfg or command line arguments, caching of certain kinds of data (eventually), and retrieving auxiliary data sets from various places.

add_args(*parser*)

Add arguments specific to this component to the given argparse parser.

process_args(*args*)

Take argparse-processed arguments associated with this component and deal with them appropriately (setting instance variables, etc)

open(*mode='r'*)

close()

property weight_dsspec

property parent_id_dsspec

class westpa.cli.tools.w_ntop.IterRangeSelection(*data_manager=None*)

Bases: [WESTToolComponent](#)

Select and record limits on iterations used in analysis and/or reporting. This class provides both the user-facing command-line options and parsing, and the application-side API for recording limits in HDF5.

HDF5 datasets calculated based on a restricted set of iterations should be tagged with the following attributes:

first_iter

The first iteration included in the calculation.

last_iter

One past the last iteration included in the calculation.

iter_step

Blocking or sampling period for iterations included in the calculation.

add_args(*parser*)

Add arguments specific to this component to the given argparse parser.

process_args(*args, override_iter_start=None, override_iter_stop=None, default_iter_step=1*)

Take argparse-processed arguments associated with this component and deal with them appropriately (setting instance variables, etc)

iter_block_iter()

Return an iterable of (block_start,block_end) over the blocks of iterations selected by --first-iter/--last-iter/--step-iter.

n_iter_blocks()

Return the number of blocks of iterations (as returned by `iter_block_iter`) selected by --first-iter/--last-iter/--step-iter.

record_data_iter_range(*h5object, iter_start=None, iter_stop=None*)

Store attributes `iter_start` and `iter_stop` on the given HDF5 object (group/dataset)

record_data_iter_step(*h5object, iter_step=None*)

Store attribute `iter_step` on the given HDF5 object (group/dataset).

check_data_iter_range_least(*h5object*, *iter_start=None*, *iter_stop=None*)

Check that the given HDF5 object contains (as denoted by its *iter_start/iter_stop* attributes) data at least for the iteration range specified.

check_data_iter_range_equal(*h5object*, *iter_start=None*, *iter_stop=None*)

Check that the given HDF5 object contains (as denoted by its *iter_start/iter_stop* attributes) data exactly for the iteration range specified.

check_data_iter_step_conformant(*h5object*, *iter_step=None*)

Check that the given HDF5 object contains per-iteration data at an iteration stride suitable for extracting data with the given stride (in other words, the given *iter_step* is a multiple of the stride with which data was recorded).

check_data_iter_step_equal(*h5object*, *iter_step=None*)

Check that the given HDF5 object contains per-iteration data at an iteration stride the same as that specified.

slice_per_iter_data(*dataset*, *iter_start=None*, *iter_stop=None*, *iter_step=None*, *axis=0*)

Return the subset of the given dataset corresponding to the given iteration range and stride. Unless otherwise specified, the first dimension of the dataset is the one sliced.

iter_range(*iter_start=None*, *iter_stop=None*, *iter_step=None*, *dtype=None*)

Return a sequence for the given iteration numbers and stride, filling in missing values from those stored on self. The smallest data type capable of holding *iter_stop* is returned unless otherwise specified using the *dtype* argument.

class westpa.cli.tools.w_ntop.**ProgressIndicatorComponent**

Bases: [WESTToolComponent](#)

add_args(*parser*)

Add arguments specific to this component to the given argparse parser.

process_args(*args*)

Take argparse-processed arguments associated with this component and deal with them appropriately (setting instance variables, etc)

westpa.cli.tools.w_ntop.**seg_id_dtype**

alias of int64

westpa.cli.tools.w_ntop.**n_iter_dtype**

alias of uint32

westpa.cli.tools.w_ntop.**weight_dtype**

alias of float64

westpa.cli.tools.w_ntop.**assignments_list_to_table**(*nsecs*, *nbins*, *assignments*)

Convert a list of bin assignments (integers) to a boolean table indicating indicating if a given segment is in a given bin

class westpa.cli.tools.w_ntop.**WNTopTool**

Bases: [WESTTool](#)

prog = 'w_ntop'

```

description = 'Select walkers from bins . An assignment file mapping walkers
to\nbins at each timepoint is required (see ``w_assign --help`` for
further\ninformation on generating this file). By default, high-weight walkers
are\nselected (hence the name ``w_ntop``: select the N top-weighted walkers
from\neach bin); however, minimum weight walkers and randomly-selected walkers\nmay
be selected instead.\n\n\
n-----\
nOutput
format\
n-----\n\
nThe output file (-o/--output, by default "ntop.h5") contains the
following\ndatasets:\n\n ``/n_iter`` [iteration]\n *(Integer)* Iteration numbers for
each entry in other datasets.\n\n ``/n_segs`` [iteration][bin]\n *(Integer)* Number
of segments in each bin/state in the given iteration.\n This will generally be the
same as the number requested with\n ``--n/--count`` but may be smaller if the
requested number of walkers\n does not exist.\n\n ``/seg_ids``
[iteration][bin][segment]\n *(Integer)* Matching segments in each iteration for each
bin.\n For an iteration ``n_iter``, only the first ``n_iter`` entries are\n valid.
For example, the full list of matching seg_ids in bin 0 in the\n first stored
iteration is ``seg_ids[0][0][:n_segs[0]]``.\n\n ``/weights``
[iteration][bin][segment]\n *(Floating-point)* Weights for each matching segment in
``/seg_ids``.\n\n\
n-----\
nCommand-line
arguments\
n-----\n'

```

add_args(parser)

Add arguments specific to this tool to the given argparse parser.

process_args(args)

Take argparse-processed arguments associated with this tool and deal with them appropriately (setting instance variables, etc)

go()

Perform the analysis associated with this tool.

`westpa.cli.tools.w_ntop.entry_point()`

6.1.17 w_multi_west

The `w_multi_west` tool combines multiple WESTPA simulations into a single aggregate simulation to facilitate the analysis of the set of simulations. In particular, the tool creates a single `west.h5` file that contains all of the data from the `west.h5` files of the individual simulations. Each iteration `x` in the new file contains all of the segments from iteration `x` from each of the set of simulation, all normalized to the total weight.

6.1.17.1 Overview

usage:

```
w_multi_west [-h] [-m master] [-n sims] [--quiet | --verbose | --debug] [--version]
              [-W WEST_H5FILE] [-a aux] [--auxall] [--ibstates]
              [--serial | --parallel | --work-manager WORK_MANAGER] [--n-workers N_
↪WORKERS]
              [--zmq-mode MODE] [--zmq-comm-mode COMM_MODE] [--zmq-write-host-info INFO_
↪FILE]
              [--zmq-read-host-info INFO_FILE] [--zmq-upstream-rr-endpoint ENDPOINT]
              [--zmq-upstream-ann-endpoint ENDPOINT] [--zmq-downstream-rr-endpoint_
↪ENDPOINT]
              [--zmq-downstream-ann-endpoint ENDPOINT] [--zmq-master-heartbeat MASTER_
↪HEARTBEAT]
              [--zmq-worker-heartbeat WORKER_HEARTBEAT] [--zmq-timeout-factor FACTOR]
              [--zmq-startup-timeout STARTUP_TIMEOUT] [--zmq-shutdown-timeout SHUTDOWN_
↪TIMEOUT]
```

optional arguments:

```
-h, --help          show this help message and exit
```

General options::

-m, --master directory	Master path of simulations where all the smaller simulations are stored (default: Current Directory)
-n, --sims n	Number of simulation directories. Assumes leading zeros. (default: 0)
--quiet	emit only essential information
--verbose	emit extra information
--version	show program's version number and exit

6.1.17.2 Command-Line Options

See the [general command-line tool reference](#) for more information on the general options.

6.1.17.2.1 Input/output options

These arguments allow the user to specify where to read input simulation result data and where to output calculated progress coordinate probability distribution data.

Both input and output files are *hdf5* format:

```
-W, --west, --WEST_H5FILE file
    The name of the main .h5 file inside each simulation directory. (Default: west.h5)

-o, --output file
    Store this tool's output in file. (Default: multi.h5)

-a, --aux auxdata
    Name of additional auxiliary dataset to be combined. Can be called multiple times.
    (Default: None)

-aa, --auxall
    Combine all auxiliary datasets as labeled in ``west.h5`` in folder 01. (Default: False)

-nr, --no-reweight
    Do not perform reweighting. (Default: False)

-ib, --ibstates
    Attempt to combine ``ibstates`` dataset if the basis states are identical across
    all simulations. Needed when tracing with ``westpa.analysis``. (Default: False)
```

6.1.17.3 Examples

If you have five simulations, set up your directory such that you have five directories are named numerically with leading zeroes, and each directory contains a `west.h5` file. For this example, each `west.h5` also contains an auxiliary dataset called `RMSD`. If you run `ls`, you will see the following output:

```
01 02 03 04 05
```

To run the `w_multi_west` tool, do the following:

```
w_multi_west.py -m . -n 5 --aux=RMSD
```

If you used any custom `WESTSystem`, include that in the directory where you run the code.

To proceed in analyzing the aggregated simulation data as a single simulation, rename the output file `multi.h5` to `west.h5`.

6.1.17.4 westpa.cli.tools.w_multi_west module

class westpa.cli.tools.w_multi_west.WESTTool

Bases: *WESTToolComponent*

Base class for WEST command line tools

prog = None

usage = None

description = None

epilog = None

add_args(*parser*)

Add arguments specific to this tool to the given argparse parser.

process_args(*args*)

Take argparse-processed arguments associated with this tool and deal with them appropriately (setting instance variables, etc)

make_parser(*prog=None, usage=None, description=None, epilog=None, args=None*)

make_parser_and_process(*prog=None, usage=None, description=None, epilog=None, args=None*)

A convenience function to create a parser, call `add_all_args()`, and then call `process_all_args()`. The argument namespace is returned.

go()

Perform the analysis associated with this tool.

main()

A convenience function to make a parser, parse and process arguments, then call `self.go()`

westpa.cli.tools.w_multi_west.n_iter_dtype

alias of `uint32`

class westpa.cli.tools.w_multi_west.ProgressIndicatorComponent

Bases: *WESTToolComponent*

add_args(*parser*)

Add arguments specific to this component to the given argparse parser.

process_args(*args*)

Take argparse-processed arguments associated with this component and deal with them appropriately (setting instance variables, etc)

class westpa.cli.tools.w_multi_west.WESTMultiTool(*wm_env=None*)

Bases: *WESTParallelTool*

Base class for command-line tools which work with multiple simulations. Automatically parses for and gives commands to load multiple files.

make_parser_and_process(*prog=None, usage=None, description=None, epilog=None, args=None*)

A convenience function to create a parser, call `add_all_args()`, and then call `process_all_args()`. The argument namespace is returned.

parse_from_yaml(*yamlfilepath*)

Parse options from YAML input file. Command line arguments take precedence over options specified in the YAML hierarchy. TODO: add description on how YAML files should be constructed.

add_args(*parser*)

Add arguments specific to this tool to the given argparse parser.

exception NoSimulationsException

Bases: Exception

generate_file_list(*key_list*)

A convenience function which takes in a list of keys that are filenames, and returns a dictionary which contains all the individual files loaded inside of a dictionary keyed to the filename.

process_args(*args*)

Take argparse-processed arguments associated with this tool and deal with them appropriately (setting instance variables, etc)

go()

Perform the analysis associated with this tool.

main()

A convenience function to make a parser, parse and process arguments, then run self.go() in the master process.

westpa.cli.tools.w_multi_west.get_bin_mapper(*we_h5file, hashval*)

Look up the given hash value in the binning table, unpickling and returning the corresponding bin mapper if available, or raising KeyError if not.

westpa.cli.tools.w_multi_west.create_idtype_array(*input_array*)

Return a new array with the new istate_dtype while preserving old data.

class westpa.cli.tools.w_multi_west.WMultiWest

Bases: *WESTMultiTool*

prog = 'w_multi_west'

description = 'Tool designed to combine multiple WESTPA simulations while accounting for\nreweighting.\n'

n-----\n

nCommand-line

noptions\

n-----\n'

add_args(*parser*)

Add arguments specific to this tool to the given argparse parser.

open_files()

process_args(*args*)

Take argparse-processed arguments associated with this tool and deal with them appropriately (setting instance variables, etc)

total_number_of_walkers()

go()

Perform the analysis associated with this tool.

westpa.cli.tools.w_multi_west.entry_point()

6.1.18 w_red

usage:

```
w_red [-h] [-r RCFILE] [--quiet] [--verbose] [--version] [--max-queue-length MAX_QUEUE_
↪LENGTH]
        [--debug] [--terminal]
        [--serial | --parallel | --work-manager WORK_MANAGER] [--n-workers N_WORKERS]
        [--zmq-mode MODE] [--zmq-comm-mode COMM_MODE] [--zmq-write-host-info INFO_
↪FILE]
        [--zmq-read-host-info INFO_FILE] [--zmq-upstream-rr-endpoint ENDPOINT]
        [--zmq-upstream-ann-endpoint ENDPOINT] [--zmq-downstream-rr-endpoint_
↪ENDPOINT]
        [--zmq-downstream-ann-endpoint ENDPOINT] [--zmq-master-heartbeat MASTER_
↪HEARTBEAT]
        [--zmq-worker-heartbeat WORKER_HEARTBEAT] [--zmq-timeout-factor FACTOR]
        [--zmq-startup-timeout STARTUP_TIMEOUT] [--zmq-shutdown-timeout SHUTDOWN_
↪TIMEOUT]
```

optional arguments:

```
-h, --help          show this help message and exit
```

general options:

-r RCFILE, --rcfile RCFILE	use RCFILE as the WEST run-time configuration file (default: west.cfg)
--quiet	emit only essential information
--verbose	emit extra information
--version	show program's version number and exit

parallelization options:

```
--max-queue-length MAX_QUEUE_LENGTH
        Maximum number of tasks that can be queued. Useful to limit RAM_
↪use for tasks that
        have very large requests/response. Default: no limit.
```

parallelization options:

```
--serial          run in serial mode
--parallel        run in parallel mode (using processes)
--work-manager WORK_MANAGER
```

6.1.18.1 westpa.cli.tools.w_red module

`westpa.cli.tools.w_red.H5File`

alias of `File`

class `westpa.cli.tools.w_red.WESTParallelTool(wm_env=None)`

Bases: `WESTTool`

Base class for command-line tools parallelized with wwmgr. This automatically adds and processes wwmgr command-line arguments and creates a work manager at `self.work_manager`.

make_parser_and_process(*prog=None, usage=None, description=None, epilog=None, args=None*)

A convenience function to create a parser, call `add_all_args()`, and then call `process_all_args()`. The argument namespace is returned.

add_args(*parser*)

Add arguments specific to this tool to the given argparse parser.

process_args(*args*)

Take argparse-processed arguments associated with this tool and deal with them appropriately (setting instance variables, etc)

go()

Perform the analysis associated with this tool.

main()

A convenience function to make a parser, parse and process arguments, then run `self.go()` in the master process.

class `westpa.cli.tools.w_red.DurationCorrector(durations, weights, dtau, maxduration=None)`

Bases: `object`

static from_kinetics_file(*directh5, istate, fstate, dtau, n_iters=None*)

property event_duration_histogram

property cumulative_event_duration_histogram

correction(*iters, freqs=None*)

Return the correction factor

```
t=theta tau=t |
    ||
    || ~ |
    || f(tau) dtau dt | * maxduration
    ||
    t=0 tau=0 |
```

[_]

where

$\sim^{\wedge} f(\tau)$ is proportional to $f(\tau)/(\theta-\tau)$, and is normalized to

\wedge

integrate to 1, and $f(\tau)$ is sum of the weights of walkers with duration time τ .

```
westpa.cli.tools.w_red.get_raw_rates(directh5, istate, fstate, n_iters=None)
```

```
westpa.cli.tools.w_red.calc_avg_rate(directh5_path, istate, fstate, **kwargs)
```

Return the raw or RED-corrected rate constant with the confidence interval.

Parameters

- **nstiter** (duration of each iteration (number of steps)) –
- **ntpr** (report interval (number of steps)) –

```
westpa.cli.tools.w_red.calc_rates(directh5_path, istate, fstate, **kwargs)
```

Return the raw and RED-corrected rate constants vs. iterations. This code is faster than calling `calc_rate()` iteratively

Parameters

- **nstiter** (duration of each iteration (number of steps)) –
- **ntpr** (report interval (number of steps)) –

```
class westpa.cli.tools.w_red.RateCalculator(directh5, istate, fstate, assignh5=None, **kwargs)
```

Bases: `object`

property `conditional_fluxes`

property `populations`

property `tau`

property `dtau`

property `istate`

property `fstate`

property `n_iters`

`calc_rate(i_iter=None, red=False, **kwargs)`

`calc_rates(n_iters=None, **kwargs)`

```
class westpa.cli.tools.w_red.WRed
```

Bases: `WESTParallelTool`

`prog = 'w_red'`

`description = 'Apply the RED scheme to estimate steady-state WE fluxes from\nshorter trajectories.\n\n'`

`n-----\n`

`nSource`

`data\n-----\n`

`n\nSource data is provided as a w_ipa "scheme" which is typically defined\nin the west.cfg file. For instance, if a user wishes to estimate RED\nfluxes for a scheme named "DEFAULT" that argument would be provided\nto w_red and WRed would estimate RED fluxes based off of the data\ncontained in the assign.h5 and direct.h5 files in ANALYSIS/DEFAULT.\n\n'`

`go()`

Perform the analysis associated with this tool.

```
westpa.cli.tools.w_red.entry_point()
```

6.1.19 plothist

Use the `plothist` tool to plot the results of `w_pdist`. This tool uses an `hdf5` file as its input (i.e. the output of another analysis tool), and outputs a `pdf` image.

The `plothist` tool operates in one of three (mutually exclusive) plotting modes:

- **evolution:** Plots the relevant data as a time evolution over specified number of simulation iterations
- **average:** Plots the relevant data as a time average over a specified number of iterations
- **instant:** Plots the relevant data for a single specified iteration

6.1.19.1 Overview

The basic usage, independent of plotting mode, is as follows:

usage:

```
| ``plothist [-h] [-r RCFILE] [--quiet | --verbose | --debug] [--version]``  
| ``{instant,average,evolution} input ...``
```

Note that the user must specify a plotting mode (i.e. ‘instant’, ‘average’, or ‘evolution’) and an input file, `input`.

Therefore, this tool is always called as:

```
plothist mode input_file [``other`` ``options``]
```

6.1.19.1.1 ‘instant’ mode

usage:

```
| ``plothist instant [-h] input [-o PLOT_OUTPUT]``  
| ``[--hdf5-output HDF5_OUTPUT] [--text-output TEXT_  
↪OUTPUT]``  
| ``[--title TITLE] [--range RANGE] [--linear | --  
↪energy | --log10]``  
| ``[--iter N_ITER] ``  
| ``[DIMENSION] [ADDTLDIM]``
```

6.1.19.1.2 ‘average’ mode

usage:

```
| ``plothist average [-h] input [-o PLOT_OUTPUT]``  
| ``[--hdf5-output HDF5_OUTPUT] [--text-output TEXT_  
↪OUTPUT]``  
| ``[--title TITLE] [--range RANGE] [--linear | --  
↪energy | --log10]``  
| ``[--first-iter N_ITER] [--last-iter N_  
↪ITER] ``  
| ``[DIMENSION] [ADDTLDIM]``
```

6.1.19.1.3 ‘evolution’ mode

usage:

```
| ``plothist evolution [-h] input [-o PLOT_OUTPUT]``  
| ``                                     [--hdf5-output HDF5_OUTPUT]``  
| ``                                     [--title TITLE] [--range RANGE] [--linear | --  
↪energy | --log10]``  
| ``                                     [--first-iter N_ITER] [--last-iter N_ITER]``  
| ``                                     [--step- ``  
↪iter STEP]``  
| ``                                     [DIMENSION]``
```

6.1.19.2 Command-Line Options

See the [command-line tool index](#) for more information on the general options.

Unless specified (as a **Note** in the command-line option description), the command-line options below are shared for all three plotting modes

6.1.19.2.1 Input/output options

No matter the mode, an input *hdf5* file must be specified. There are three possible outputs that are mode or user-specified: A text file, an *hdf5* file, and a pdf image.

Specifying input file

`input`

Specify the input *hdf5* file ‘input’. This is the output file from a previous analysis tool (e.g. ‘pdist.h5’)

Output plot pdf file

``-o ‘plot_output’, -plot_output ‘plot_output’``

Specify the name of the pdf plot image output (**Default:** ‘hist.pdf’). **Note:** You can suppress plotting entirely by specifying an empty string as *plot_output* (i.e. -o ‘ ’ or --plot_output ‘ ’)

Additional output options

Note: *plothist* provides additional, optional arguments to output the data points used to construct the plot:

``--hdf5-output ‘hdf5_output’``

Output plot data *hdf5* file ‘hdf5_output’ (**Default:** No *hdf5* output file)

``--text-output ‘text_output’``

Output plot data as a text file named ‘text_output’ (**Default:** No text output file) **Note:** This option is only available for 1 dimensional histogram plots (that is, ‘average’ and ‘instant’ modes only)

6.1.19.2.2 Plotting options

The following options allow the user to specify a plot title, the type of plot (i.e. energy or probability distribution), whether to apply a log transformation to the data, and the range of data values to include.

`--title "title"`

Optionally specify a title, `"title"`, for the plot (**Default:** No title)

`--range "<nowiki>'</nowiki>LB, UB<nowiki>'</nowiki>"`

Optionally specify the data range to be plotted as "LB, UB" (e.g. `--range "-1, 10"` - note that the quotation marks are necessary if specifying a negative bound). For 1 dimensional histograms, the range affects the y axis. For 2 dimensional plots (e.g. evolution plot with 1 dimensional progress coordinate), it corresponds to the range of the color bar

Mutually exclusive plotting options

The following three options determine how the plotted data is represented (**Default:** `--energy`)

`--energy`

Plots the probability distribution on an inverted natural log scale (i.e. $-\ln[P(x)]$), corresponding to the free energy (**Default**)

`--linear`

Plots the probability distribution function as a linear scale

`--log10`

Plots the (base-10) logarithm of the probability distribution

6.1.19.2.3 Iteration selection options

Depending on plotting mode, you can select either a range or a single iteration to plot.

`--instant` mode only:

`--iter "n_iter"`

Plot the distribution for iteration `"n_iter"` (**Default:** Last completed iteration)

`--average` and `--evolution` modes only:

`--first-iter "first_iter"`

Begin averaging or plotting at iteration `"first_iter"` (**Default:** 1)

`--last-iter "last_iter"`

Average or plot up to and including `"last_iter"` (**Default:** Last completed iteration)

`--evolution` mode only:

`--iter_step "n_step"`

Average every `"n_step"` iterations together when plotting in `'evolution'` mode (**Default:** 1 - i.e. plot each iteration)

6.1.19.2.4 Specifying progress coordinate dimension

For progress coordinates with dimensions greater than 1, you can specify the dimension of the progress coordinate to use, the of progress coordinate values to include, and the progress coordinate axis label with a single positional argument:

```dimension```

Specify 'dimension' as `'int[:[LB,UB]:label]'`, where 'int' specifies the dimension (starting at 0), and, optionally, 'LB,UB' specifies the lower and upper range bounds, and/or 'label' specifies the axis label (**Default:** `int = 0`, full range, default label is 'dimension int'; e.g 'dimension 0')

For 'average' and 'instant' modes, you can plot two dimensions at once using a color map if this positional argument is specified:

```addtl_dimension```

Specify the other dimension to include as 'addtl_dimension'

6.1.19.3 Examples

These examples assume the input file is created using `w_pdist` and is named 'pdist.h5'

6.1.19.3.1 Basic plotting

Plot the energy ($-\ln(P(x))$) for the last iteration

```
plothist instant pdist.h5
```

Plot the evolution of the log10 of the probability distribution over all iterations

```
``plothist evolution pdist.h5 -log10``
```

Plot the average linear probability distribution over all iterations

```
plothist average pdist.h5 --linear
```

6.1.19.3.2 Specifying progress coordinate

Plot the average probability distribution as the energy, label the x-axis 'pcoord', over the entire range of the progress coordinate

```
plothist average pdist.h5 0::pcoord
```

Same as above, but only plot the energies for with progress coordinate between 0 and 10

```
plothist average pdist.h5 '0:0,10:pcoord'
```

(Note: the quotes are needed if specifying a range that includes a negative bound)

(For a simulation that uses at least 2 progress coordinates) plot the probability distribution for the 5th iteration, representing the first two progress coordinates as a heatmap

```
plothist instant pdist.h5 0 1 --iter 5 --linear
```

6.1.19.4 westpa.cli.tools.plothist module

class westpa.cli.tools.plothist.NonUniformImage(ax, *, interpolation='nearest', **kwargs)

Bases: AxesImage

Parameters

- **ax** (~matplotlib.axes.Axes) – The axes the image will belong to.
- **interpolation** ({'nearest', 'bilinear'}, default: 'nearest') – The interpolation scheme used in the resampling.
- ****kwargs** – All other keyword arguments are identical to those of *AxesImage*.

mouseover = False

make_image(renderer, magnification=1.0, unsampled=False)

Normalize, rescale, and colormap this image's data for rendering using *renderer*, with the given *magnification*.

If *unsampled* is True, the image will not be scaled, but an appropriate affine transformation will be returned instead.

Returns

- **image** ((M, N, 4) *numpy.uint8* array) – The RGBA image, resampled unless *unsampled* is True.
- **x, y** (*float*) – The upper left corner where the image should be drawn, in pixel space.
- **trans** (~matplotlib.transforms.Affine2D) – The affine transformation from image to pixel space.

set_data(x, y, A)

Set the grid for the pixel centers, and the pixel values.

Parameters

- **x** (*1D array-like*) – Monotonic arrays of shapes (N,) and (M,), respectively, specifying pixel centers.
- **y** (*1D array-like*) – Monotonic arrays of shapes (N,) and (M,), respectively, specifying pixel centers.
- **A** (*array-like*) – (M, N) ~*numpy.ndarray* or masked array of values to be colormapped, or (M, N, 3) RGB array, or (M, N, 4) RGBA array.

set_array(*args)

Retained for backwards compatibility - use *set_data* instead.

Parameters

A (*array-like*) –

set_interpolation(s)

Parameters

s ({'nearest', 'bilinear'} or None) – If None, use :rc:`image.interpolation`.

get_extent()

Return the image extent as tuple (left, right, bottom, top).

set_filternorm(filternorm)

Set whether the resize filter normalizes the weights.

See help for ~*Axes.imshow*.

Parameters**filternorm** (*bool*) –**set_filterrad**(*filterrad*)

Set the resize filter radius only applicable to some interpolation schemes – see help for imshow

Parameters**filterrad** (*positive float*) –**set_norm**(*norm*)

Set the normalization instance.

Parameters**norm** (*.Normalize or str or None*) –**Notes**

If there are any colorbars using the mappable for this norm, setting the norm of the mappable will reset the norm, locator, and formatters on the colorbar to default.

set_cmap(*cmap*)

Set the colormap for luminance data.

Parameters**cmap** (*.Colormap or str or None*) –

set(*, *agg_filter*=<UNSET>, *alpha*=<UNSET>, *animated*=<UNSET>, *array*=<UNSET>, *clim*=<UNSET>, *clip_box*=<UNSET>, *clip_on*=<UNSET>, *clip_path*=<UNSET>, *cmap*=<UNSET>, *data*=<UNSET>, *extent*=<UNSET>, *filternorm*=<UNSET>, *filterrad*=<UNSET>, *gid*=<UNSET>, *in_layout*=<UNSET>, *interpolation*=<UNSET>, *interpolation_stage*=<UNSET>, *label*=<UNSET>, *mouseover*=<UNSET>, *norm*=<UNSET>, *path_effects*=<UNSET>, *picker*=<UNSET>, *rasterized*=<UNSET>, *resample*=<UNSET>, *sketch_params*=<UNSET>, *snap*=<UNSET>, *transform*=<UNSET>, *url*=<UNSET>, *visible*=<UNSET>, *zorder*=<UNSET>)

Set multiple properties at once.

Supported properties are

Properties:

agg_filter: a filter function, which takes a (m, n, 3) float array and a dpi value, and returns a (m, n, 3) array and two offsets from the bottom left corner of the image
alpha: float or 2D array-like or None
animated: bool
array: unknown
clim: (vmin: float, vmax: float)
clip_box: ~matplotlib.transforms.BboxBase or None
clip_on: bool
clip_path: Patch or (Path, Transform) or None
cmap: unknown
data: unknown
extent: 4-tuple of float
figure: ~matplotlib.figure.Figure
filternorm: unknown
filterrad: unknown
gid: str
in_layout: bool
interpolation: {'nearest', 'bilinear'} or None
interpolation_stage: {'data', 'rgba'} or None
label: object
mouseover: bool
norm: unknown
path_effects: list of .AbstractPathEffect
picker: None or bool or float or callable
rasterized: bool
resample: bool or None
sketch_params: (scale: float, length: float, randomness: float)
snap: bool or None
transform: ~matplotlib.transforms.Transform
url: str
visible: bool
zorder: float

class westpa.cli.tools.plothist.WESTMasterCommandBases: [WESTTool](#)

Base class for command-line tools that employ subcommands

subparsers_title = None**subcommands** = None**include_help_command** = True

add_args(*parser*)

Add arguments specific to this tool to the given argparse parser.

process_args(*args*)

Take argparse-processed arguments associated with this tool and deal with them appropriately (setting instance variables, etc)

go()

Perform the analysis associated with this tool.

class westpa.cli.tools.plothist.**WESTSubcommand**(*parent*)

Bases: [WESTToolComponent](#)

Base class for command-line tool subcommands. A little sugar for making this more uniform.

subcommand = None

help_text = None

description = None

add_to_subparsers(*subparsers*)

go()

property work_manager

The work manager for this tool. Raises AttributeError if this is not a parallel tool.

westpa.cli.tools.plothist.**normhistnd**(*hist*, *binbounds*)

Normalize the N-dimensional histogram *hist* with corresponding bin boundaries *binbounds*. Modifies *hist* in place and returns the normalization factor used.

westpa.cli.tools.plothist.**get_object**(*object_name*, *path=None*)

Attempt to load the given object, using additional path information if given.

westpa.cli.tools.plothist.**sum_except_along**(*array*, *axes*)

Reduce the given array by addition over all axes except those listed in the scalar or iterable *axes*

class westpa.cli.tools.plothist.**PlotHistBase**(*parent*)

Bases: [WESTSubcommand](#)

add_args(*parser*)

Add arguments specific to this component to the given argparse parser.

process_args(*args*)

Take argparse-processed arguments associated with this component and deal with them appropriately (setting instance variables, etc)

parse_dimspec(*dimspec*)

parse_range(*rangespec*)

class westpa.cli.tools.plothist.**PlotSupports2D**(*parent*)

Bases: [PlotHistBase](#)

add_args(*parser*)

Add arguments specific to this component to the given argparse parser.

process_args(args)

Take argparse-processed arguments associated with this component and deal with them appropriately (setting instance variables, etc)

class westpa.cli.tools.plothist.**InstantPlotHist**(parent)

Bases: [*PlotSupports2D*](#)

subcommand = 'instant'

help_text = 'plot probability distribution for a single WE iteration'

description = 'Plot a probability distribution for a single WE iteration. The probability distribution must have been previously extracted with ``w_pdist`` (or, at least, must be compatible with the output format of ``w_pdist``; see ``w_pdist --help`` for more information).\\n'

add_args(parser)

Add arguments specific to this component to the given argparse parser.

process_args(args)

Take argparse-processed arguments associated with this component and deal with them appropriately (setting instance variables, etc)

do_instant_plot_1d()

Plot the histogram for iteration self.n_iter

do_instant_plot_2d()

Plot the histogram for iteration self.n_iter

go()

class westpa.cli.tools.plothist.**AveragePlotHist**(parent)

Bases: [*PlotSupports2D*](#)

subcommand = 'average'

help_text = 'plot average of a probability distribution over a WE simulation'

description = 'Plot a probability distribution averaged over multiple iterations. The probability distribution must have been previously extracted with ``w_pdist`` (or, at least, must be compatible with the output format of ``w_pdist``; see ``w_pdist --help`` for more information).\\n'

add_args(parser)

Add arguments specific to this component to the given argparse parser.

process_args(args)

Take argparse-processed arguments associated with this component and deal with them appropriately (setting instance variables, etc)

do_average_plot_1d()

Plot the average histogram for iterations self.iter_start to self.iter_stop

do_average_plot_2d()

Plot the histogram for iteration self.n_iter

go()

```

class westpa.cli.tools.plothist.EvolutionPlotHist(parent)
    Bases: PlotHistBase

    subcommand = 'evolution'

    help_text = 'plot evolution of a probability distribution over the course of a WE
simulation'

    description = 'Plot a probability distribution as it evolves over iterations.
The\nprobability distribution must have been previously extracted with
``w_pdist``\n(or, at least, must be compatible with the output format of
``w_pdist``; see\n``w_pdist --help`` for more information).\n'

    add_args(parser)
        Add arguments specific to this component to the given argparse parser.

    process_args(args)
        Take argparse-processed arguments associated with this component and deal with them appropriately (set-
ting instance variables, etc)

    go()
        Plot the evolution of the histogram for iterations self.iter_start to self.iter_stop

class westpa.cli.tools.plothist.PlotHistTool
    Bases: WESTMasterCommand

    prog = 'plohist'

    subparsers_title = 'plotting modes'

    subcommands = [<class 'westpa.cli.tools.plothist.InstantPlotHist'>, <class
'westpa.cli.tools.plothist.AveragePlotHist'>, <class
'westpa.cli.tools.plothist.EvolutionPlotHist'>]

    description = 'Plot probability density functions (histograms) generated by w_pdist
or other\nprograms conforming to the same output format. This program operates in
one of\nthree modes:\n\n instant\n Plot 1-D and 2-D histograms for an individual
iteration. See\n ``plohist instant --help`` for more information.\n\n average\n
Plot 1-D and 2-D histograms, averaged over several iterations. See\n ``plohist
average --help`` for more information.\n\n evolution\n Plot the time evolution 1-D
histograms as waterfall (heat map) plots.\n See ``plohist evolution --help`` for
more information.\n\n This program takes the output of ``w_pdist`` as input (see
``w_pdist --help``\nfor more information), and can generate any kind of graphical
output that\nmatplotlib supports.\n\n\
n-----\
nCommand-line
options\
n-----\n'

westpa.cli.tools.plothist.entry_point()

```

6.1.20 ploterr

usage:

```
ploterr [-h] [-r RCFILE] [--quiet | --verbose | --debug] [--version]
        {help,d.kinetics,d.probs,rw.probs,rw.kinetics,generic} ...
```

Plots error ranges for weighted ensemble datasets.

6.1.20.1 Command-line options

optional arguments:

```
-h, --help          show this help message and exit
```

general options:

```
-r RCFILE, --rcfile RCFILE
                        use RCFILE as the WEST run-time configuration file (default: west.
→cfg)
--quiet                emit only essential information
--verbose              emit extra information
--debug                enable extra checks and emit copious information
--version              show program's version number and exit
```

supported input formats:

```
{help,d.kinetics,d.probs,rw.probs,rw.kinetics,generic}
  help                print help for this command or individual subcommands
  d.kinetics           output of w_direct kinetics
  d.probs              output of w_direct probs
  rw.probs             output of w_reweight probs
  rw.kinetics          output of w_reweight kinetics
  generic              arbitrary HDF5 file and dataset
```

6.1.20.2 westpa.cli.tools.ploterr module

class westpa.cli.tools.ploterr.WESTMasterCommand

Bases: *WESTTool*

Base class for command-line tools that employ subcommands

subparsers_title = None

subcommands = None

include_help_command = True

add_args(*parser*)

Add arguments specific to this tool to the given argparse parser.

process_args(*args*)

Take argparse-processed arguments associated with this tool and deal with them appropriately (setting instance variables, etc)

go()

Perform the analysis associated with this tool.

class westpa.cli.tools.ploterr.**WESTSubcommand**(parent)

Bases: [WESTToolComponent](#)

Base class for command-line tool subcommands. A little sugar for making this more uniform.

subcommand = None

help_text = None

description = None

add_to_subparsers(subparsers)

go()

property work_manager

The work manager for this tool. Raises AttributeError if this is not a parallel tool.

class westpa.cli.tools.ploterr.**ProgressIndicatorComponent**

Bases: [WESTToolComponent](#)

add_args(parser)

Add arguments specific to this component to the given argparse parser.

process_args(args)

Take argparse-processed arguments associated with this component and deal with them appropriately (setting instance variables, etc)

class westpa.cli.tools.ploterr.**Plotter**(h5file, h5key, iteration=-1, interface='matplotlib')

Bases: object

This is a semi-generic plotting interface that has a built in curses based terminal plotter. It's fairly specific to what we're using it for here, but we could (and maybe should) build it out into a little library that we can use via the command line to plot things. Might be useful for looking at data later. That would also cut the size of this tool down by a good bit.

plot(i=0, j=1, tau=1, iteration=None, dim=0, interface=None)

class westpa.cli.tools.ploterr.**CommonPloterrs**(parent)

Bases: [WESTSubcommand](#)

add_args(parser)

Add arguments specific to this component to the given argparse parser.

process_args(args)

Take argparse-processed arguments associated with this component and deal with them appropriately (setting instance variables, etc)

parse_range(rangespec)

do_plot(data, output_filename, title=None, x_range=None, y_range=None, x_label=None, y_label=None)

class westpa.cli.tools.ploterr.**GenericIntervalSubcommand**(parent)

Bases: [CommonPloterrs](#)

```
description = 'Plots generic expectation/CI data. A path to the HDF5 file and the
dataset\nwithin it must be provided. This path takes the form
**FILENAME/PATH[SLICE]**.\nIf the dataset is not a vector (one dimensional) then a
slice must be provided.\nFor example, to access the state 0 to state 1 rate
evolution calculated by\n``w_kinavg``, one would use
``kinavg.h5/rate_evolution[:,0,1]``.\n\n\
n-----\
nCommand-line
arguments\
n-----\n'

subcommand = 'generic'

help_text = 'arbitrary HDF5 file and dataset'

add_args(parser)
    Add arguments specific to this component to the given argparse parser.

process_args(args)
    Take argparse-processed arguments associated with this component and deal with them appropriately (set-
    ting instance variables, etc)

load_and_validate_data()

go()

class westpa.cli.tools.ploterr.DirectKinetics(parent)
    Bases: CommonPloterrs

    subcommand = 'd.kinetics'

    help_text = 'output of w_direct kinetics'

    input_filename = 'direct.h5'

    flux_output_filename = 'flux_evolution_d_{state_label}.pdf'

    rate_output_filename = 'rate_evolution_d_{istate_label}_{fstate_label}.pdf'

    description = 'Plot evolution of state-to-state rates and total flux into states as
    generated\nby ``w_{direct/reweight} kinetics`` (when used with the
    ``--evolution-mode``\noption). Plots are generated for all rates/fluxes calculated.
    Output filenames\nrequire (and plot titles and axis labels support) substitution
    based on which\nflux/rate is being plotted:\n\n istate_label, fstate_label\n
    *(String, for rates)* Names of the initial and final states, as originally\n given
    to ``w_assign``.\n\n istate_index, fstate_index\n *(Integer, for rates)* Indices of
    initial and final states.\n\n state_label\n *(String, for fluxes)* Name of state\n\n
    state_index\n *(Integer, for fluxes)* Index of state\n'

    add_args(parser)
        Add arguments specific to this component to the given argparse parser.

    process_args(args)
        Take argparse-processed arguments associated with this component and deal with them appropriately (set-
        ting instance variables, etc)

    plot_flux(istate)
```

```

    plot_rate(istate, jstate)

    go()

class westpa.cli.tools.ploterr.DirectStateprobs(parent)
    Bases: CommonPloterrs

    subcommand = 'd.probs'

    help_text = 'output of w_direct probs'

    input_filename = 'direct.h5'

    pop_output_filename = 'pop_evolution_d_{state_label}.pdf'

    color_output_filename = 'color_evolution_d_{state_label}.pdf'

    description = 'Plot evolution of macrostate populations and associated
uncertainties. Plots\nare generated for all states calculated. Output filenames
require (and plot\ntitles and axis labels support) substitution based on which state
is being\nplotted:\n\n state_label\n *(String, for fluxes)* Name of state\n\n
state_index\n *(Integer, for fluxes)* Index of state\n'

    add_args(parser)
        Add arguments specific to this component to the given argparse parser.

    process_args(args)
        Take argparse-processed arguments associated with this component and deal with them appropriately (set-
ting instance variables, etc)

    plot_pop(istate)

    plot_color(istate)

    go()

class westpa.cli.tools.ploterr.ReweightStateprobs(parent)
    Bases: DirectStateprobs

    subcommand = 'rw.probs'

    help_text = 'output of w_reweight probs'

    input_filename = 'reweight.h5'

    pop_output_filename = 'pop_evolution_rw_{state_label}.pdf'

    color_output_filename = 'color_evolution_rw_{state_label}.pdf'

class westpa.cli.tools.ploterr.ReweightKinetics(parent)
    Bases: DirectKinetics

    subcommand = 'rw.kinetics'

    help_text = 'output of w_reweight kinetics'

    input_filename = 'reweight.h5'

    flux_output_filename = 'flux_evolution_rw_{state_label}.pdf'

```

```

    rate_output_filename = 'rate_evolution_rw_{istate_label}_{fstate_label}.pdf'

class westpa.cli.tools.ploterr.PloterrsTool
    Bases: WESTMasterCommand

    prog = 'ploterrs'

    subcommands = [<class 'westpa.cli.tools.ploterr.DirectKinetics'>, <class
'westpa.cli.tools.ploterr.DirectStateprobs'>, <class
'westpa.cli.tools.ploterr.ReweightStateprobs'>, <class
'westpa.cli.tools.ploterr.ReweightKinetics'>, <class
'westpa.cli.tools.ploterr.GenericIntervalSubcommand'>]

    subparsers_title = 'supported input formats'

    description = 'Plots error ranges for weighted ensemble datasets.\n\n\
n-----\
nCommand-line
options\
n-----\n'

westpa.cli.tools.ploterr.entry_point()

```

6.1.21 westpa.cli package

6.1.21.1 w_kinavg

WARNING: w_kinavg is being deprecated. Please use w_direct instead.

usage:

```

w_kinavg trace [-h] [-W WEST_H5FILE] [--first-iter N_ITER] [--last-iter N_ITER] [--step-
↪iter STEP]
                [-a ASSIGNMENTS] [-o OUTPUT] [-k KINETICS] [--disable-bootstrap] [--
↪disable-correl]
                [--alpha ALPHA] [--autocorrel-alpha ACALPHA] [--nsets NSETS]
                [-e {cumulative,blocked,none}] [--window-frac WINDOW_FRAC] [--
↪disable-averages]

```

Calculate average rates/fluxes and associated errors from weighted ensemble data. Bin assignments (usually “assign.h5”) and kinetics data (usually “direct.h5”) data files must have been previously generated (see “w_assign -help” and “w_direct init -help” for information on generating these files).

The evolution of all datasets may be calculated, with or without confidence intervals.

6.1.21.1.1 Output format

The output file (-o/-output, usually “direct.h5”) contains the following dataset:

```

/avg_rates [state,state]
(Structured -- see below) State-to-state rates based on entire window of
iterations selected.

/avg_total_fluxes [state]

```

(continues on next page)

(continued from previous page)

(Structured -- see below) Total fluxes into each state based on entire window of iterations selected.

/avg_conditional_fluxes [state,state]

(Structured -- see below) State-to-state fluxes based on entire window of iterations selected.

If --evolution-mode is specified, then the following additional datasets are available:

/rate_evolution [window][state][state]

(Structured -- see below). State-to-state rates based on windows of iterations of varying width. If --evolution-mode=cumulative, then these windows **all** begin at the iteration specified **with** --start-iter **and** grow **in** length by --step-iter **for** each successive element. If --evolution-mode=blocked, then these windows are **all** of width --step-iter (excluding the last, which may be shorter), the first of which begins at iteration --start-iter.

/target_flux_evolution [window,state]

(Structured -- see below). Total flux into a given macro state based on windows of iterations of varying width, **as in** /rate_evolution.

/conditional_flux_evolution [window,state,state]

(Structured -- see below). State-to-state fluxes based on windows of varying width, **as in** /rate_evolution.

The structure of these datasets is as follows:

iter_start

(Integer) Iteration at which the averaging window begins (inclusive).

iter_stop

(Integer) Iteration at which the averaging window ends (exclusive).

expected

(Floating-point) Expected (mean) value of the observable **as** evaluated within this window, **in** units of inverse tau.

ci_lbound

(Floating-point) Lower bound of the confidence interval of the observable within this window, **in** units of inverse tau.

ci_ubound

(Floating-point) Upper bound of the confidence interval of the observable within this window, **in** units of inverse tau.

stderr

(Floating-point) The standard error of the mean of the observable within this window, **in** units of inverse tau.

corr_len

(Integer) Correlation length of the observable within this window, **in** units of tau.

Each of these datasets is also stamped with a number of attributes:

```
mcbs_alpha
  (Floating-point) Alpha value of confidence intervals. (For example,
  *alpha=0.05* corresponds to a 95% confidence interval.)

mcbs_nsets
  (Integer) Number of bootstrap data sets used in generating confidence
  intervals.

mcbs_acalpha
  (Floating-point) Alpha value for determining correlation lengths.
```

6.1.21.1.2 Command-line options

optional arguments:

```
-h, --help          show this help message and exit
```

WEST input data options:

```
-W WEST_H5FILE, --west-data WEST_H5FILE
                        Take WEST data from WEST_H5FILE (default: read from the HDF5 file
↳ specified in         west.cfg).
```

iteration range:

```
--first-iter N_ITER   Begin analysis at iteration N_ITER (default: 1).
--last-iter N_ITER    Conclude analysis with N_ITER, inclusive (default: last completed
↳ iteration).
--step-iter STEP       Analyze/report in blocks of STEP iterations.
```

input/output options:

```
-a ASSIGNMENTS, --assignments ASSIGNMENTS
                        Bin assignments and macrostate definitions are in ASSIGNMENTS
↳ (default:            assign.h5).
-o OUTPUT, --output OUTPUT
                        Store results in OUTPUT (default: kinavg.h5).
```

input/output options:

```
-k KINETICS, --kinetics KINETICS
                        Populations and transition rates are stored in KINETICS (default:
↳ kintrace.h5).
```

confidence interval calculation options:

```
--disable-bootstrap, -db
                        Enable the use of Monte Carlo Block Bootstrapping.
--disable-correl, -dc
```

(continues on next page)

(continued from previous page)

```

--alpha ALPHA          Disable the correlation analysis.
--autocorrel-alpha ACALPHA Calculate a (1-ALPHA) confidence interval' (default: 0.05)
                        Evaluate autocorrelation to (1-ACALPHA) significance. Note that
↳ too small an
                        ACALPHA will result in failure to detect autocorrelation in a
↳ noisy flux signal.
                        (Default: same as ALPHA.)
--nsets NSETS          Use NSETS samples for bootstrapping (default: chosen based on
↳ ALPHA)

```

calculation options:

```

-e {cumulative,blocked,none}, --evolution-mode {cumulative,blocked,none}
                        How to calculate time evolution of rate estimates. ``cumulative``
↳ evaluates rates
                        over windows starting with --start-iter and getting progressively
↳ wider to --stop-
                        iter by steps of --step-iter. ``blocked`` evaluates rates over
↳ windows of width
                        --step-iter, the first of which begins at --start-iter. ``none``
↳ (the default)
                        disables calculation of the time evolution of rate estimates.
--window-frac WINDOW_FRAC Fraction of iterations to use in each window when running in
↳ ``cumulative`` mode.
                        The (1 - frac) fraction of iterations will be discarded from the
↳ start of each
                        window.

```

misc options:

```

--disable-averages, -da Whether or not the averages should be printed to the console (set
↳ to FALSE if flag
                        is used).

```

6.1.21.1.3 westpa.cli.tools.w_kinavg module

```
class westpa.cli.tools.w_kinavg.WESTMasterCommand
```

Bases: [WESTTool](#)

Base class for command-line tools that employ subcommands

subparsers_title = None

subcommands = None

include_help_command = True

add_args(parser)

Add arguments specific to this tool to the given argparse parser.

process_args(args)

Take argparse-processed arguments associated with this tool and deal with them appropriately (setting instance variables, etc)

go()

Perform the analysis associated with this tool.

class westpa.cli.tools.w_kinavg.**WESTParallelTool**(wm_env=None)

Bases: [*WESTTool*](#)

Base class for command-line tools parallelized with wwmgr. This automatically adds and processes wwmgr command-line arguments and creates a work manager at self.work_manager.

make_parser_and_process(prog=None, usage=None, description=None, epilog=None, args=None)

A convenience function to create a parser, call add_all_args(), and then call process_all_args(). The argument namespace is returned.

add_args(parser)

Add arguments specific to this tool to the given argparse parser.

process_args(args)

Take argparse-processed arguments associated with this tool and deal with them appropriately (setting instance variables, etc)

go()

Perform the analysis associated with this tool.

main()

A convenience function to make a parser, parse and process arguments, then run self.go() in the master process.

class westpa.cli.tools.w_kinavg.**DKinAvg**(parent)

Bases: [*AverageCommands*](#)

subcommand = 'kinetics'

help_text = 'Generates rate and flux values from a WESTPA simulation via tracing.'

default_kinetics_file = 'direct.h5'


```

description = 'Calculate average rates/fluxes and associated errors from weighted
ensemble\ndata. Bin assignments (usually "assign.h5") and kinetics data
(usually\n"direct.h5") data files must have been previously generated
(see\n"w_assign --help" and "w_direct init --help" for information on\ngenerating
these files).\n\nThe evolution of all datasets may be calculated, with or without
confidence\nintervals.\n\
n-----\
nOutput
format\
n-----\n\
nThe output file (-o/--output, usually "direct.h5") contains the
following\ndataset:\n\n /avg_rates [state,state]\n (Structured -- see below)
State-to-state rates based on entire window of\n iterations selected.\n\n
/avg_total_fluxes [state]\n (Structured -- see below) Total fluxes into each state
based on entire\n window of iterations selected.\n\n /avg_conditional_fluxes
[state,state]\n (Structured -- see below) State-to-state fluxes based on entire
window of\n iterations selected.\n\nIf --evolution-mode is specified, then the
following additional datasets are\navailable:\n\n /rate_evolution
[window][state][state]\n (Structured -- see below). State-to-state rates based on
windows of\n iterations of varying width. If --evolution-mode=cumulative, then\n
these windows all begin at the iteration specified with\n --start-iter and grow in
length by --step-iter for each successive\n element. If --evolution-mode=blocked,
then these windows are all of\n width --step-iter (excluding the last, which may be
shorter), the first\n of which begins at iteration --start-iter.\n\n
/target_flux_evolution [window,state]\n (Structured -- see below). Total flux into a
given macro state based on\n windows of iterations of varying width, as in
/rate_evolution.\n\n /conditional_flux_evolution [window,state,state]\n (Structured
-- see below). State-to-state fluxes based on windows of\n varying width, as in
/rate_evolution.\n\nThe structure of these datasets is as follows:\n\n iter_start\n
(Integer) Iteration at which the averaging window begins (inclusive).\n\n
iter_stop\n (Integer) Iteration at which the averaging window ends (exclusive).\n\n
expected\n (Floating-point) Expected (mean) value of the observable as evaluated
within\n this window, in units of inverse tau.\n\n ci_lbound\n (Floating-point)
Lower bound of the confidence interval of the observable\n within this window, in
units of inverse tau.\n\n ci_ubound\n (Floating-point) Upper bound of the confidence
interval of the observable\n within this window, in units of inverse tau.\n\n
stderr\n (Floating-point) The standard error of the mean of the observable\n within
this window, in units of inverse tau.\n\n corr_len\n (Integer) Correlation length of
the observable within this window, in units\n of tau.\n\nEach of these datasets is
also stamped with a number of attributes:\n\n mcbs_alpha\n (Floating-point) Alpha
value of confidence intervals. (For example,\n *alpha=0.05* corresponds to a 95%
confidence interval.)\n\n mcbs_nsets\n (Integer) Number of bootstrap data sets used
in generating confidence\n intervals.\n\n mcbs_acalpha\n (Floating-point) Alpha
value for determining correlation lengths.\n\n\
n-----\
nCommand-line
options\
n-----\n'

w_kinavg()

go()

```

```
westpa.cli.tools.w_kinavg.warn()
```

Issue a warning, or maybe ignore it or raise an exception.

message

Text of the warning message.

category

The Warning category subclass. Defaults to UserWarning.

stacklevel

How far up the call stack to make this warning appear. A value of 2 for example attributes the warning to the caller of the code calling warn().

source

If supplied, the destroyed object which emitted a ResourceWarning

skip_file_prefixes

An optional tuple of module filename prefixes indicating frames to skip during stacklevel computations for stack frame attribution.

```
class westpa.cli.tools.w_kinavg.WKinAvg(parent)
```

Bases: *DKinAvg*

```
subcommand = 'trace'
```

```
help_text = 'averages and CIs for path-tracing kinetics analysis'
```

```
default_kinetics_file = 'kintrace.h5'
```

```
default_output_file = 'kinavg.h5'
```

```
class westpa.cli.tools.w_kinavg.WDirect
```

Bases: *WESTMasterCommand*, *WESTParallelTool*

```
prog = 'w_kinavg'
```

```
subcommands = [<class 'westpa.cli.tools.w_kinavg.WKinAvg'>]
```

```
subparsers_title = 'direct kinetics analysis schemes'
```

```

description = 'Calculate average rates and associated errors from weighted ensemble
data. Bin\nassignments (usually "assignments.h5") and kinetics data
(usually\n"kintrace.h5" or "kinmat.h5") data files must have been previously
generated\n(see "w_assign --help" and "w_kinetics --help" for information on
generating\nthese files).\n\
n-----\
nOutput
format\
n-----\n\
nThe output file (-o/--output, usually "kinavg.h5") contains the
following\ndataset:\n\n /avg_rates [state,state]\n (Structured -- see below)
State-to-state rates based on entire window of\n iterations selected.\n\nFor trace
mode, the following additional datasets are generated:\n\n /avg_total_fluxes
[state]\n (Structured -- see below) Total fluxes into each state based on entire\n
window of iterations selected.\n\n /avg_conditional_fluxes [state,state]\n
(Structured -- see below) State-to-state fluxes based on entire window of\n
iterations selected.\n\nIf --evolution-mode is specified, then the following
additional dataset is\navailable:\n\n /rate_evolution [window][state][state]\n
(Structured -- see below). State-to-state rates based on windows of\n iterations of
varying width. If --evolution-mode=cumulative, then\n these windows all begin at the
iteration specified with\n --start-iter and grow in length by --step-iter for each
successive\n element. If --evolution-mode=blocked, then these windows are all of\n
width --step-iter (excluding the last, which may be shorter), the first\n of which
begins at iteration --start-iter.\n\nIf --evolution-mode is specified in trace mode,
the following additional\ndatasets are available:\n\n /target_flux_evolution
[window,state]\n (Structured -- see below). Total flux into a given macro state
based on\n windows of iterations of varying width, as in /rate_evolution.\n\n
/conditional_flux_evolution [window,state,state]\n (Structured -- see below).
State-to-state fluxes based on windows of\n varying width, as in
/rate_evolution.\n\nThe structure of these datasets is as follows:\n\n iter_start\n
(Integer) Iteration at which the averaging window begins (inclusive).\n\n
iter_stop\n (Integer) Iteration at which the averaging window ends (exclusive).\n\n
expected\n (Floating-point) Expected (mean) value of the rate as evaluated within\n
this window, in units of inverse tau.\n\n ci_lbound\n (Floating-point) Lower bound
of the confidence interval on the rate\n within this window, in units of inverse
tau.\n\n ci_ubound\n (Floating-point) Upper bound of the confidence interval on the
rate\n within this window, in units of inverse tau.\n\n corr_len\n (Integer)
Correlation length of the rate within this window, in units\n of tau.\n\nEach of
these datasets is also stamped with a number of attributes:\n\n mcbs_alpha\n
(Floating-point) Alpha value of confidence intervals. (For example,\n *alpha=0.05*
corresponds to a 95% confidence interval.)\n\n mcbs_nsets\n (Integer) Number of
bootstrap data sets used in generating confidence\n intervals.\n\n mcbs_acalpha\n
(Floating-point) Alpha value for determining correlation lengths.\n\n\
n-----\
nCommand-line
options\
n-----\n'

```

```
westpa.cli.tools.w_kinavg.entry_point()
```

6.1.21.2 w_kinetics

WARNING: w_kinetics is being deprecated. Please use w_direct instead.

usage:

```
w_kinetics trace [-h] [-W WEST_H5FILE] [--first-iter N_ITER] [--last-iter N_ITER]
               [--step-iter STEP] [-a ASSIGNMENTS] [-o OUTPUT]
```

Calculate state-to-state rates and transition event durations by tracing trajectories.

A bin assignment file (usually “assign.h5”) including trajectory labeling is required (see “w_assign -help” for information on generating this file).

This subcommand for w_direct is used as input for all other w_direct subcommands, which will convert the flux data in the output file into average rates/fluxes/populations with confidence intervals.

6.1.21.2.1 Output format

The output file (-o/-output, by default “direct.h5”) contains the following datasets:

```
``/conditional_fluxes`` [iteration][state][state]
  *(Floating-point)* Macrostate-to-macrostate fluxes. These are **not**
  normalized by the population of the initial macrostate.

``/conditional_arrivals`` [iteration][stateA][stateB]
  *(Integer)* Number of trajectories arriving at state *stateB* in a given
  iteration, given that they departed from *stateA*.

``/total_fluxes`` [iteration][state]
  *(Floating-point)* Total flux into a given macrostate.

``/arrivals`` [iteration][state]
  *(Integer)* Number of trajectories arriving at a given state in a given
  iteration, regardless of where they originated.

``/duration_count`` [iteration]
  *(Integer)* The number of event durations recorded in each iteration.

``/durations`` [iteration][event duration]
  *(Structured -- see below)* Event durations for transition events ending
  during a given iteration. These are stored as follows:

    istate
      *(Integer)* Initial state of transition event.
    fstate
      *(Integer)* Final state of transition event.
    duration
      *(Floating-point)* Duration of transition, in units of tau.
    weight
      *(Floating-point)* Weight of trajectory at end of transition, **not**
      normalized by initial state population.
```

Because state-to-state fluxes stored in this file are not normalized by initial macrostate population, they cannot be used as rates without further processing. The w_direct kinetics command is used to perform this normalization while

taking statistical fluctuation and correlation into account. See `w_direct kinetics --help` for more information. Target fluxes (total flux into a given state) require no such normalization.

6.1.21.2.2 Command-line options

optional arguments:

```
-h, --help          show this help message and exit
```

WEST input data options:

```
-W WEST_H5FILE, --west-data WEST_H5FILE
                        Take WEST data from WEST_H5FILE (default: read from the HDF5 file
↳ specified in
                        west.cfg).
```

iteration range:

```
--first-iter N_ITER   Begin analysis at iteration N_ITER (default: 1).
--last-iter N_ITER    Conclude analysis with N_ITER, inclusive (default: last completed
↳ iteration).
--step-iter STEP       Analyze/report in blocks of STEP iterations.
```

input/output options:

```
-a ASSIGNMENTS, --assignments ASSIGNMENTS
                        Bin assignments and macrostate definitions are in ASSIGNMENTS
↳ (default:
                        assign.h5).
-o OUTPUT, --output OUTPUT
                        Store results in OUTPUT (default: kintrace.h5).
```

6.1.21.2.3 westpa.cli.tools.w_kinetics module

class `westpa.cli.tools.w_kinetics.WESTMasterCommand`

Bases: `WESTTool`

Base class for command-line tools that employ subcommands

subparsers_title = `None`

subcommands = `None`

include_help_command = `True`

add_args(*parser*)

Add arguments specific to this tool to the given argparse parser.

process_args(*args*)

Take argparse-processed arguments associated with this tool and deal with them appropriately (setting instance variables, etc)

go()

Perform the analysis associated with this tool.

class westpa.cli.tools.w_kinetics.WESTParallelTool(*wm_env=None*)

Bases: [WESTTool](#)

Base class for command-line tools parallelized with wwmgr. This automatically adds and processes wwmgr command-line arguments and creates a work manager at self.work_manager.

make_parser_and_process(*prog=None, usage=None, description=None, epilog=None, args=None*)

A convenience function to create a parser, call add_all_args(), and then call process_all_args(). The argument namespace is returned.

add_args(*parser*)

Add arguments specific to this tool to the given argparse parser.

process_args(*args*)

Take argparse-processed arguments associated with this tool and deal with them appropriately (setting instance variables, etc)

go()

Perform the analysis associated with this tool.

main()

A convenience function to make a parser, parse and process arguments, then run self.go() in the master process.

westpa.cli.tools.w_kinetics.**warn**()

Issue a warning, or maybe ignore it or raise an exception.

message

Text of the warning message.

category

The Warning category subclass. Defaults to UserWarning.

stacklevel

How far up the call stack to make this warning appear. A value of 2 for example attributes the warning to the caller of the code calling warn().

source

If supplied, the destroyed object which emitted a ResourceWarning

skip_file_prefixes

An optional tuple of module filename prefixes indicating frames to skip during stacklevel computations for stack frame attribution.

class westpa.cli.tools.w_kinetics.DKinetics(*parent*)

Bases: [WESTKineticsBase](#), [WKinetics](#)

subcommand = 'init'

default_kinetics_file = 'direct.h5'

default_output_file = 'direct.h5'

help_text = 'calculate state-to-state kinetics by tracing trajectories'

```

description = 'Calculate state-to-state rates and transition event durations by
tracing\ntrajectories.\n\nA bin assignment file (usually "assign.h5") including
trajectory labeling\nis required (see "w_assign --help" for information on
generating this file).\n\nThis subcommand for w_direct is used as input for all
other w_direct\nsubcommands, which will convert the flux data in the output file
into\naverage rates/fluxes/populations with confidence intervals.\n\
n-----\
nOutput
format\
n-----\n\
nThe output file (-o/--output, by default "direct.h5") contains the\nfollowing
datasets:\n\n ``/conditional_fluxes`` [iteration][state][state]\n *(Floating-point)*
Macrostate-to-macrostate fluxes. These are **not**\n normalized by the population of
the initial macrostate.\n\n ``/conditional_arrivals`` [iteration][stateA][stateB]\n
*(Integer)* Number of trajectories arriving at state *stateB* in a given\n
iteration, given that they departed from *stateA*.\n\n ``/total_fluxes``
[iteration][state]\n *(Floating-point)* Total flux into a given macrostate.\n\n
``/arrivals`` [iteration][state]\n *(Integer)* Number of trajectories arriving at a
given state in a given\n iteration, regardless of where they originated.\n\n
``/duration_count`` [iteration]\n *(Integer)* The number of event durations recorded
in each iteration.\n\n ``/durations`` [iteration][event duration]\n *(Structured --
see below)* Event durations for transition events ending\n during a given iteration.
These are stored as follows:\n\n istate\n *(Integer)* Initial state of transition
event.\n fstate\n *(Integer)* Final state of transition event.\n duration\n
*(Floating-point)* Duration of transition, in units of tau.\n weight\n
*(Floating-point)* Weight of trajectory at end of transition, **not**\n normalized
by initial state population.\n\nBecause state-to-state fluxes stored in this file
are not normalized by\ninitial macrostate population, they cannot be used as rates
without further\nprocessing. The ``w_direct kinetics`` command is used to perform
this normalization\nwhile taking statistical fluctuation and correlation into
account. See\n``w_direct kinetics --help`` for more information. Target fluxes
(total flux\ninto a given state) require no such normalization.\n\
n-----\
nCommand-line
options\
n-----\n'

open_files()

go()

```

```
class westpa.cli.tools.w_kinetics.WKinetics(parent)
```

```
    Bases: DKinetics
```

```
    subcommand = 'trace'
```

```
    help_text = 'averages and CIs for path-tracing kinetics analysis'
```

```
    default_output_file = 'kintrace.h5'
```

```
class westpa.cli.tools.w_kinetics.WDirect
```

```
    Bases: WESTMasterCommand, WESTParallelTool
```

```
    prog = 'w_kinetics'
```

```

subcommands = [<class 'westpa.cli.tools.w_kinetics.WKinetics'>]

subparsers_title = 'calculate state-to-state kinetics by tracing trajectories'

description = 'Calculate state-to-state rates and transition event durations by
tracing\ntrajectories.\n\nA bin assignment file (usually "assign.h5") including
trajectory labeling\nis required (see "w_assign --help" for information on
generating this file).\n\nThe output generated by this program is used as input for
the ``w_kinavg``\ntool, which converts the flux data in the output file into average
rates\nwith confidence intervals. See ``w_kinavg trace --help`` for
more\ninformation.\n\
n-----\
nOutput
format\
n-----\
n
The output file (-o/--output, by default "kintrace.h5") contains the\nfollowing
datasets:\n\n ``/conditional_fluxes`` [iteration][state][state]\n *(Floating-point)*
Macrostate-to-macrostate fluxes. These are **not**\n normalized by the population of
the initial macrostate.\n\n ``/conditional_arrivals`` [iteration][stateA][stateB]\n
*(Integer)* Number of trajectories arriving at state *stateB* in a given\n
iteration, given that they departed from *stateA*.\n\n ``/total_fluxes``
[iteration][state]\n *(Floating-point)* Total flux into a given macrostate.\n\n
``/arrivals`` [iteration][state]\n *(Integer)* Number of trajectories arriving at a
given state in a given\n iteration, regardless of where they originated.\n\n
``/duration_count`` [iteration]\n *(Integer)* The number of event durations recorded
in each iteration.\n\n ``/durations`` [iteration][event duration]\n *(Structured --
see below)* Event durations for transition events ending\nduring a given iteration.
These are stored as follows:\n\n istate\n *(Integer)* Initial state of transition
event.\n fstate\n *(Integer)* Final state of transition event.\n duration\n
*(Floating-point)* Duration of transition, in units of tau.\n weight\n
*(Floating-point)* Weight of trajectory at end of transition, **not**\n normalized
by initial state population.\n\nBecause state-to-state fluxes stored in this file
are not normalized by\ninitial macrostate population, they cannot be used as rates
without further\nprocessing. The ``w_kinavg`` command is used to perform this
normalization\nwhile taking statistical fluctuation and correlation into account.
See\n``w_kinavg trace --help`` for more information. Target fluxes (total flux\ninto
a given state) require no such normalization.\n\
n-----\
nCommand-line
options\
n-----\n'

westpa.cli.tools.w_kinetics.entry_point()

```


6.1.21.3 w_stateprobs

WARNING: w_stateprobs is being deprecated. Please use w_direct instead.

usage:

```
w_stateprobs trace [-h] [-W WEST_H5FILE] [--first-iter N_ITER] [--last-iter N_ITER]
                  [--step-iter STEP] [-a ASSIGNMENTS] [-o OUTPUT] [-k KINETICS]
                  [--disable-bootstrap] [--disable-correl] [--alpha ALPHA]
                  [--autocorrel-alpha ACALPHA] [--nsets NSETS] [-e {cumulative,
                  ↪blocked,none}]
                  [--window-frac WINDOW_FRAC] [--disable-averages]
```

Calculate average populations and associated errors in state populations from weighted ensemble data. Bin assignments, including macrostate definitions, are required. (See “w_assign -help” for more information).

6.1.21.3.1 Output format

The output file (-o/-output, usually “direct.h5”) contains the following dataset:

```
/avg_state_probs [state]
  (Structured -- see below) Population of each state across entire
  range specified.

/avg_color_probs [state]
  (Structured -- see below) Population of each ensemble across entire
  range specified.
```

If -evolution-mode is specified, then the following additional datasets are available:

```
/state_pop_evolution [window][state]
  (Structured -- see below). State populations based on windows of
  iterations of varying width. If --evolution-mode=cumulative, then
  these windows all begin at the iteration specified with
  --start-iter and grow in length by --step-iter for each successive
  element. If --evolution-mode=blocked, then these windows are all of
  width --step-iter (excluding the last, which may be shorter), the first
  of which begins at iteration --start-iter.

/color_prob_evolution [window][state]
  (Structured -- see below). Ensemble populations based on windows of
  iterations of varying width. If --evolution-mode=cumulative, then
  these windows all begin at the iteration specified with
  --start-iter and grow in length by --step-iter for each successive
  element. If --evolution-mode=blocked, then these windows are all of
  width --step-iter (excluding the last, which may be shorter), the first
  of which begins at iteration --start-iter.
```

The structure of these datasets is as follows:

```
iter_start
  (Integer) Iteration at which the averaging window begins (inclusive).
```

(continues on next page)

(continued from previous page)

`iter_stop`
(Integer) Iteration at which the averaging window ends (exclusive).

`expected`
(Floating-point) Expected (mean) value of the observable `as` evaluated within this window, `in` units of inverse tau.

`ci_lbound`
(Floating-point) Lower bound of the confidence interval of the observable within this window, `in` units of inverse tau.

`ci_ubound`
(Floating-point) Upper bound of the confidence interval of the observable within this window, `in` units of inverse tau.

`stderr`
(Floating-point) The standard error of the mean of the observable within this window, `in` units of inverse tau.

`corr_len`
(Integer) Correlation length of the observable within this window, `in` units of tau.

Each of these datasets is also stamped with a number of attributes:

`mcbs_alpha`
(Floating-point) Alpha value of confidence intervals. (For example, `*alpha=0.05*` corresponds to a 95% confidence interval.)

`mcbs_nsets`
(Integer) Number of bootstrap data sets used `in` generating confidence intervals.

`mcbs_acalpha`
(Floating-point) Alpha value `for` determining correlation lengths.

6.1.21.3.2 Command-line options

optional arguments:

`-h, --help` show this help message `and` exit

WEST input data options:

`-W WEST_H5FILE, --west-data WEST_H5FILE`
Take WEST data `from` `WEST_H5FILE` (default: read `from` `the` HDF5 file, `↪` specified `in` `west.cfg`).

iteration range:

```
--first-iter N_ITER    Begin analysis at iteration N_ITER (default: 1).
--last-iter N_ITER     Conclude analysis with N_ITER, inclusive (default: last completed
↳ iteration).
--step-iter STEP       Analyze/report in blocks of STEP iterations.
```

input/output options:

```
-a ASSIGNMENTS, --assignments ASSIGNMENTS
    Bin assignments and macrostate definitions are in ASSIGNMENTS
↳ (default:
    assign.h5).
-o OUTPUT, --output OUTPUT
    Store results in OUTPUT (default: stateprobs.h5).
```

input/output options:

```
-k KINETICS, --kinetics KINETICS
    Populations and transition rates are stored in KINETICS (default:
↳ assign.h5).
```

confidence interval calculation options:

```
--disable-bootstrap, -db
    Enable the use of Monte Carlo Block Bootstrapping.
--disable-correl, -dc
    Disable the correlation analysis.
--alpha ALPHA          Calculate a (1-ALPHA) confidence interval' (default: 0.05)
--autocorrel-alpha ACALPHA
    Evaluate autocorrelation to (1-ACALPHA) significance. Note that
↳ too small an
    ACALPHA will result in failure to detect autocorrelation in a
↳ noisy flux signal.
    (Default: same as ALPHA.)
--nsets NSETS          Use NSETS samples for bootstrapping (default: chosen based on
↳ ALPHA)
```

calculation options:

```
-e {cumulative,blocked,none}, --evolution-mode {cumulative,blocked,none}
    How to calculate time evolution of rate estimates. ``cumulative``
↳ evaluates rates
    over windows starting with --start-iter and getting progressively
↳ wider to --stop-
    iter by steps of --step-iter. ``blocked`` evaluates rates over
↳ windows of width
    --step-iter, the first of which begins at --start-iter. ``none``
↳ (the default)
    disables calculation of the time evolution of rate estimates.
--window-frac WINDOW_FRAC
    Fraction of iterations to use in each window when running in
↳ ``cumulative`` mode.
    The (1 - frac) fraction of iterations will be discarded from the
↳ start of each
    window.
```

misc options:

```
--disable-averages, -da
                        Whether or not the averages should be printed to the console (set_
↳to FALSE if flag      is used).
```

6.1.21.3.3 westpa.cli.tools.w_stateprobs module

class westpa.cli.tools.w_stateprobs.WESTMasterCommand

Bases: [WESTTool](#)

Base class for command-line tools that employ subcommands

subparsers_title = None

subcommands = None

include_help_command = True

add_args(parser)

Add arguments specific to this tool to the given argparse parser.

process_args(args)

Take argparse-processed arguments associated with this tool and deal with them appropriately (setting instance variables, etc)

go()

Perform the analysis associated with this tool.

class westpa.cli.tools.w_stateprobs.WESTParallelTool(wm_env=None)

Bases: [WESTTool](#)

Base class for command-line tools parallelized with wwmgr. This automatically adds and processes wwmgr command-line arguments and creates a work manager at self.work_manager.

make_parser_and_process(prog=None, usage=None, description=None, epilog=None, args=None)

A convenience function to create a parser, call add_all_args(), and then call process_all_args(). The argument namespace is returned.

add_args(parser)

Add arguments specific to this tool to the given argparse parser.

process_args(args)

Take argparse-processed arguments associated with this tool and deal with them appropriately (setting instance variables, etc)

go()

Perform the analysis associated with this tool.

main()

A convenience function to make a parser, parse and process arguments, then run self.go() in the master process.

`westpa.cli.tools.w_stateprobs.warn()`

Issue a warning, or maybe ignore it or raise an exception.

message

Text of the warning message.

category

The Warning category subclass. Defaults to UserWarning.

stacklevel

How far up the call stack to make this warning appear. A value of 2 for example attributes the warning to the caller of the code calling warn().

source

If supplied, the destroyed object which emitted a ResourceWarning

skip_file_prefixes

An optional tuple of module filename prefixes indicating frames to skip during stacklevel computations for stack frame attribution.

class `westpa.cli.tools.w_stateprobs.DStateProbs(parent)`

Bases: [*AverageCommands*](#)

subcommand = 'probs'

help_text = 'Calculates color and state probabilities via tracing.'

default_kinetics_file = 'direct.h5'

```

description = 'Calculate average populations and associated errors in state
populations from\nweighted ensemble data. Bin assignments, including macrostate
definitions,\nare required. (See "w_assign --help" for more information).\n\
n-----\
nOutput
format\
n-----\n\
nThe output file (-o/--output, usually "direct.h5") contains the
following\ndataset:\n\n /avg_state_probs [state]\n (Structured -- see below)
Population of each state across entire\n range specified.\n\n /avg_color_probs
[state]\n (Structured -- see below) Population of each ensemble across entire\n
range specified.\n\nIf --evolution-mode is specified, then the following additional
datasets are\navailable:\n\n /state_pop_evolution [window][state]\n (Structured --
see below). State populations based on windows of\n iterations of varying width. If
--evolution-mode=cumulative, then\n these windows all begin at the iteration
specified with\n --start-iter and grow in length by --step-iter for each
successive\n element. If --evolution-mode=blocked, then these windows are all of\n
width --step-iter (excluding the last, which may be shorter), the first\n of which
begins at iteration --start-iter.\n\n /color_prob_evolution [window][state]\n
(Structured -- see below). Ensemble populations based on windows of\n iterations of
varying width. If --evolution-mode=cumulative, then\n these windows all begin at the
iteration specified with\n --start-iter and grow in length by --step-iter for each
successive\n element. If --evolution-mode=blocked, then these windows are all of\n
width --step-iter (excluding the last, which may be shorter), the first\n of which
begins at iteration --start-iter.\n\nThe structure of these datasets is as
follows:\n\n iter_start\n (Integer) Iteration at which the averaging window begins
(inclusive).\n\n iter_stop\n (Integer) Iteration at which the averaging window ends
(exclusive).\n\n expected\n (Floating-point) Expected (mean) value of the observable
as evaluated within\n this window, in units of inverse tau.\n\n ci_lbound\n
(Floating-point) Lower bound of the confidence interval of the observable\n within
this window, in units of inverse tau.\n\n ci_ubound\n (Floating-point) Upper bound
of the confidence interval of the observable\n within this window, in units of
inverse tau.\n\n stderr\n (Floating-point) The standard error of the mean of the
observable\n within this window, in units of inverse tau.\n\n corr_len\n (Integer)
Correlation length of the observable within this window, in units\n of tau.\n\nEach
of these datasets is also stamped with a number of attributes:\n\n mcbs_alpha\n
(Floating-point) Alpha value of confidence intervals. (For example,\n *alpha=0.05*
corresponds to a 95% confidence interval.)\n\n mcbs_nsets\n (Integer) Number of
bootstrap data sets used in generating confidence\n intervals.\n\n mcbs_acalpha\n
(Floating-point) Alpha value for determining correlation lengths.\n\n\
n-----\
nCommand-line
options\
n-----\n'

```

```
calculate_state_populations(pops)
```

```
w_stateprobs()
```

```
go()
```

```
class westpa.cli.tools.w_stateprobs.WStateProbs(parent)
```

```
    Bases: DStateProbs
```

```
    subcommand = 'trace'
```

```

help_text = 'averages and CIs for path-tracing kinetics analysis'

default_output_file = 'stateprobs.h5'

default_kinetics_file = 'assign.h5'

class westpa.cli.tools.w_stateprobs.WDirect
    Bases: WESTMasterCommand, WESTParallelTool

    prog = 'w_stateprobs'

    subcommands = [<class 'westpa.cli.tools.w_stateprobs.WStateProbs'>]

    subparsers_title = 'calculate state-to-state kinetics by tracing trajectories'

    description = 'Calculate average populations and associated errors in state
populations from\nweighted ensemble data. Bin assignments, including macrostate
definitions,\nare required. (See "w_assign --help" for more information).\n\
n-----\
nOutput
format\
n-----\n\
nThe output file (-o/--output, usually "stateprobs.h5") contains the
following\ndataset:\n\n /avg_state_pops [state]\n (Structured -- see below)
Population of each state across entire\n range specified.\n\nIf --evolution-mode is
specified, then the following additional dataset is\navailable:\n\n
/state_pop_evolution [window][state]\n (Structured -- see below). State populations
based on windows of\n iterations of varying width. If --evolution-mode=cumulative,
then\n these windows all begin at the iteration specified with\n --start-iter and
grow in length by --step-iter for each successive\n element. If
--evolution-mode=blocked, then these windows are all of\n width --step-iter
(excluding the last, which may be shorter), the first\n of which begins at iteration
--start-iter.\n\nThe structure of these datasets is as follows:\n\n iter_start\n
(Integer) Iteration at which the averaging window begins (inclusive).\n\n
iter_stop\n (Integer) Iteration at which the averaging window ends (exclusive).\n\n
expected\n (Floating-point) Expected (mean) value of the rate as evaluated within\n
this window, in units of inverse tau.\n\n ci_lbound\n (Floating-point) Lower bound
of the confidence interval on the rate\n within this window, in units of inverse
tau.\n\n ci_ubound\n (Floating-point) Upper bound of the confidence interval on the
rate\n within this window, in units of inverse tau.\n\n corr_len\n (Integer)
Correlation length of the rate within this window, in units\n of tau.\n\nEach of
these datasets is also stamped with a number of attributes:\n\n mcbs_alpha\n
(Floating-point) Alpha value of confidence intervals. (For example,\n *alpha=0.05*
corresponds to a 95% confidence interval.)\n\n mcbs_nsets\n (Integer) Number of
bootstrap data sets used in generating confidence\n intervals.\n\n mcbs_acalpha\n
(Floating-point) Alpha value for determining correlation lengths.\n\n\
n-----\
nCommand-line
options\
n-----\n'

westpa.cli.tools.w_stateprobs.entry_point()

```

6.1.21.4 w_dumpsegs

6.1.21.4.1 westpa.cli.tools.w_dumpsegs module

`westpa.cli.tools.w_dumpsegs.warn()`

Issue a warning, or maybe ignore it or raise an exception.

message

Text of the warning message.

category

The Warning category subclass. Defaults to UserWarning.

stacklevel

How far up the call stack to make this warning appear. A value of 2 for example attributes the warning to the caller of the code calling `warn()`.

source

If supplied, the destroyed object which emitted a ResourceWarning

skip_file_prefixes

An optional tuple of module filename prefixes indicating frames to skip during stacklevel computations for stack frame attribution.

class `westpa.cli.tools.w_dumpsegs.WESTTool`

Bases: [*WESTToolComponent*](#)

Base class for WEST command line tools

prog = None

usage = None

description = None

epilog = None

add_args(*parser*)

Add arguments specific to this tool to the given argparse parser.

process_args(*args*)

Take argparse-processed arguments associated with this tool and deal with them appropriately (setting instance variables, etc)

make_parser(*prog=None, usage=None, description=None, epilog=None, args=None*)

make_parser_and_process(*prog=None, usage=None, description=None, epilog=None, args=None*)

A convenience function to create a parser, call `add_all_args()`, and then call `process_all_args()`. The argument namespace is returned.

go()

Perform the analysis associated with this tool.

main()

A convenience function to make a parser, parse and process arguments, then call `self.go()`

class `westpa.cli.tools.w_dumpsegs.WESTDataReader`

Bases: [*WESTToolComponent*](#)

Tool for reading data from WEST-related HDF5 files. Coordinates finding the main HDF5 file from `west.cfg` or command line arguments, caching of certain kinds of data (eventually), and retrieving auxiliary data sets from various places.

add_args(*parser*)

Add arguments specific to this component to the given argparse parser.

process_args(*args*)

Take argparse-processed arguments associated with this component and deal with them appropriately (setting instance variables, etc)

open(*mode='r'*)

close()

property *weight_dsspec*

property *parent_id_dsspec*

```
class westpa.cli.tools.w_dumpsegs.Segment(n_iter=None, seg_id=None, weight=None,
                                         endpoint_type=None, parent_id=None,
                                         wtg_parent_ids=None, pcoord=None, status=None,
                                         walltime=None, cputime=None, data=None)
```

Bases: object

A class wrapping segment data that must be passed through the work manager or data manager. Most fields are self-explanatory. One item worth noting is that a negative parent ID means that the segment starts from the initial state with ID -(segment.parent_id+1)

SEG_STATUS_UNSET = 0

SEG_STATUS_PREPARED = 1

SEG_STATUS_COMPLETE = 2

SEG_STATUS_FAILED = 3

SEG_INITPOINT_UNSET = 0

SEG_INITPOINT_CONTINUES = 1

SEG_INITPOINT_NEWTRAJ = 2

SEG_ENDPOINT_UNSET = 0

SEG_ENDPOINT_CONTINUES = 1

SEG_ENDPOINT_MERGED = 2

SEG_ENDPOINT_RECYCLED = 3

statuses = {'SEG_STATUS_COMPLETE': 2, 'SEG_STATUS_FAILED': 3, 'SEG_STATUS_PREPARED': 1, 'SEG_STATUS_UNSET': 0}

initpoint_types = {'SEG_INITPOINT_CONTINUES': 1, 'SEG_INITPOINT_NEWTRAJ': 2, 'SEG_INITPOINT_UNSET': 0}

endpoint_types = {'SEG_ENDPOINT_CONTINUES': 1, 'SEG_ENDPOINT_MERGED': 2, 'SEG_ENDPOINT_RECYCLED': 3, 'SEG_ENDPOINT_UNSET': 0}

status_names = {0: 'SEG_STATUS_UNSET', 1: 'SEG_STATUS_PREPARED', 2: 'SEG_STATUS_COMPLETE', 3: 'SEG_STATUS_FAILED'}

```
initpoint_type_names = {0: 'SEG_INITPOINT_UNSET', 1: 'SEG_INITPOINT_CONTINUES', 2:
'SEG_INITPOINT_NEWTRAJ'}

endpoint_type_names = {0: 'SEG_ENDPOINT_UNSET', 1: 'SEG_ENDPOINT_CONTINUES', 2:
'SEG_ENDPOINT_MERGED', 3: 'SEG_ENDPOINT_RECYCLED'}

static initial_pcoord(segment)
    Return the initial progress coordinate point of this segment.

static final_pcoord(segment)
    Return the final progress coordinate point of this segment.

property initpoint_type
property initial_state_id
property status_text
property endpoint_type_text

class westpa.cli.tools.w_dumpsecs.WDumpSegs
    Bases: WESTTool
    prog = 'w_dumpsecs'

    description = 'Dump segment data as text. This is very inefficient, so this tool
    should be used\nas a last resort (use hdfview/h5ls to look at data, and access HDF5
    directly for\nsignificant analysis tasks).\n'

    add_args(parser)
        Add arguments specific to this tool to the given argparse parser.

    process_args(args)
        Take argparse-processed arguments associated with this tool and deal with them appropriately (setting
        instance variables, etc)

    go()
        Perform the analysis associated with this tool.

westpa.cli.tools.w_dumpsecs.entry_point()
```

6.1.21.5 w_postanalysis_matrix

6.1.21.5.1 westpa.cli.tools.w_postanalysis_matrix module

```
class westpa.cli.tools.w_postanalysis_matrix.WESTMasterCommand
    Bases: WESTTool
    Base class for command-line tools that employ subcommands

    subparsers_title = None
    subcommands = None
    include_help_command = True
```

add_args(*parser*)

Add arguments specific to this tool to the given argparse parser.

process_args(*args*)

Take argparse-processed arguments associated with this tool and deal with them appropriately (setting instance variables, etc)

go()

Perform the analysis associated with this tool.

class westpa.cli.tools.w_postanalysis_matrix.**WESTParallelTool**(*wm_env=None*)

Bases: [WESTTool](#)

Base class for command-line tools parallelized with wwmgr. This automatically adds and processes wwmgr command-line arguments and creates a work manager at self.work_manager.

make_parser_and_process(*prog=None, usage=None, description=None, epilog=None, args=None*)

A convenience function to create a parser, call add_all_args(), and then call process_all_args(). The argument namespace is returned.

add_args(*parser*)

Add arguments specific to this tool to the given argparse parser.

process_args(*args*)

Take argparse-processed arguments associated with this tool and deal with them appropriately (setting instance variables, etc)

go()

Perform the analysis associated with this tool.

main()

A convenience function to make a parser, parse and process arguments, then run self.go() in the master process.

westpa.cli.tools.w_postanalysis_matrix.**warn**()

Issue a warning, or maybe ignore it or raise an exception.

message

Text of the warning message.

category

The Warning category subclass. Defaults to UserWarning.

stacklevel

How far up the call stack to make this warning appear. A value of 2 for example attributes the warning to the caller of the code calling warn().

source

If supplied, the destroyed object which emitted a ResourceWarning

skip_file_prefixes

An optional tuple of module filename prefixes indicating frames to skip during stacklevel computations for stack frame attribution.

class westpa.cli.tools.w_postanalysis_matrix.**RWMatrix**(*parent*)

Bases: [WESTKineticsBase](#), [FluxMatrix](#)

subcommand = 'init'

default_kinetics_file = 'reweight.h5'

default_output_file = 'reweight.h5'

```
help_text = 'create a color-labeled transition matrix from a WESTPA simulation'

description = 'Generate a colored transition matrix from a WE assignment file. The
subsequent\nanalysis requires that the assignments are calculated using only the
initial and\nfinal time points of each trajectory segment. This may require
downsampling the\nh5file generated by a WE simulation. In the future w_assign may be
enhanced to optionally\ngenerate the necessary assignment file from a h5file with
intermediate time points.\nAdditionally, this analysis is currently only valid on
simulations performed under\nneither equilibrium or steady-state conditions without
recycling target states.\n\
n-----\
nOutput
format\
n-----\n\
nThe output file (-o/--output, by default "reweight.h5") contains the\nfollowing
datasets:\n\n ``/bin_populations`` [window, bin]\n The reweighted populations of
each bin based on windows. Bins contain\n one color each, so to recover the original
un-colored spatial bins,\n one must sum over all states.\n\n ``/iterations``
[iteration]\n *(Structured -- see below)* Sparse matrix data from each\n iteration.
They are reconstructed and averaged within the\n w_reweight {kinetics/probs}
routines so that observables may\n be calculated. Each group contains 4 vectors of
data:\n\n flux\n *(Floating-point)* The weight of a series of flux events\n cols\n
*(Integer)* The bin from which a flux event began.\n cols\n *(Integer)* The bin into
which the walker fluxed.\n obs\n *(Integer)* How many flux events were observed
during this\n iteration.\n\
n-----\
nCommand-line
options\
n-----\n'
```

add_args(*parser*)

Add arguments specific to this component to the given argparse parser.

process_args(*args*)

Take argparse-processed arguments associated with this component and deal with them appropriately (setting instance variables, etc)

go()

class westpa.cli.tools.w_postanalysis_matrix.**PAMatrix**(*parent*)

Bases: [RWMMatrix](#)

subcommand = 'init'

help_text = 'averages and CIs for path-tracing kinetics analysis'

default_output_file = 'flux_matrices.h5'

class westpa.cli.tools.w_postanalysis_matrix.**WReweight**

Bases: [WESTMasterCommand](#), [WESTParallelTool](#)

prog = 'w_postanalysis_matrix'

subcommands = [[class](#) 'westpa.cli.tools.w_postanalysis_matrix.PAMatrix']>]

subparsers_title = 'calculate state-to-state kinetics by tracing trajectories'

```

description = 'Generate a colored transition matrix from a WE assignment file. The
subsequent\nanalysis requires that the assignments are calculated using only the
initial and\nfinal time points of each trajectory segment. This may require
downsampling the\nh5file generated by a WE simulation. In the future w_assign may be
enhanced to optionally\ngenerate the necessary assignment file from a h5file with
intermediate time points.\nAdditionally, this analysis is currently only valid on
simulations performed under\nneither equilibrium or steady-state conditions without
recycling target states.\n\
n-----\
nOutput
format\
n-----\n\
nThe output file (-o/--output, by default "reweight.h5") contains the\nfollowing
datasets:\n\n ``/bin_populations`` [window, bin]\n The reweighted populations of
each bin based on windows. Bins contain\n one color each, so to recover the original
un-colored spatial bins,\n one must sum over all states.\n\n ``/iterations``
[iteration]\n *(Structured -- see below)* Sparse matrix data from each\n iteration.
They are reconstructed and averaged within the\n w_reweight {kinetics/probs}
routines so that observables may\n be calculated. Each group contains 4 vectors of
data:\n\n flux\n *(Floating-point)* The weight of a series of flux events\n cols\n
*(Integer)* The bin from which a flux event began.\n cols\n *(Integer)* The bin into
which the walker fluxed.\n obs\n *(Integer)* How many flux events were observed
during this\n iteration.\n\
n-----\
nCommand-line
options\
n-----\n'

```

```
westpa.cli.tools.w_postanalysis_matrix.entry_point()
```

6.1.21.6 w_postanalysis_reweight

6.1.21.6.1 westpa.cli.tools.w_postanalysis_reweight module

```
class westpa.cli.tools.w_postanalysis_reweight.WESTMasterCommand
```

Bases: [WESTTool](#)

Base class for command-line tools that employ subcommands

subparsers_title = None

subcommands = None

include_help_command = True

add_args(*parser*)

Add arguments specific to this tool to the given argparse parser.

process_args(*args*)

Take argparse-processed arguments associated with this tool and deal with them appropriately (setting instance variables, etc)

go()

Perform the analysis associated with this tool.

```
class westpa.cli.tools.w_postanalysis_reweight.WESTParallelTool(wm_env=None)
```

Bases: [WESTTool](#)

Base class for command-line tools parallelized with wwmgr. This automatically adds and processes wwmgr command-line arguments and creates a work manager at self.work_manager.

```
make_parser_and_process(prog=None, usage=None, description=None, epilog=None, args=None)
```

A convenience function to create a parser, call add_all_args(), and then call process_all_args(). The argument namespace is returned.

```
add_args(parser)
```

Add arguments specific to this tool to the given argparse parser.

```
process_args(args)
```

Take argparse-processed arguments associated with this tool and deal with them appropriately (setting instance variables, etc)

```
go()
```

Perform the analysis associated with this tool.

```
main()
```

A convenience function to make a parser, parse and process arguments, then run self.go() in the master process.

```
westpa.cli.tools.w_postanalysis_reweight.warn()
```

Issue a warning, or maybe ignore it or raise an exception.

```
message
```

Text of the warning message.

```
category
```

The Warning category subclass. Defaults to UserWarning.

```
stacklevel
```

How far up the call stack to make this warning appear. A value of 2 for example attributes the warning to the caller of the code calling warn().

```
source
```

If supplied, the destroyed object which emitted a ResourceWarning

```
skip_file_prefixes
```

An optional tuple of module filename prefixes indicating frames to skip during stacklevel computations for stack frame attribution.

```
class westpa.cli.tools.w_postanalysis_reweight.RWAverage(parent)
```

Bases: [RWStateProbs](#), [RWRate](#)

```
subcommand = 'average'
```

```
help_text = 'Averages and returns fluxes, rates, and color/state populations.'
```

```
default_kinetics_file = 'reweight.h5'
```

```
default_output_file = 'reweight.h5'
```

```
description = 'A convenience function to run kinetics/probs. Bin
assignments,\nincluding macrostate definitions, are required. (See\n"nw_assign
--help" for more information).\n\nFor more information on the individual subcommands
this subs in for, run\nnw_reweight {kinetics/probs} --help.\n\
n-----\
nCommand-line
options\
n-----\n'
```

```

go()

class westpa.cli.tools.w_postanalysis_reweight.PAAverage(parent)
    Bases: RWAverage
    subcommand = 'average'
    help_text = ''
    default_output_file = 'kinrw.h5'
    default_kinetics_file = 'flux_matrices.h5'

class westpa.cli.tools.w_postanalysis_reweight.WRweight
    Bases: WESTMasterCommand, WESTParallelTool
    prog = 'w_postanalysis_reweight'
    subcommands = [<class 'westpa.cli.tools.w_postanalysis_reweight.PAAverage'>]
    subparsers_title = 'calculate state-to-state kinetics by tracing trajectories'
    description = 'A convenience function to run kinetics/probs. Bin
    assignments,\nincluding macrostate definitions, are required. (See\n"w_assign
    --help" for more information).\n\nFor more information on the individual subcommands
    this subs in for, run\nw_reweight {kinetics/probs} --help.\n\
    n-----\
    nCommand-line
    options\
    n-----\n'

westpa.cli.tools.w_postanalysis_reweight.entry_point()

```

6.1.21.7 w_reweight

6.1.21.7.1 westpa.cli.tools.w_reweight module

```

class westpa.cli.tools.w_reweight.WESTMasterCommand
    Bases: WESTTool
    Base class for command-line tools that employ subcommands
    subparsers_title = None
    subcommands = None
    include_help_command = True
    add_args(parser)
        Add arguments specific to this tool to the given argparse parser.
    process_args(args)
        Take argparse-processed arguments associated with this tool and deal with them appropriately (setting
        instance variables, etc)

```

go()

Perform the analysis associated with this tool.

class westpa.cli.tools.w_reweight.**WESTParallelTool**(wm_env=None)

Bases: [WESTTool](#)

Base class for command-line tools parallelized with wwmgr. This automatically adds and processes wwmgr command-line arguments and creates a work manager at self.work_manager.

make_parser_and_process(prog=None, usage=None, description=None, epilog=None, args=None)

A convenience function to create a parser, call add_all_args(), and then call process_all_args(). The argument namespace is returned.

add_args(parser)

Add arguments specific to this tool to the given argparse parser.

process_args(args)

Take argparse-processed arguments associated with this tool and deal with them appropriately (setting instance variables, etc)

go()

Perform the analysis associated with this tool.

main()

A convenience function to make a parser, parse and process arguments, then run self.go() in the master process.

class westpa.cli.tools.w_reweight.**WESTKineticsBase**(parent)

Bases: [WESTSubcommand](#)

Common argument processing for w_direct/w_reweight subcommands. Mostly limited to handling input and output from w_assign.

add_args(parser)

Add arguments specific to this component to the given argparse parser.

process_args(args)

Take argparse-processed arguments associated with this component and deal with them appropriately (setting instance variables, etc)

class westpa.cli.tools.w_reweight.**AverageCommands**(parent)

Bases: [WESTKineticsBase](#)

default_output_file = 'direct.h5'

add_args(parser)

Add arguments specific to this component to the given argparse parser.

process_args(args)

Take argparse-processed arguments associated with this component and deal with them appropriately (setting instance variables, etc)

stamp_mcbs_info(dataset)

open_files()

open_assignments()

print_averages(dataset, header, dim=1)


```
run_calculation(pi, nstates, start_iter, stop_iter, step_iter, dataset, eval_block, name, dim,
                 do_averages=False, **extra)
```

```
westpa.cli.tools.w_reweight.generate_future(work_manager, name, eval_block, kwargs)
```

```
westpa.cli.tools.w_reweight.mcbs_ci_correl(estimator_datasets, estimator, alpha, n_sets=None,
                                             args=None, autocorrel_alpha=None,
                                             autocorrel_n_sets=None, subsample=None,
                                             do_correl=True, mcbs_enable=None,
                                             estimator_kwargs={})
```

Perform a Monte Carlo bootstrap estimate for the $(1-\alpha)$ confidence interval on the given dataset with the given estimator. This routine is appropriate for time-correlated data, using the method described in Huber & Kim, “Weighted-ensemble Brownian dynamics simulations for protein association reactions” (1996), doi:10.1016/S0006-3495(96)79552-8 to determine a statistically-significant correlation time and then reducing the dataset by a factor of that correlation time before running a “classic” Monte Carlo bootstrap.

Returns (estimate, ci_lb, ci_ub, correl_time) where estimate is the application of the given estimator to the input dataset, ci_lb and ci_ub are the lower and upper limits, respectively, of the $(1-\alpha)$ confidence interval on estimate, and correl_time is the correlation time of the dataset, significant to $(1-\text{autocorrel_alpha})$.

estimator is called as **estimator(dataset, *args, **kwargs)**. Common estimators include:

- np.mean – calculate the confidence interval on the mean of dataset
- np.median – calculate a confidence interval on the median of dataset
- np.std – calculate a confidence interval on the standard deviation of dataset.

n_sets is the number of synthetic data sets to generate using the given estimator, which will be chosen using `get_bssize()` if *n_sets* is not given.

autocorrel_alpha (which defaults to *alpha*) can be used to adjust the significance level of the autocorrelation calculation. Note that too high a significance level (too low an *alpha*) for evaluating the significance of autocorrelation values can result in a failure to detect correlation if the autocorrelation function is noisy.

The given *subsample* function is used, if provided, to subsample the dataset prior to running the full Monte Carlo bootstrap. If none is provided, then a random entry from each correlated block is used as the value for that block. Other reasonable choices include `np.mean`, `np.median`, `(lambda x: x[0])` or `(lambda x: x[-1])`. In particular, using *subsample*=`np.mean` will converge to the block averaged mean and standard error, while accounting for any non-normality in the distribution of the mean.

```
westpa.cli.tools.w_reweight.reweight_for_c(rows, cols, obs, flux, insert, indices, nstates, nbins,
                                             state_labels, state_map, nfbins, istate, jstate, stride,
                                             bin_last_state_map, bin_state_map, return_obs,
                                             obs_threshold=1)
```

```
class westpa.cli.tools.w_reweight.FluxMatrix
```

Bases: object

```
w_postanalysis_matrix()
```

```
class westpa.cli.tools.w_reweight.RWMatrix(parent)
```

Bases: `WESTKineticsBase`, `FluxMatrix`

```
subcommand = 'init'
```

```
default_kinetics_file = 'reweight.h5'
```

```
default_output_file = 'reweight.h5'
```

```
help_text = 'create a color-labeled transition matrix from a WESTPA simulation'
```

```
description = 'Generate a colored transition matrix from a WE assignment file. The
subsequent analysis requires that the assignments are calculated using only the
initial and final time points of each trajectory segment. This may require
downsampling the h5file generated by a WE simulation. In the future w_assign may be
enhanced to optionally generate the necessary assignment file from a h5file with
intermediate time points. Additionally, this analysis is currently only valid on
simulations performed under neither equilibrium or steady-state conditions without
recycling target states.'
```

```
n-----\
nOutput
format\
n-----\n\
nThe output file (-o/--output, by default "reweight.h5") contains the following
datasets:\n\n ``/bin_populations`` [window, bin]\n The reweighted populations of
each bin based on windows. Bins contain one color each, so to recover the original
un-colored spatial bins, one must sum over all states.\n\n ``/iterations``
[iteration]\n *(Structured -- see below)* Sparse matrix data from each iteration.
They are reconstructed and averaged within the w_reweight {kinetics/probs}
routines so that observables may be calculated. Each group contains 4 vectors of
data:\n\n flux\n *(Floating-point)* The weight of a series of flux events\n cols\n
*(Integer)* The bin from which a flux event began.\n cols\n *(Integer)* The bin into
which the walker fluxed.\n obs\n *(Integer)* How many flux events were observed
during this iteration.\n\n
n-----\
nCommand-line
options\
n-----\n'
```

add_args(parser)

Add arguments specific to this component to the given argparse parser.

process_args(args)

Take argparse-processed arguments associated with this component and deal with them appropriately (setting instance variables, etc)

go()

class westpa.cli.tools.w_reweight.RWReweight(parent)

Bases: [AverageCommands](#)

help_text = 'Parent class for all reweighting routines, as they all use the same estimator code.'

add_args(parser)

Add arguments specific to this component to the given argparse parser.

process_args(args)

Take argparse-processed arguments associated with this component and deal with them appropriately (setting instance variables, etc)

accumulate_statistics(start_iter, stop_iter)

This function pulls previously generated flux matrix data into memory. The data is assumed to exist within an HDF5 file that is available as a property. The data is kept as a single dimensional numpy array to use with the cython estimator.

generate_reweight_data()

This function ensures all the appropriate files are loaded, sets appropriate attributes necessary for all calling functions/children, and then calls the function to load in the flux matrix data.

```
class westpa.cli.tools.w_reweight.RWRate(parent)
```

Bases: [*RWReweight*](#)

```
subcommand = 'kinetics'
```

```
help_text = 'Generates rate and flux values from a WESTPA simulation via  
reweighting.'
```

```
default_kinetics_file = 'reweight.h5'
```

```
default_output_file = 'reweight.h5'
```

```

description = 'Calculate average rates from weighted ensemble data using the
postanalysis\nreweighting scheme. Bin assignments (usually "assign.h5") and
pre-calculated\niteration flux matrices (usually "reweight.h5") data files must have
been\npreviously generated using w_reweight matrix (see "w_assign --help"
and\n"w_reweight init --help" for information on generating these files).\n\
n-----\
nOutput
format\
n-----\n
The
output file (-o/--output, usually "kinrw.h5") contains the following\ndataset:\n\n
/avg_rates [state,state]\n (Structured -- see below) State-to-state rates based on
entire window of\n iterations selected.\n\n
/avg_total_fluxes [state]\n (Structured
-- see below) Total fluxes into each state based on entire\n window of iterations
selected.\n\n
/avg_conditional_fluxes [state,state]\n (Structured -- see below)
State-to-state fluxes based on entire window of\n iterations selected.\n\n
If
--evolution-mode is specified, then the following additional datasets
are\navailable:\n\n
/rate_evolution [window][state][state]\n (Structured -- see
below). State-to-state rates based on windows of\n iterations of varying width. If
--evolution-mode=cumulative, then\n these windows all begin at the iteration
specified with\n --start-iter and grow in length by --step-iter for each
successive\n element. If --evolution-mode=blocked, then these windows are all of\n
width --step-iter (excluding the last, which may be shorter), the first\n of which
begins at iteration --start-iter.\n\n
/target_flux_evolution [window,state]\n
(Structured -- see below). Total flux into a given macro state based on\n windows of
iterations of varying width, as in /rate_evolution.\n\n
/conditional_flux_evolution
[window,state,state]\n (Structured -- see below). State-to-state fluxes based on
windows of\n varying width, as in /rate_evolution.\n\n
The structure of these
datasets is as follows:\n\n
iter_start\n (Integer) Iteration at which the averaging
window begins (inclusive).\n\n
iter_stop\n (Integer) Iteration at which the
averaging window ends (exclusive).\n\n
expected\n (Floating-point) Expected (mean)
value of the observable as evaluated within\n this window, in units of inverse
tau.\n\n
ci_lbound\n (Floating-point) Lower bound of the confidence interval of the
observable\n within this window, in units of inverse tau.\n\n
ci_ubound\n
(Floating-point) Upper bound of the confidence interval of the observable\n within
this window, in units of inverse tau.\n\n
stderr\n (Floating-point) The standard
error of the mean of the observable\n within this window, in units of inverse
tau.\n\n
corr_len\n (Integer) Correlation length of the observable within this
window, in units\n of tau.\n\n
Each of these datasets is also stamped with a number
of attributes:\n\n
mcbs_alpha\n (Floating-point) Alpha value of confidence
intervals. (For example,\n *alpha=0.05* corresponds to a 95% confidence
interval.)\n\n
mcbs_nsets\n (Integer) Number of bootstrap data sets used in
generating confidence\n intervals.\n\n
mcbs_acalpha\n (Floating-point) Alpha value
for determining correlation lengths.\n\n\
n-----\
nCommand-line
options\
n-----\n '

```

w_postanalysis_reweight()

This function ensures the data is ready to send in to the estimator and the bootstrapping routine, then does so. Much of this is simply setting up appropriate args and kwargs, then passing them in to the 'run_calculation' function, which sets up future objects to send to the work manager. The results are returned, and then written to the appropriate HDF5 dataset. This function is specific for the rates and fluxes from the reweighting method.

`go()`

`class westpa.cli.tools.w_reweight.RWStateProbs(parent)`

Bases: [*RWReweight*](#)

`subcommand = 'probs'`

`help_text = 'Calculates color and state probabilities via reweighting.'`

`default_kinetics_file = 'reweight.h5'`

```

description = 'Calculate average populations from weighted ensemble data using the
postanalysis\nreweighting scheme. Bin assignments (usually "assign.h5") and
pre-calculated\niteration flux matrices (usually "reweight.h5") data files must have
been\npreviously generated using w_reweight matrix (see "w_assign --help"
and\n"w_reweight init --help" for information on generating these files).\n\
n-----\n
nOutput
format\
n-----\n
nThe output file (-o/--output, usually "direct.h5") contains the
following\ndataset:\n\n /avg_state_probs [state]\n (Structured -- see below)
Population of each state across entire\n range specified.\n\n /avg_color_probs
[state]\n (Structured -- see below) Population of each ensemble across entire\n
range specified.\n\nIf --evolution-mode is specified, then the following additional
datasets are\navailable:\n\n /state_pop_evolution [window][state]\n (Structured --
see below). State populations based on windows of\n iterations of varying width. If
--evolution-mode=cumulative, then\n these windows all begin at the iteration
specified with\n --start-iter and grow in length by --step-iter for each
successive\n element. If --evolution-mode=blocked, then these windows are all of\n
width --step-iter (excluding the last, which may be shorter), the first\n of which
begins at iteration --start-iter.\n\n /color_prob_evolution [window][state]\n
(Structured -- see below). Ensemble populations based on windows of\n iterations of
varying width. If --evolution-mode=cumulative, then\n these windows all begin at the
iteration specified with\n --start-iter and grow in length by --step-iter for each
successive\n element. If --evolution-mode=blocked, then these windows are all of\n
width --step-iter (excluding the last, which may be shorter), the first\n of which
begins at iteration --start-iter.\n\nThe structure of these datasets is as
follows:\n\n iter_start\n (Integer) Iteration at which the averaging window begins
(inclusive).\n\n iter_stop\n (Integer) Iteration at which the averaging window ends
(exclusive).\n\n expected\n (Floating-point) Expected (mean) value of the observable
as evaluated within\n this window, in units of inverse tau.\n\n ci_lbound\n
(Floating-point) Lower bound of the confidence interval of the observable\n within
this window, in units of inverse tau.\n\n ci_ubound\n (Floating-point) Upper bound
of the confidence interval of the observable\n within this window, in units of
inverse tau.\n\n stderr\n (Floating-point) The standard error of the mean of the
observable\n within this window, in units of inverse tau.\n\n corr_len\n (Integer)
Correlation length of the observable within this window, in units\n of
tau.\n\nEach of these datasets is also stamped with a number of attributes:\n\n
mcbs_alpha\n (Floating-point) Alpha value of confidence intervals. (For example,\n
*alpha=0.05* corresponds to a 95% confidence interval.)\n\n mcbs_nsets\n (Integer)
Number of bootstrap data sets used in generating confidence\n intervals.\n\n
mcbs_acalpha\n (Floating-point) Alpha value for determining correlation lengths.\n\
n-----\n
nCommand-line
options\
n-----\n'

```

w_postanalysis_stateprobs()

This function ensures the data is ready to send in to the estimator and the bootstrapping routine, then does so. Much of this is simply setting up appropriate args and kwargs, then passing them in to the 'run_calculation' function, which sets up future objects to send to the work manager. The results are returned, and then written to the appropriate HDF5 dataset. This function is specific for the color (steady-state) and macrostate probabilities from the reweighting method.

go()

```

class westpa.cli.tools.w_reweight.RWAll(parent)
    Bases: RWMMatrix, RWStateProbs, RWRate

    subcommand = 'all'

    help_text = 'Runs the full suite, including the generation of the flux matrices.'

    default_kinetics_file = 'reweight.h5'

    default_output_file = 'reweight.h5'

    description = 'A convenience function to run init/kinetics/probs. Bin
    assignments,\nincluding macrostate definitions, are required. (See\n"w_assign
    --help" for more information).\n\nFor more information on the individual subcommands
    this subs in for, run\nnw_reweight {init/kinetics/probs} --help.\n\
    n-----\
    nCommand-line
    options\
    n-----\n'

    go()

class westpa.cli.tools.w_reweight.RWAverage(parent)
    Bases: RWStateProbs, RWRate

    subcommand = 'average'

    help_text = 'Averages and returns fluxes, rates, and color/state populations.'

    default_kinetics_file = 'reweight.h5'

    default_output_file = 'reweight.h5'

    description = 'A convenience function to run kinetics/probs. Bin
    assignments,\nincluding macrostate definitions, are required. (See\n"w_assign
    --help" for more information).\n\nFor more information on the individual subcommands
    this subs in for, run\nnw_reweight {kinetics/probs} --help.\n\
    n-----\
    nCommand-line
    options\
    n-----\n'

    go()

class westpa.cli.tools.w_reweight.WRweight
    Bases: WESTMasterCommand, WESTParallelTool

    prog = 'w_reweight'

    subcommands = [<class 'westpa.cli.tools.w_reweight.RWMMatrix'>, <class
    'westpa.cli.tools.w_reweight.RWAverage'>, <class
    'westpa.cli.tools.w_reweight.RWRate'>, <class
    'westpa.cli.tools.w_reweight.RWStateProbs'>, <class
    'westpa.cli.tools.w_reweight.RWAll'>]

    subparsers_title = 'reweighting kinetics analysis scheme'

westpa.cli.tools.w_reweight.entry_point()

```

6.1.21.8 w_fluxanl

w_fluxanl calculates the probability flux of a weighted ensemble simulation based on a pre-defined target state. Also calculates confidence interval of average flux. Monte Carlo bootstrapping techniques are used to account for autocorrelation between fluxes and/or errors that are not normally distributed.

6.1.21.8.1 Overview

usage:

```
w_fluxanl [-h] [-r RCFILE] [--quiet | --verbose | --debug] [--version]
          [-W WEST_H5FILE] [-o OUTPUT]
          [--first-iter N_ITER] [--last-iter N_ITER]
          [-a ALPHA] [--autocorrel-alpha ACALPHA] [-N NSETS] [--evol] [--
↪evol-step ESTEP]
```

Note: All command line arguments are optional for w_fluxanl.

6.1.21.8.2 Command-Line Options

See the [general command-line tool reference](#) for more information on the general options.

Input/output options

These arguments allow the user to specify where to read input simulation result data and where to output calculated progress coordinate probability distribution data.

Both input and output files are *hdf5* format.:

```
-W, --west-data file
  Read simulation result data from file *file*. (**Default:** The
  *hdf5* file specified in the configuration file)

-o, --output file
  Store this tool's output in *file*. (**Default:** The *hdf5* file
  **pcpdist.h5**)
```

Iteration range options

Specify the range of iterations over which to construct the progress coordinate probability distribution.:

```
--first-iter n_iter
  Construct probability distribution starting with iteration *n_iter*
  (**Default:** 1)

--last-iter n_iter
  Construct probability distribution's time evolution up to (and
  including) iteration *n_iter* (**Default:** Last completed
  iteration)
```


Confidence interval and bootstrapping options

Specify alpha values of constructed confidence intervals.:

```
-a alpha
  Calculate a (1 - *alpha*) confidence interval for the mean flux
  (**Default:** 0.05)

--autocorrel-alpha ACalpha
  Identify autocorrelation of fluxes at *ACalpha* significance level.
  Note: Specifying an *ACalpha* level that is too small may result in
  failure to find autocorrelation in noisy flux signals (**Default:**
  Same level as *alpha*)

-N n_sets, --nsets n_sets
  Use *n_sets* samples for bootstrapping (**Default:** Chosen based
  on *alpha*)

--evol
  Calculate the time evolution of flux confidence intervals
  (**Warning:** computationally expensive calculation)

--evol-step estep
  (if ``--evol`` specified) Calculate the time evolution of flux
  confidence intervals for every *estep* iterations (**Default:** 1)
```

6.1.21.8.3 Examples

Calculate the time evolution flux every 5 iterations:

```
w_fluxanl --evol --evol-step 5
```

Calculate mean flux confidence intervals at 0.01 significance level and calculate autocorrelations at 0.05 significance:

```
w_fluxanl --alpha 0.01 --autocorrel-alpha 0.05
```

Calculate the mean flux confidence intervals using a custom bootstrap sample size of 500:

```
w_fluxanl --n-sets 500
```

6.1.21.8.4 westpa.cli.tools.w_fluxanl module

`westpa.cli.tools.w_fluxanl.fftconvolve(in1, in2, mode='full', axes=None)`

Convolve two N-dimensional arrays using FFT.

Convolve *in1* and *in2* using the fast Fourier transform method, with the output size determined by the *mode* argument.

This is generally much faster than *convolve* for large arrays ($n > \sim 500$), but can be slower when only a few output values are needed, and can only output float arrays (int or object array inputs will be cast to float).

As of v0.19, *convolve* automatically chooses this method or the direct method based on an estimation of which is faster.

Parameters

- **in1** (*array_like*) – First input.
- **in2** (*array_like*) – Second input. Should have the same number of dimensions as *in1*.
- **mode** (*str* {'full', 'valid', 'same'}, *optional*) – A string indicating the size of the output:

full

The output is the full discrete linear convolution of the inputs. (Default)

valid

The output consists only of those elements that do not rely on the zero-padding. In 'valid' mode, either *in1* or *in2* must be at least as large as the other in every dimension.

same

The output is the same size as *in1*, centered with respect to the 'full' output.

- **axes** (*int or array_like of ints or None, optional*) – Axes over which to compute the convolution. The default is over all axes.

Returns

out – An N-dimensional array containing a subset of the discrete linear convolution of *in1* with *in2*.

Return type

array

See also:

convolve

Uses the direct convolution or FFT convolution algorithm depending on which is faster.

oaconvolve

Uses the overlap-add method to do convolution, which is generally faster when the input arrays are large and significantly different in size.

Examples

Autocorrelation of white noise is an impulse.

```
>>> import numpy as np
>>> from scipy import signal
>>> rng = np.random.default_rng()
>>> sig = rng.standard_normal(1000)
>>> autocorr = signal.fftconvolve(sig, sig[::-1], mode='full')
```

```
>>> import matplotlib.pyplot as plt
>>> fig, (ax_orig, ax_mag) = plt.subplots(2, 1)
>>> ax_orig.plot(sig)
>>> ax_orig.set_title('White noise')
>>> ax_mag.plot(np.arange(-len(sig)+1, len(sig)), autocorr)
>>> ax_mag.set_title('Autocorrelation')
>>> fig.tight_layout()
>>> fig.show()
```

Gaussian blur implemented using FFT convolution. Notice the dark borders around the image, due to the zero-padding beyond its boundaries. The *convolve2d* function allows for other types of image boundaries, but is far

slower.

```
>>> from scipy import datasets
>>> face = datasets.face(gray=True)
>>> kernel = np.outer(signal.windows.gaussian(70, 8),
...                  signal.windows.gaussian(70, 8))
>>> blurred = signal.fftconvolve(face, kernel, mode='same')
```

```
>>> fig, (ax_orig, ax_kernel, ax_blurred) = plt.subplots(3, 1,
...                                                    figsize=(6, 15))
>>> ax_orig.imshow(face, cmap='gray')
>>> ax_orig.set_title('Original')
>>> ax_orig.set_axis_off()
>>> ax_kernel.imshow(kernel, cmap='gray')
>>> ax_kernel.set_title('Gaussian kernel')
>>> ax_kernel.set_axis_off()
>>> ax_blurred.imshow(blurred, cmap='gray')
>>> ax_blurred.set_title('Blurred')
>>> ax_blurred.set_axis_off()
>>> fig.show()
```

`westpa.cli.tools.w_fluxanl.warn()`

Issue a warning, or maybe ignore it or raise an exception.

message

Text of the warning message.

category

The Warning category subclass. Defaults to UserWarning.

stacklevel

How far up the call stack to make this warning appear. A value of 2 for example attributes the warning to the caller of the code calling `warn()`.

source

If supplied, the destroyed object which emitted a ResourceWarning

skip_file_prefixes

An optional tuple of module filename prefixes indicating frames to skip during stacklevel computations for stack frame attribution.

`westpa.cli.tools.w_fluxanl.weight_dtype`

alias of float64

`westpa.cli.tools.w_fluxanl.n_iter_dtype`

alias of uint32

```
class westpa.cli.tools.w_fluxanl.NewWeightEntry(source_type, weight, prev_seg_id=None,
                                                prev_init_pcoord=None, prev_final_pcoord=None,
                                                new_init_pcoord=None, target_state_id=None,
                                                initial_state_id=None)
```

Bases: object

NW_SOURCE_RECYCLED = 0

class westpa.cli.tools.w_fluxanl.WESTTool

Bases: [WESTToolComponent](#)

Base class for WEST command line tools

prog = None

usage = None

description = None

epilog = None

add_args(*parser*)

Add arguments specific to this tool to the given argparse parser.

process_args(*args*)

Take argparse-processed arguments associated with this tool and deal with them appropriately (setting instance variables, etc)

make_parser(*prog=None, usage=None, description=None, epilog=None, args=None*)

make_parser_and_process(*prog=None, usage=None, description=None, epilog=None, args=None*)

A convenience function to create a parser, call `add_all_args()`, and then call `process_all_args()`. The argument namespace is returned.

go()

Perform the analysis associated with this tool.

main()

A convenience function to make a parser, parse and process arguments, then call `self.go()`

class westpa.cli.tools.w_fluxanl.**WESTDataReader**

Bases: [*WESTToolComponent*](#)

Tool for reading data from WEST-related HDF5 files. Coordinates finding the main HDF5 file from `west.cfg` or command line arguments, caching of certain kinds of data (eventually), and retrieving auxiliary data sets from various places.

add_args(*parser*)

Add arguments specific to this component to the given argparse parser.

process_args(*args*)

Take argparse-processed arguments associated with this component and deal with them appropriately (setting instance variables, etc)

open(*mode='r'*)

close()

property `weight_dsspec`

property `parent_id_dsspec`

class westpa.cli.tools.w_fluxanl.**IterRangeSelection**(*data_manager=None*)

Bases: [*WESTToolComponent*](#)

Select and record limits on iterations used in analysis and/or reporting. This class provides both the user-facing command-line options and parsing, and the application-side API for recording limits in HDF5.

HDF5 datasets calculated based on a restricted set of iterations should be tagged with the following attributes:

first_iter

The first iteration included in the calculation.

last_iter

One past the last iteration included in the calculation.

iter_step

Blocking or sampling period for iterations included in the calculation.

add_args(*parser*)

Add arguments specific to this component to the given argparse parser.

process_args(*args*, *override_iter_start=None*, *override_iter_stop=None*, *default_iter_step=1*)

Take argparse-processed arguments associated with this component and deal with them appropriately (setting instance variables, etc)

iter_block_iter()

Return an iterable of (block_start,block_end) over the blocks of iterations selected by `--first-iter/--last-iter/--step-iter`.

n_iter_blocks()

Return the number of blocks of iterations (as returned by `iter_block_iter`) selected by `--first-iter/--last-iter/--step-iter`.

record_data_iter_range(*h5object*, *iter_start=None*, *iter_stop=None*)

Store attributes `iter_start` and `iter_stop` on the given HDF5 object (group/dataset)

record_data_iter_step(*h5object*, *iter_step=None*)

Store attribute `iter_step` on the given HDF5 object (group/dataset).

check_data_iter_range_least(*h5object*, *iter_start=None*, *iter_stop=None*)

Check that the given HDF5 object contains (as denoted by its `iter_start/iter_stop` attributes) data at least for the iteration range specified.

check_data_iter_range_equal(*h5object*, *iter_start=None*, *iter_stop=None*)

Check that the given HDF5 object contains (as denoted by its `iter_start/iter_stop` attributes) data exactly for the iteration range specified.

check_data_iter_step_conformant(*h5object*, *iter_step=None*)

Check that the given HDF5 object contains per-iteration data at an iteration stride suitable for extracting data with the given stride (in other words, the given `iter_step` is a multiple of the stride with which data was recorded).

check_data_iter_step_equal(*h5object*, *iter_step=None*)

Check that the given HDF5 object contains per-iteration data at an iteration stride the same as that specified.

slice_per_iter_data(*dataset*, *iter_start=None*, *iter_stop=None*, *iter_step=None*, *axis=0*)

Return the subset of the given dataset corresponding to the given iteration range and stride. Unless otherwise specified, the first dimension of the dataset is the one sliced.

iter_range(*iter_start=None*, *iter_stop=None*, *iter_step=None*, *dtype=None*)

Return a sequence for the given iteration numbers and stride, filling in missing values from those stored on `self`. The smallest data type capable of holding `iter_stop` is returned unless otherwise specified using the `dtype` argument.

westpa.cli.tools.w_fluxanl.extract_fluxes(*iter_start=None*, *iter_stop=None*, *data_manager=None*)

Extract flux values from the WEST HDF5 file for iterations \geq `iter_start` and $<$ `iter_stop`, optionally using another data manager instance instead of the global one returned by `westpa.rc.get_data_manager()`.

Returns a dictionary mapping target names (if available, target index otherwise) to a 1-D array of type `fluxentry_dtype`, which contains columns for iteration number, flux, and count.

class westpa.cli.tools.w_fluxanl.WFluxanlTool

Bases: [WESTTool](#)

```
prog = 'w_fluxanl'

description = 'Extract fluxes into pre-defined target states from WEST
data,\naverage, and construct confidence intervals. Monte Carlo bootstrapping\nis
used to account for the correlated and possibly non-Gaussian statistical\nerror in
flux measurements.\n\nAll non-graphical output (including that to the terminal and
HDF5) assumes that\nthe propagation/resampling period ``tau`` is equal to unity; to
obtain results\nin familiar units, divide all fluxes and multiply all correlation
lengths by\nthe true value of ``tau``.\n'

output_format_version = 2

add_args(parser)
    Add arguments specific to this tool to the given argparse parser.

process_args(args)
    Take argparse-processed arguments associated with this tool and deal with them appropriately (setting
    instance variables, etc)

calc_store_flux_data()

calc_evol_flux()

go()
    Perform the analysis associated with this tool.

westpa.cli.tools.w_fluxanl.entry_point()
```

6.2 westpa.core package

6.2.1 westpa.core.binning package

6.2.1.1 westpa.core.binning module

class westpa.core.binning.NopMapper

Bases: [BinMapper](#)

Put everything into one bin.

assign(coords, mask=None, output=None)

class westpa.core.binning.FuncBinMapper(func, nbins, args=None, kwargs=None)

Bases: [BinMapper](#)

Binning using a custom function which must iterate over input coordinate sets itself.

assign(coords, mask=None, output=None)

class westpa.core.binning.PiecewiseBinMapper(functions)

Bases: [BinMapper](#)

Binning using a set of functions returning boolean values; if the Nth function returns True for a coordinate tuple, then that coordinate is in the Nth bin.

assign(coords, mask=None, output=None)

class westpa.core.binning.RectilinearBinMapper(*boundaries*)

Bases: [BinMapper](#)

Bin into a rectangular grid based on tuples of float values

property boundaries

assign(*coords*, *mask=None*, *output=None*)

class westpa.core.binning.RecursiveBinMapper(*base_mapper*, *start_index=0*)

Bases: [BinMapper](#)

Nest mappers one within another.

property labels

property start_index

add_mapper(*mapper*, *replaces_bin_at*)

Replace the bin containing the coordinate tuple *replaces_bin_at* with the specified mapper.

assign(*coords*, *mask=None*, *output=None*)

class westpa.core.binning.VectorizingFuncBinMapper(*func*, *nbins*, *args=None*, *kwargs=None*)

Bases: [BinMapper](#)

Binning using a custom function which is evaluated once for each (unmasked) coordinate tuple provided.

assign(*coords*, *mask=None*, *output=None*)

class westpa.core.binning.VoronoiBinMapper(*dfunc*, *centers*, *dfargs=None*, *dfkwargs=None*)

Bases: [BinMapper](#)

A one-dimensional mapper which assigns a multidimensional pcoord to the closest center based on a distance metric. Both the list of centers and the distance function must be supplied.

assign(*coords*, *mask=None*, *output=None*)

westpa.core.binning.map_mab(*coords*, *mask*, *output*, **args*, ***kwargs*)

Binning which adaptively places bins based on the positions of extrema segments and bottleneck segments, which are where the difference in probability is the greatest along the progress coordinate. Operates per dimension and places a fixed number of evenly spaced bins between the segments with the min and max pcoord values. Extrema and bottleneck segments are assigned their own bins.

westpa.core.binning.map_binless(*coords*, *mask*, *output*, **args*, ***kwargs*)

Adaptively groups walkers according to a user-defined grouping function that is defined externally. Very general implementation but limited to only a two dimensional progress coordinate (for now).

class westpa.core.binning.MABBinMapper(*nbins*, *direction=None*, *skip=None*, *bottleneck=True*, *pca=False*, *mab_log=False*, *bin_log=False*)

Bases: [FuncBinMapper](#)

Adaptively place bins in between minimum and maximum segments along the progress coordinte. Extrema and bottleneck segments are assigned to their own bins.

class westpa.core.binning.BinlessMapper(*ngroups*, *ndims*, *group_function*, ***group_function_kwargs*)

Bases: [FuncBinMapper](#)

Adaptively group walkers according to a user-defined grouping function that is defined externally.

```
class westpa.core.binning.MABDriver(rc=None, system=None)
```

Bases: [WEDriver](#)

```
assign(segments, initializing=False)
```

Assign segments to initial and final bins, and update the (internal) lists of used and available initial states. This function is adapted to the MAB scheme, so that the initial and final segments are sent to the bin mapper at the same time, otherwise the initial and final bin boundaries can be inconsistent.

```
class westpa.core.binning.MABSimManager(rc=None)
```

Bases: [WESimManager](#)

```
initialize_simulation(basis_states, target_states, start_states, segs_per_state=1, suppress_we=False)
```

Initialize a new weighted ensemble simulation, taking `segs_per_state` initial states from each of the given `basis_states`.

`w_init` is the forward-facing version of this function

```
propagate()
```

```
prepare_iteration()
```

```
class westpa.core.binning.BinlessDriver(rc=None, system=None)
```

Bases: [WEDriver](#)

```
assign(segments, initializing=False)
```

Assign segments to initial and final bins, and update the (internal) lists of used and available initial states. This function is adapted to the MAB scheme, so that the initial and final segments are sent to the bin mapper at the same time, otherwise the initial and final bin boundaries can be inconsistent.

```
class westpa.core.binning.BinlessSimManager(rc=None)
```

Bases: [WESimManager](#)

```
initialize_simulation(basis_states, target_states, start_states, segs_per_state=1, suppress_we=False)
```

Initialize a new weighted ensemble simulation, taking `segs_per_state` initial states from each of the given `basis_states`.

`w_init` is the forward-facing version of this function

```
propagate()
```

```
prepare_iteration()
```

```
westpa.core.binning.accumulate_labeled_populations(weights, bin_assignments, label_assignments,
                                                    labeled_bin_pops)
```

For a set of segments in one iteration, calculate the average population in each bin, with separation by last-visited macrostate.

```
westpa.core.binning.assign_and_label(nsegs_lb, nsegs_ub, parent_ids, assign, nstates, state_map,
                                     last_labels, pcoords, subsample)
```

Assign trajectories to bins and last-visited macrostates for each timepoint.

```
westpa.core.binning.accumulate_state_populations_from_labeled(labeled_bin_pops, state_map,
                                                             state_pops,
                                                             check_state_map=True)
```

```
westpa.core.binning.assignments_list_to_table(nsegs, nbins, assignments)
```

Convert a list of bin assignments (integers) to a boolean table indicating if a given segment is in a given bin

`westpa.core.binning.coord_dtype`

alias of `float32`

`westpa.core.binning.index_dtype`

alias of `uint16`

class `westpa.core.binning.Bin(iterable=None, label=None)`

Bases: `set`

property `weight`

Total weight of all walkers in this bin

reweight(*new_weight*)

Reweight all walkers in this bin so that the total weight is *new_weight*

6.2.1.2 `westpa.core.binning.assign` module

Bin assignment for WEST simulations. This module defines “bin mappers” which take vectors of coordinates (or rather, coordinate tuples), and assign each a definite integer value identifying a bin. Critical portions are implemented in a Cython extension module.

A number of pre-defined bin mappers are available here:

- [`RectilinearBinMapper`](#), for bins divided by N-dimensional grids
- [`FuncBinMapper`](#), for functions which directly calculate bin assignments for a number of coordinate values. This is best used with C/Cython/Numba functions, or intelligently-tuned numpy-based Python functions.
- [`VectorizingFuncBinMapper`](#), for functions which calculate a bin assignment for a single coordinate value. This is best used for arbitrary Python functions.
- [`PiecewiseBinMapper`](#), for using a set of boolean-valued functions, one per bin, to determine assignments. This is likely to be much slower than a `FuncBinMapper` or `VectorizingFuncBinMapper` equipped with an appropriate function, and its use is discouraged.

One “super-mapper” is available, for assembling more complex bin spaces from simpler components:

- [`RecursiveBinMapper`](#), for nesting one set of bins within another.

Users are also free to implement their own mappers. A bin mapper must implement, at least, an `assign(coords, mask=None, output=None)` method, which is responsible for mapping each of the vector of coordinate tuples `coords` to an integer (`np.uint16`) indicating a what bin that coordinate tuple falls into. The optional `mask` (a numpy bool array) specifies that some coordinates are to be skipped; this is used, for instance, by the recursive (nested) bin mapper to minimize the number of calculations required to definitively assign a coordinate tuple to a bin. Similarly, the optional `output` must be an integer (`uint16`) array of the same length as `coords`, into which assignments are written. The `assign()` function must return a reference to `output`. (This is used to avoid allocating many temporary output arrays in complex binning scenarios.)

A user-defined bin mapper must also make an `nbins` property available, containing the total number of bins within the mapper.

class `westpa.core.binning.assign.Bin(iterable=None, label=None)`

Bases: `set`

property `weight`

Total weight of all walkers in this bin

reweight(*new_weight*)

Reweight all walkers in this bin so that the total weight is *new_weight*

`westpa.core.binning.assign.output_map(output, omap, mask)`

For each output for which mask is true, execute `output[i] = omap[output[i]]`

`westpa.core.binning.assign.apply_down(func, args, kwargs, coords, mask, output)`

Apply `func(coord, *args, **kwargs)` to each input coordinate tuple, skipping any for which mask is false and writing results to output.

`westpa.core.binning.assign.apply_down_argmin_across(func, args, kwargs, func_output_len, coords, mask, output)`

Apply `func(coord, *args, **kwargs)` to each input coordinate tuple, skipping any for which mask is false and writing results to output.

`westpa.core.binning.assign.rectilinear_assign(coords, mask, output, boundaries, boundlens)`

For bins delimited by sets boundaries on a rectilinear grid (`boundaries`), assign coordinates to bins, assuming C ordering of indices within the grid. `boundlens` is the number of boundaries in each dimension.

`westpa.core.binning.assign.index_dtype`

alias of `uint16`

`westpa.core.binning.assign.coord_dtype`

alias of `float32`

class `westpa.core.binning.assign.BinMapper`

Bases: `object`

hashfunc(**, usedforsecurity=True*)

Returns a sha256 hash object; optionally initialized with a string

construct_bins(*type_=<class 'westpa.core.binning.bins.Bin'>*)

Construct and return an array of bins of type `type`

pickle_and_hash()

Pickle this mapper and calculate a hash of the result (thus identifying the contents of the pickled data), returning a tuple (`pickled_data`, `hash`). This will raise `PickleError` if this mapper cannot be pickled, in which case code that would otherwise rely on detecting a topology change must assume a topology change happened, even if one did not.

class `westpa.core.binning.assign.NopMapper`

Bases: `BinMapper`

Put everything into one bin.

assign(*coords, mask=None, output=None*)

class `westpa.core.binning.assign.RectilinearBinMapper(boundaries)`

Bases: `BinMapper`

Bin into a rectangular grid based on tuples of float values

property boundaries

assign(*coords, mask=None, output=None*)

class `westpa.core.binning.assign.PiecewiseBinMapper(functions)`

Bases: `BinMapper`

Binning using a set of functions returning boolean values; if the Nth function returns `True` for a coordinate tuple, then that coordinate is in the Nth bin.

assign(*coords*, *mask=None*, *output=None*)

class westpa.core.binning.assign.**FuncBinMapper**(*func*, *nbins*, *args=None*, *kwargs=None*)

Bases: [BinMapper](#)

Binning using a custom function which must iterate over input coordinate sets itself.

assign(*coords*, *mask=None*, *output=None*)

class westpa.core.binning.assign.**VectorizingFuncBinMapper**(*func*, *nbins*, *args=None*, *kwargs=None*)

Bases: [BinMapper](#)

Binning using a custom function which is evaluated once for each (unmasked) coordinate tuple provided.

assign(*coords*, *mask=None*, *output=None*)

class westpa.core.binning.assign.**VoronoiBinMapper**(*dfunc*, *centers*, *dfargs=None*, *dfkwargs=None*)

Bases: [BinMapper](#)

A one-dimensional mapper which assigns a multidimensional pcoord to the closest center based on a distance metric. Both the list of centers and the distance function must be supplied.

assign(*coords*, *mask=None*, *output=None*)

class westpa.core.binning.assign.**RecursiveBinMapper**(*base_mapper*, *start_index=0*)

Bases: [BinMapper](#)

Nest mappers one within another.

property labels

property start_index

add_mapper(*mapper*, *replaces_bin_at*)

Replace the bin containing the coordinate tuple *replaces_bin_at* with the specified mapper.

assign(*coords*, *mask=None*, *output=None*)

6.2.1.3 westpa.core.binning.bins module

class westpa.core.binning.bins.**Bin**(*iterable=None*, *label=None*)

Bases: `set`

property weight

Total weight of all walkers in this bin

reweight(*new_weight*)

Reweight all walkers in this bin so that the total weight is *new_weight*

6.2.2 Minimal Adaptive Binning (MAB) Scheme

6.2.2.1 westpa.core.binning.mab module

class westpa.core.binning.mab.**FuncBinMapper**(*func, nbins, args=None, kwargs=None*)

Bases: [BinMapper](#)

Binning using a custom function which must iterate over input coordinate sets itself.

assign(*coords, mask=None, output=None*)

westpa.core.binning.mab.**expandvars**(*path*)

Expand shell variables of form \$var and \${var}. Unknown variables are left unchanged.

westpa.core.binning.mab.**map_mab**(*coords, mask, output, *args, **kwargs*)

Binning which adaptively places bins based on the positions of extrema segments and bottleneck segments, which are where the difference in probability is the greatest along the progress coordinate. Operates per dimension and places a fixed number of evenly spaced bins between the segments with the min and max pcoord values. Extrema and bottleneck segments are assigned their own bins.

class westpa.core.binning.mab.**MABBinMapper**(*nbins, direction=None, skip=None, bottleneck=True, pca=False, mab_log=False, bin_log=False*)

Bases: [FuncBinMapper](#)

Adaptively place bins in between minimum and maximum segments along the progress coordinte. Extrema and bottleneck segments are assigned to their own bins.

6.2.2.2 westpa.core.binning.mab_driver

class westpa.core.binning.mab_driver.**WEDriver**(*rc=None, system=None*)

Bases: `object`

A class implemented Huber & Kim's weighted ensemble algorithm over Segment objects. This class handles all binning, recycling, and preparation of new Segment objects for the next iteration. Binning is accomplished using `system.bin_mapper`, and per-bin target counts are from `system.bin_target_counts`.

The workflow is as follows:

- 1) Call `new_iteration()` every new iteration, providing any recycling targets that are in force and any available initial states for recycling.
- 2) Call `assign()` to assign segments to bins based on their initial and end points. This returns the number of walkers that were recycled.
- 3) Call `run_we()`, optionally providing a set of initial states that will be used to recycle walkers.

Note the presence of `flux_matrix`, `transition_matrix`, `current_iter_segments`, `next_iter_segments`, `recycling_segments`, `initial_binning`, `final_binning`, `next_iter_binning`, and `new_weights` (to be documented soon).

weight_split_threshold = 2.0

weight_merge_cutoff = 1.0

largest_allowed_weight = 1.0

smallest_allowed_weight = 1e-310

process_config()

property next_iter_segments

Newly-created segments for the next iteration

property current_iter_segments

Segments for the current iteration

property next_iter_assignments

Bin assignments (indices) for initial points of next iteration.

property current_iter_assignments

Bin assignments (indices) for endpoints of current iteration.

property recycling_segments

Segments designated for recycling

property n_recycled_segs

Number of segments recycled this iteration

property n_istates_needed

Number of initial states needed to support recycling for this iteration

check_threshold_configs()

Check to see if weight thresholds parameters are valid

clear()

Explicitly delete all Segment-related state.

new_iteration(*initial_states=None, target_states=None, new_weights=None, bin_mapper=None, bin_target_counts=None*)

Prepare for a new iteration. *initial_states* is a sequence of all *InitialState* objects valid for use in to generating new segments for the *next* iteration (after the one being begun with the call to *new_iteration*); that is, these are states available to recycle to. Target states which generate recycling events are specified in *target_states*, a sequence of *TargetState* objects. Both *initial_states* and *target_states* may be empty as required.

The optional *new_weights* is a sequence of *NewWeightEntry* objects which will be used to construct the initial flux matrix.

The given *bin_mapper* will be used for assignment, and *bin_target_counts* used for splitting/merging target counts; each will be obtained from the system object if omitted or *None*.

add_initial_states(*initial_states*)

Add newly-prepared initial states to the pool available for recycling.

property all_initial_states

Return an iterator over all initial states (available or used)

assign(*segments, initializing=False*)

Assign segments to initial and final bins, and update the (internal) lists of used and available initial states. If *initializing* is *True*, then the “final” bin assignments will be identical to the initial bin assignments, a condition required for seeding a new iteration from pre-existing segments.

populate_initial(*initial_states, weights, system=None*)

Create walkers for a new weighted ensemble simulation.

One segment is created for each provided initial state, then binned and split/merged as necessary. After this function is called, *next_iter_segments* will yield the new segments to create, *used_initial_states* will contain data about which of the provided initial states were used, and *avail_initial_states* will contain data about which initial states were unused (because their corresponding walkers were merged out of existence).

rebin_current(*parent_segments*)

Reconstruct walkers for the current iteration based on (presumably) new binning. The previous iteration's segments must be provided (as *parent_segments*) in order to update endpoint types appropriately.

construct_next()

Construct walkers for the next iteration, by running weighted ensemble recycling and bin/split/merge on the segments previously assigned to bins using **assign**. Enough unused initial states must be present in *self.avail_initial_states* for every recycled walker to be assigned an initial state.

After this function completes, *self.flux_matrix* contains a valid flux matrix for this iteration (including any contributions from recycling from the previous iteration), and *self.next_iter_segments* contains a list of segments ready for the next iteration, with appropriate values set for weight, endpoint type, parent walkers, and so on.

class westpa.core.binning.mab_driver.**MABDriver**(*rc=None, system=None*)

Bases: [WEDriver](#)

assign(*segments, initializing=False*)

Assign segments to initial and final bins, and update the (internal) lists of used and available initial states. This function is adapted to the MAB scheme, so that the initial and final segments are sent to the bin mapper at the same time, otherwise the initial and final bin boundaries can be inconsistent.

6.2.2.3 westpa.core.binning.mab_manager

class westpa.core.binning.mab_manager.**MABBinMapper**(*nbins, direction=None, skip=None, bottleneck=True, pca=False, mab_log=False, bin_log=False*)

Bases: [FuncBinMapper](#)

Adaptively place bins in between minimum and maximum segments along the progress coordinate. Extrema and bottleneck segments are assigned to their own bins.

class westpa.core.binning.mab_manager.**WESimManager**(*rc=None*)

Bases: `object`

process_config()

register_callback(*hook, function, priority=0*)

Registers a callback to execute during the given *hook* into the simulation loop. The optional *priority* is used to order when the function is called relative to other registered callbacks.

invoke_callbacks(*hook, *args, **kwargs*)

load_plugins(*plugins=None*)

report_bin_statistics(*bins, target_states, save_summary=False*)

get_bstate_pcoords(*basis_states, label='basis'*)

For each of the given *basis_states*, calculate progress coordinate values as necessary. The HDF5 file is not updated.

report_basis_states(*basis_states, label='basis'*)

report_target_states(*target_states*)

initialize_simulation(*basis_states*, *target_states*, *start_states*, *segs_per_state*=1, *suppress_we*=False)

Initialize a new weighted ensemble simulation, taking *segs_per_state* initial states from each of the given *basis_states*.

w_init is the forward-facing version of this function

prepare_iteration()

finalize_iteration()

Clean up after an iteration and prepare for the next.

get_istate_futures()

Add *n_states* initial states to the internal list of initial states assigned to recycled particles. Spare states are used if available, otherwise new states are created. If created new initial states requires generation, then a set of futures is returned representing work manager tasks corresponding to the necessary generation work.

propagate()

save_bin_data()

Calculate and write flux and transition count matrices to HDF5. Population and rate matrices are likely useless at the single-tau level and are no longer written.

check_propagation()

Check for failures in propagation or initial state generation, and raise an exception if any are found.

run_we()

Run the weighted ensemble algorithm based on the binning in *self.final_bins* and the recycled particles in *self.to_recycle*, creating and committing the next iteration's segments to storage as well.

prepare_new_iteration()

Commit data for the coming iteration to the HDF5 file.

run()

prepare_run()

Prepare a new run.

finalize_run()

Perform cleanup at the normal end of a run

pre_propagation()

post_propagation()

pre_we()

post_we()

westpa.core.binning.mab_manager.grouper(*n*, *iterable*, *fillvalue*=None)

Collect data into fixed-length chunks or blocks

```
class westpa.core.binning.mab_manager.InitialState(state_id, basis_state_id, iter_created,
                                                iter_used=None, istate_type=None,
                                                istate_status=None, pcoord=None,
                                                basis_state=None, basis_auxref=None)
```

Bases: object

Describes an initial state for a new trajectory. These are generally constructed by appropriate modification of a basis state.

Variables

- **state_id** – Integer identifier of this state, usually set by the data manager.
- **basis_state_id** – Identifier of the basis state from which this state was generated, or None.
- **basis_state** – The *BasisState* from which this state was generated, or None.
- **iter_created** – Iteration in which this state was generated (0 for simulation initialization).
- **iter_used** – Iteration in which this state was used to initiate a trajectory (None for unused).
- **istate_type** – Integer describing the type of this initial state (ISTATE_TYPE_BASIS for direct use of a basis state, ISTATE_TYPE_GENERATED for a state generated from a basis state, ISTATE_TYPE_RESTART for a state corresponding to the endpoint of a segment in another simulation, or ISTATE_TYPE_START for a state generated from a start state).
- **istate_status** – Integer describing whether this initial state has been properly prepared.
- **pcoord** – The representative progress coordinate of this state.

```
ISTATE_TYPE_UNSET = 0
```

```
ISTATE_TYPE_BASIS = 1
```

```
ISTATE_TYPE_GENERATED = 2
```

```
ISTATE_TYPE_RESTART = 3
```

```
ISTATE_TYPE_START = 4
```

```
ISTATE_UNUSED = 0
```

```
ISTATE_STATUS_PENDING = 0
```

```
ISTATE_STATUS_PREPARED = 1
```

```
ISTATE_STATUS_FAILED = 2
```

```
istate_types = {'ISTATE_TYPE_BASIS': 1, 'ISTATE_TYPE_GENERATED': 2,  
'ISTATE_TYPE_RESTART': 3, 'ISTATE_TYPE_START': 4, 'ISTATE_TYPE_UNSET': 0}
```

```
istate_type_names = {0: 'ISTATE_TYPE_UNSET', 1: 'ISTATE_TYPE_BASIS', 2:  
'ISTATE_TYPE_GENERATED', 3: 'ISTATE_TYPE_RESTART', 4: 'ISTATE_TYPE_START'}
```

```
istate_statuses = {'ISTATE_STATUS_FAILED': 2, 'ISTATE_STATUS_PENDING': 0,  
'ISTATE_STATUS_PREPARED': 1}
```

```
istate_status_names = {0: 'ISTATE_STATUS_PENDING', 1: 'ISTATE_STATUS_PREPARED', 2:  
'ISTATE_STATUS_FAILED'}
```

```
as_numpy_record()
```

```
westpa.core.binning.mab_manager.pare_basis_initial_states(basis_states, initial_states,  
                                                           segments=None)
```

Given iterables of basis and initial states (and optionally segments that use them), return minimal sets (as in `__builtins__.set`) of states needed to describe the history of the given segments and initial states.


```
class westpa.core.binning.mab_manager.Segment(n_iter=None, seg_id=None, weight=None,  
                                             endpoint_type=None, parent_id=None,  
                                             wtg_parent_ids=None, pcoord=None, status=None,  
                                             walltime=None, cputime=None, data=None)
```

Bases: object

A class wrapping segment data that must be passed through the work manager or data manager. Most fields are self-explanatory. One item worth noting is that a negative parent ID means that the segment starts from the initial state with ID $-(\text{segment.parent_id}+1)$

```
SEG_STATUS_UNSET = 0
```

```
SEG_STATUS_PREPARED = 1
```

```
SEG_STATUS_COMPLETE = 2
```

```
SEG_STATUS_FAILED = 3
```

```
SEG_INITPOINT_UNSET = 0
```

```
SEG_INITPOINT_CONTINUES = 1
```

```
SEG_INITPOINT_NEWTRAJ = 2
```

```
SEG_ENDPOINT_UNSET = 0
```

```
SEG_ENDPOINT_CONTINUES = 1
```

```
SEG_ENDPOINT_MERGED = 2
```

```
SEG_ENDPOINT_RECYCLED = 3
```

```
statuses = {'SEG_STATUS_COMPLETE': 2, 'SEG_STATUS_FAILED': 3, 'SEG_STATUS_PREPARED':  
1, 'SEG_STATUS_UNSET': 0}
```

```
initpoint_types = {'SEG_INITPOINT_CONTINUES': 1, 'SEG_INITPOINT_NEWTRAJ': 2,  
'SEG_INITPOINT_UNSET': 0}
```

```
endpoint_types = {'SEG_ENDPOINT_CONTINUES': 1, 'SEG_ENDPOINT_MERGED': 2,  
'SEG_ENDPOINT_RECYCLED': 3, 'SEG_ENDPOINT_UNSET': 0}
```

```
status_names = {0: 'SEG_STATUS_UNSET', 1: 'SEG_STATUS_PREPARED', 2:  
'SEG_STATUS_COMPLETE', 3: 'SEG_STATUS_FAILED'}
```

```
initpoint_type_names = {0: 'SEG_INITPOINT_UNSET', 1: 'SEG_INITPOINT_CONTINUES', 2:  
'SEG_INITPOINT_NEWTRAJ'}
```

```
endpoint_type_names = {0: 'SEG_ENDPOINT_UNSET', 1: 'SEG_ENDPOINT_CONTINUES', 2:  
'SEG_ENDPOINT_MERGED', 3: 'SEG_ENDPOINT_RECYCLED'}
```

```
static initial_pcoord(segment)
```

Return the initial progress coordinate point of this segment.

```
static final_pcoord(segment)
```

Return the final progress coordinate point of this segment.

```
property initpoint_type
```

property `initial_state_id`

property `status_text`

property `endpoint_type_text`

class `westpa.core.binning.mab_manager.MABSimManager`(*rc=None*)

Bases: [`WESimManager`](#)

initialize_simulation(*basis_states, target_states, start_states, segs_per_state=1, suppress_we=False*)

Initialize a new weighted ensemble simulation, taking `segs_per_state` initial states from each of the given `basis_states`.

`w_init` is the forward-facing version of this function

propagate()

prepare_iteration()

6.2.3 westpa.core.kinetics package

6.2.3.1 westpa.core.kinetics module

Kinetics analysis library

class `westpa.core.kinetics.RateAverager`(*bin_mapper, system=None, data_manager=None, work_manager=None*)

Bases: `object`

Calculate bin-to-bin kinetic properties (fluxes, rates, populations) at 1-tau resolution

extract_data(*iter_indices*)

Extract data from the `data_manger` and place in dict mirroring the same underlying layout.

task_generator(*iter_start, iter_stop, block_size*)

calculate(*iter_start=None, iter_stop=None, n_blocks=1, queue_size=1*)

Read the HDF5 file and collect flux matrices and population vectors for each bin for each iteration in the range [`iter_start`, `iter_stop`). Break the calculation into `n_blocks` blocks. If the calculation is broken up into more than one block, `queue_size` specifies the maximum number of tasks in the work queue.

`westpa.core.kinetics.calculate_labeled_fluxes`(*nstates, weights, parent_ids, micro_assignments, traj_assignments, fluxes*)

`westpa.core.kinetics.labeled_flux_to_rate`(*labeled_fluxes, labeled_pops, output=None*)

Convert a labeled flux matrix and corresponding labeled bin populations to a labeled rate matrix.

`westpa.core.kinetics.calculate_labeled_fluxes_alllags`(*nstates, weights, parent_ids, micro_assignments, traj_assignments, fluxes*)

`westpa.core.kinetics.nested_to_flat_matrix`(*input*)

Convert nested flux/rate matrix into a flat supermatrix.

`westpa.core.kinetics.nested_to_flat_vector`(*input*)

Convert nested labeled population vector into a flat vector.

`westpa.core.kinetics.flat_to_nested_matrix(nstates, nbins, input)`

Convert flat supermatrix into nested matrix.

`westpa.core.kinetics.flat_to_nested_vector(nstates, nbins, input)`

Convert flat “supervector” into nested vector.

`westpa.core.kinetics.find_macrostate_transitions(nstates, weights, label_assignments, state_assignments, dt, state, macro_fluxes, macro_counts, target_fluxes, target_counts, durations)`

`westpa.core.kinetics.sequence_macro_flux_to_rate(dataset, pops, istate, jstate, pairwise=True, stride=None)`

Convert a sequence of macrostate fluxes and corresponding list of trajectory ensemble populations to a sequence of rate matrices.

If the optional `pairwise` is true (the default), then rates are normalized according to the relative probability of the initial state among the pair of states (initial, final); this is probably what you want, as these rates will then depend only on the definitions of the states involved (and never the remaining states). Otherwise (“`pairwise`” is false), the rates are normalized according the probability of the initial state among *all* other states.

class `westpa.core.kinetics.WKinetics`

Bases: object

`w_kinetics()`

6.2.3.2 `westpa.core.kinetics.events` module

`westpa.core.kinetics.events.weight_dtype`

alias of float64

`westpa.core.kinetics.events.index_dtype`

alias of uint16

`westpa.core.kinetics.events.find_macrostate_transitions(nstates, weights, label_assignments, state_assignments, dt, state, macro_fluxes, macro_counts, target_fluxes, target_counts, durations)`

class `westpa.core.kinetics.events.WKinetics`

Bases: object

`w_kinetics()`

6.2.3.3 `westpa.core.kinetics.matrates` module

Routines for implementing Letteri et al.’s macrostate-to-macrostate rate calculations using extrapolation to steady-state populations from average rate matrices

Internally, “labeled” objects (bin populations labeled by history, rate matrix elements labeled by history) are stored as nested arrays – e.g. `rates[initial_label, final_label, initial_bin, final_bin]`. These are converted to the flat forms required for, say, eigenvalue calculations internally, and the results converted back. This is because these conversions are not expensive, and saves users of this code from having to know how the flattened indexing works (something I screwed up all too easily during development) – mcz

`westpa.core.kinetics.matrates.weight_dtype`

alias of float64

`westpa.core.kinetics.matrates.calculate_labeled_fluxes`(*nstates*, *weights*, *parent_ids*,
micro_assignments, *traj_assignments*,
fluxes)

`westpa.core.kinetics.matrates.calculate_labeled_fluxes_alllags`(*nstates*, *weights*, *parent_ids*,
micro_assignments,
traj_assignments, *fluxes*)

`westpa.core.kinetics.matrates.labeled_flux_to_rate`(*labeled_fluxes*, *labeled_pops*, *output=None*)

Convert a labeled flux matrix and corresponding labeled bin populations to a labeled rate matrix.

`westpa.core.kinetics.matrates.nested_to_flat_matrix`(*input*)

Convert nested flux/rate matrix into a flat supermatrix.

`westpa.core.kinetics.matrates.nested_to_flat_vector`(*input*)

Convert nested labeled population vector into a flat vector.

`westpa.core.kinetics.matrates.flat_to_nested_vector`(*nstates*, *nbins*, *input*)

Convert flat “supervector” into nested vector.

exception `westpa.core.kinetics.matrates.ConsistencyWarning`

Bases: `UserWarning`

`westpa.core.kinetics.matrates.get_steady_state`(*rates*)

Get steady state solution for a rate matrix. As an optimization, returns the flattened labeled population vector (of length *nstates***nbins*); to convert to the nested vector used for storage, use `nested_to_flat_vector()`.

`westpa.core.kinetics.matrates.get_macrostate_rates`(*labeled_rates*, *labeled_pops*, *extrapolate=True*)

Using a labeled rate matrix and labeled bin populations, calculate the steady state probability distribution and consequent state-to-state rates.

Returns (*ss*, *macro_rates*), where *ss* is the steady-state probability distribution and *macro_rates* is the state-to-state rate matrix.

`westpa.core.kinetics.matrates.estimate_rates`(*nbins*, *state_labels*, *weights*, *parent_ids*, *bin_assignments*,
label_assignments, *state_map*, *labeled_pops*,
all_lags=False, *labeled_fluxes=None*,
labeled_rates=None, *unlabeled_rates=None*)

Estimate fluxes and rates over multiple iterations. The number of iterations is determined by how many vectors of weights, parent IDs, bin assignments, and label assignments are passed.

If *all_lags* is true, then the average is over all possible lags within the length-*N* window given, otherwise simply the length *N* lag.

Returns labeled flux matrix, labeled rate matrix, and unlabeled rate matrix.

6.2.3.4 westpa.core.kinetics.rate_averaging module

`westpa.core.kinetics.rate_averaging.namedtuple`(*typename*, *field_names*, *, *rename=False*, *defaults=None*, *module=None*)

Returns a new subclass of tuple with named fields.

```
>>> Point = namedtuple('Point', ['x', 'y'])
>>> Point.__doc__           # docstring for the new class
'Point(x, y)'
>>> p = Point(11, y=22)     # instantiate with positional args or keywords
>>> p[0] + p[1]             # indexable like a plain tuple
33
>>> x, y = p                # unpack like a regular tuple
>>> x, y
(11, 22)
>>> p.x + p.y               # fields also accessible by name
33
>>> d = p._asdict()         # convert to a dictionary
>>> d['x']
11
>>> Point(**d)              # convert from a dictionary
Point(x=11, y=22)
>>> p._replace(x=100)       # _replace() is like str.replace() but targets_
↪ named fields
Point(x=100, y=22)
```

class `westpa.core.kinetics.rate_averaging.zip_longest`

Bases: object

`zip_longest(iter1 [,iter2 [...]], [fillvalue=None])` → zip_longest object

Return a zip_longest object whose `__next__()` method returns a tuple where the i-th element comes from the i-th iterable argument. The `__next__()` method continues until the longest iterable in the argument sequence is exhausted and then it raises `StopIteration`. When the shorter iterables are exhausted, the fillvalue is substituted in their place. The fillvalue defaults to None or can be specified by a keyword argument.

`westpa.core.kinetics.rate_averaging.flux_assign`(*weights*, *init_assignments*, *final_assignments*, *flux_matrix*)

`westpa.core.kinetics.rate_averaging.pop_assign`(*weights*, *assignments*, *populations*)

`westpa.core.kinetics.rate_averaging.calc_rates`(*fluxes*, *populations*, *rates*, *mask*)

Calculate a rate matrices from flux and population matrices. A matrix of the same shape as fluxes, is also produced, to be used for generating a mask for the rate matrices where initial state populations are zero.

class `westpa.core.kinetics.rate_averaging.StreamingStats1D`

Bases: object

Calculate mean and variance of a series of one-dimensional arrays of shape (nbins,) using an online algorithm. The statistics are accumulated along what would be axis=0 if the input arrays were stacked vertically.

This code has been adapted from: http://www.johndcook.com/skewness_kurtosis.html

M1

M2

mean

n

update(*x*, *mask*)

Update the running set of statistics given

Parameters

- **x** (*1d ndarray*) – values from a single observation
- **mask** (*1d ndarray*) – A uint8 array to exclude entries from the accumulated statistics.

var

class westpa.core.kinetics.rate_averaging.**StreamingStats2D**

Bases: object

Calculate mean and variance of a series of two-dimensional arrays of shape (nbins, nbins) using an online algorithm. The statistics are accumulated along what would be axis=0 if the input arrays were stacked vertically.

This code has been adapted from: http://www.johndcook.com/skewness_kurtosis.html

M1

M2

mean

n

update(*x*, *mask*)

Update the running set of statistics given

Parameters

- **x** (*2d ndarray*) – values from a single observation
- **mask** (*2d ndarray*) – A uint8 array to exclude entries from the accumulated statistics.

var

class westpa.core.kinetics.rate_averaging.**StreamingStatsTuple**(*M1*, *M2*, *n*)

Bases: tuple

Create new instance of StreamingStatsTuple(*M1*, *M2*, *n*)

M1

Alias for field number 0

M2

Alias for field number 1

n

Alias for field number 2

westpa.core.kinetics.rate_averaging.**grouper**(*n*, *iterable*, *fillvalue=None*)

Collect data into fixed-length chunks or blocks

westpa.core.kinetics.rate_averaging.**tuple2stats**(*stat_tuple*)

`westpa.core.kinetics.rate_averaging.process_iter_chunk(bin_mapper, iter_indices, iter_data=None)`

Calculate the flux matrices and populations of a set of iterations specified by `iter_indices`. Optionally provide the necessary arrays to perform the calculation in `iter_data`. Otherwise get data from the `data_manager` directly.

class `westpa.core.kinetics.rate_averaging.RateAverager(bin_mapper, system=None, data_manager=None, work_manager=None)`

Bases: `object`

Calculate bin-to-bin kinetic properties (fluxes, rates, populations) at 1-tau resolution

extract_data(`iter_indices`)

Extract data from the `data_manger` and place in dict mirroring the same underlying layout.

task_generator(`iter_start, iter_stop, block_size`)

calculate(`iter_start=None, iter_stop=None, n_blocks=1, queue_size=1`)

Read the HDF5 file and collect flux matrices and population vectors for each bin for each iteration in the range [`iter_start`, `iter_stop`). Break the calculation into `n_blocks` blocks. If the calculation is broken up into more than one block, `queue_size` specifies the maximum number of tasks in the work queue.

6.2.4 westpa.core.propagators package

6.2.4.1 westpa.core.propagators module

`westpa.core.propagators.blocked_iter(blocksize, iterable, fillvalue=None)`

class `westpa.core.propagators.WESTPropagator(rc=None)`

Bases: `object`

prepare_iteration(`n_iter, segments`)

Perform any necessary per-iteration preparation. This is run by the work manager.

finalize_iteration(`n_iter, segments`)

Perform any necessary post-iteration cleanup. This is run by the work manager.

get_pcoord(`state`)

Get the progress coordinate of the given basis or initial state.

gen_istate(`basis_state, initial_state`)

Generate a new initial state from the given basis state.

propagate(`segments`)

Propagate one or more segments, including any necessary per-iteration setup and teardown for this propagator.

clear_basis_initial_states()

update_basis_initial_states(`basis_states, initial_states`)

6.2.4.2 westpa.core.propagators.executable module

class westpa.core.propagators.executable.**BytesIO**(*initial_bytes=b''*)

Bases: `_BufferedIOBase`

Buffered I/O implementation using an in-memory bytes buffer.

close()

Disable all I/O operations.

closed

True if the file is closed.

flush()

Does nothing.

getbuffer()

Get a read-write view over the contents of the BytesIO object.

getvalue()

Retrieve the entire contents of the BytesIO object.

isatty()

Always returns False.

BytesIO objects are not connected to a TTY-like device.

read(*size=-1, /*)

Read at most size bytes, returned as a bytes object.

If the size argument is negative, read until EOF is reached. Return an empty bytes object at EOF.

read1(*size=-1, /*)

Read at most size bytes, returned as a bytes object.

If the size argument is negative or omitted, read until EOF is reached. Return an empty bytes object at EOF.

readable()

Returns True if the IO object can be read.

readinto(*buffer, /*)

Read bytes into buffer.

Returns number of bytes read (0 for EOF), or None if the object is set not to block and has no data to read.

readline(*size=-1, /*)

Next line from the file, as a bytes object.

Retain newline. A non-negative size argument limits the maximum number of bytes to return (an incomplete line may be returned then). Return an empty bytes object at EOF.

readlines(*size=None, /*)

List of bytes objects, each a line from the file.

Call `readline()` repeatedly and return a list of the lines so read. The optional size argument, if given, is an approximate bound on the total number of bytes in the lines returned.

seek(*pos*, *whence*=0, /)

Change stream position.

Seek to byte offset pos relative to position indicated by whence:

0 Start of stream (the default). pos should be ≥ 0 ; 1 Current position - pos may be negative; 2 End of stream - pos usually negative.

Returns the new absolute position.

seekable()

Returns True if the IO object can be seeked.

tell()

Current file position, an integer.

truncate(*size*=None, /)

Truncate the file to at most size bytes.

Size defaults to the current file position, as returned by tell(). The current file position is unchanged. Returns the new size.

writable()

Returns True if the IO object can be written.

write(*b*, /)

Write bytes to file.

Return the number of bytes written.

writelines(*lines*, /)

Write lines to the file.

Note that newlines are not added. lines can be any iterable object producing bytes-like objects. This is equivalent to calling write() for each element.

westpa.core.propagators.executable.get_object(*object_name*, *path*=None)

Attempt to load the given object, using additional path information if given.

class westpa.core.propagators.executable.WESTPropagator(*rc*=None)

Bases: object

prepare_iteration(*n_iter*, *segments*)

Perform any necessary per-iteration preparation. This is run by the work manager.

finalize_iteration(*n_iter*, *segments*)

Perform any necessary post-iteration cleanup. This is run by the work manager.

get_pcoord(*state*)

Get the progress coordinate of the given basis or initial state.

gen_istate(*basis_state*, *initial_state*)

Generate a new initial state from the given basis state.

propagate(*segments*)

Propagate one or more segments, including any necessary per-iteration setup and teardown for this propagator.

clear_basis_initial_states()

update_basis_initial_states(*basis_states*, *initial_states*)

```
class westpa.core.propagators.executable.BasisState(label, probability, pcoord=None, auxref=None,
                                                    state_id=None)
```

Bases: object

Describes an basis (micro)state. These basis states are used to generate initial states for new trajectories, either at the beginning of the simulation (i.e. at `w_init`) or due to recycling.

Variables

- **state_id** – Integer identifier of this state, usually set by the data manager.
- **label** – A descriptive label for this microstate (may be empty)
- **probability** – Probability of this state to be selected when creating a new trajectory.
- **pcoord** – The representative progress coordinate of this state.
- **auxref** – A user-provided (string) reference for locating data associated with this state (usually a filesystem path).

```
classmethod states_to_file(states, fileobj)
```

Write a file defining basis states, which may then be read by `states_from_file()`.

```
classmethod states_from_file(statefile)
```

Read a file defining basis states. Each line defines a state, and contains a label, the probability, and optionally a data reference, separated by whitespace, as in:

```
unbound      1.0
```

or:

```
unbound_0    0.6      state0.pdb
unbound_1    0.4      state1.pdb
```

```
as_numpy_record()
```

Return the data for this state as a numpy record array.

```
class westpa.core.propagators.executable.InitialState(state_id, basis_state_id, iter_created,
                                                       iter_used=None, istate_type=None,
                                                       istate_status=None, pcoord=None,
                                                       basis_state=None, basis_auxref=None)
```

Bases: object

Describes an initial state for a new trajectory. These are generally constructed by appropriate modification of a basis state.

Variables

- **state_id** – Integer identifier of this state, usually set by the data manager.
- **basis_state_id** – Identifier of the basis state from which this state was generated, or `None`.
- **basis_state** – The *BasisState* from which this state was generated, or `None`.
- **iter_created** – Iteration in which this state was generated (0 for simulation initialization).
- **iter_used** – Iteration in which this state was used to initiate a trajectory (`None` for unused).
- **istate_type** – Integer describing the type of this initial state (`ISTATE_TYPE_BASIS` for direct use of a basis state, `ISTATE_TYPE_GENERATED` for a state generated from

a basis state, `ISTATE_TYPE_RESTART` for a state corresponding to the endpoint of a segment in another simulation, or `ISTATE_TYPE_START` for a state generated from a start state).

- **istate_status** – Integer describing whether this initial state has been properly prepared.
- **pcoord** – The representative progress coordinate of this state.

`ISTATE_TYPE_UNSET = 0`

`ISTATE_TYPE_BASIS = 1`

`ISTATE_TYPE_GENERATED = 2`

`ISTATE_TYPE_RESTART = 3`

`ISTATE_TYPE_START = 4`

`ISTATE_UNUSED = 0`

`ISTATE_STATUS_PENDING = 0`

`ISTATE_STATUS_PREPARED = 1`

`ISTATE_STATUS_FAILED = 2`

`istate_types = {'ISTATE_TYPE_BASIS': 1, 'ISTATE_TYPE_GENERATED': 2,
'ISTATE_TYPE_RESTART': 3, 'ISTATE_TYPE_START': 4, 'ISTATE_TYPE_UNSET': 0}`

`istate_type_names = {0: 'ISTATE_TYPE_UNSET', 1: 'ISTATE_TYPE_BASIS', 2:
'ISTATE_TYPE_GENERATED', 3: 'ISTATE_TYPE_RESTART', 4: 'ISTATE_TYPE_START'}`

`istate_statuses = {'ISTATE_STATUS_FAILED': 2, 'ISTATE_STATUS_PENDING': 0,
'ISTATE_STATUS_PREPARED': 1}`

`istate_status_names = {0: 'ISTATE_STATUS_PENDING', 1: 'ISTATE_STATUS_PREPARED', 2:
'ISTATE_STATUS_FAILED'}`

`as_numpy_record()`

`westpa.core.propagators.executable.return_state_type(state_obj)`

Convenience function for returning the state ID and type of the `state_obj` pointer

`class westpa.core.propagators.executable.Segment(n_iter=None, seg_id=None, weight=None,
endpoint_type=None, parent_id=None,
wtg_parent_ids=None, pcoord=None, status=None,
walltime=None, cputime=None, data=None)`

Bases: `object`

A class wrapping segment data that must be passed through the work manager or data manager. Most fields are self-explanatory. One item worth noting is that a negative parent ID means that the segment starts from the initial state with ID `-(segment.parent_id+1)`

`SEG_STATUS_UNSET = 0`

`SEG_STATUS_PREPARED = 1`

`SEG_STATUS_COMPLETE = 2`

```
SEG_STATUS_FAILED = 3

SEG_INITPOINT_UNSET = 0

SEG_INITPOINT_CONTINUES = 1

SEG_INITPOINT_NEWTRAJ = 2

SEG_ENDPOINT_UNSET = 0

SEG_ENDPOINT_CONTINUES = 1

SEG_ENDPOINT_MERGED = 2

SEG_ENDPOINT_RECYCLED = 3

statuses = {'SEG_STATUS_COMPLETE': 2, 'SEG_STATUS_FAILED': 3, 'SEG_STATUS_PREPARED':
1, 'SEG_STATUS_UNSET': 0}

initpoint_types = {'SEG_INITPOINT_CONTINUES': 1, 'SEG_INITPOINT_NEWTRAJ': 2,
'SEG_INITPOINT_UNSET': 0}

endpoint_types = {'SEG_ENDPOINT_CONTINUES': 1, 'SEG_ENDPOINT_MERGED': 2,
'SEG_ENDPOINT_RECYCLED': 3, 'SEG_ENDPOINT_UNSET': 0}

status_names = {0: 'SEG_STATUS_UNSET', 1: 'SEG_STATUS_PREPARED', 2:
'SEG_STATUS_COMPLETE', 3: 'SEG_STATUS_FAILED'}

initpoint_type_names = {0: 'SEG_INITPOINT_UNSET', 1: 'SEG_INITPOINT_CONTINUES', 2:
'SEG_INITPOINT_NEWTRAJ'}

endpoint_type_names = {0: 'SEG_ENDPOINT_UNSET', 1: 'SEG_ENDPOINT_CONTINUES', 2:
'SEG_ENDPOINT_MERGED', 3: 'SEG_ENDPOINT_RECYCLED'}

static initial_pcoord(segment)
    Return the initial progress coordinate point of this segment.

static final_pcoord(segment)
    Return the final progress coordinate point of this segment.

property initpoint_type

property initial_state_id

property status_text

property endpoint_type_text
```

```
westpa.core.propagators.executable.check_bool(value, action='warn')
```

Check that the given value is boolean in type. If not, either raise a warning (if *action*=='warn') or an exception (*action*=='raise').

```
westpa.core.propagators.executable.load_trajectory(folder)
```

Load trajectory from *folder* using mdtraj and return a mdtraj.Trajectory object. The folder should contain a trajectory and a topology file (with a recognizable extension) that is supported by mdtraj. The topology file is optional if the trajectory file contains topology data (e.g., HDF5 format).

```
westpa.core.propagators.executable.safe_extract(tar, path='.', members=None, *,
                                              numeric_owner=False)
```

```
westpa.core.propagators.executable.pcoord_loader(fieldname, pcoord_return_filename, destobj,
                                                  single_point)
```

Read progress coordinate data into the `pcoord` field on `destobj`. An exception will be raised if the data is malformed. If `single_point` is true, then only one (N-dimensional) point will be read, otherwise `system.pcoord_len` points will be read.

```
westpa.core.propagators.executable.aux_data_loader(fieldname, data_filename, segment, single_point)
```

```
westpa.core.propagators.executable.npy_data_loader(fieldname, coord_file, segment, single_point)
```

```
westpa.core.propagators.executable.pickle_data_loader(fieldname, coord_file, segment, single_point)
```

```
westpa.core.propagators.executable.trajectory_loader(fieldname, coord_folder, segment,
                                                      single_point)
```

Load data from the trajectory return. `coord_folder` should be the path to a folder containing trajectory files. `segment` is the `Segment` object that the data is associated with. Please see `load_trajectory` for more details. `single_point` is not used by this loader.

```
westpa.core.propagators.executable.restart_loader(fieldname, restart_folder, segment, single_point)
```

Load data from the restart return. The loader will tar all files in `restart_folder` and store it in the per-iteration HDF5 file. `segment` is the `Segment` object that the data is associated with. `single_point` is not used by this loader.

```
westpa.core.propagators.executable.restart_writer(path, segment)
```

Prepare the necessary files from the per-iteration HDF5 file to run `segment`.

```
westpa.core.propagators.executable.seglog_loader(fieldname, log_file, segment, single_point)
```

Load data from the log return. The loader will tar all files in `log_file` and store it in the per-iteration HDF5 file. `segment` is the `Segment` object that the data is associated with. `single_point` is not used by this loader.

```
class westpa.core.propagators.executable.ExecutablePropagator(rc=None)
```

Bases: [WESTPropagator](#)

```
ENV_CURRENT_ITER = 'WEST_CURRENT_ITER'
```

```
ENV_CURRENT_SEG_ID = 'WEST_CURRENT_SEG_ID'
```

```
ENV_CURRENT_SEG_DATA_REF = 'WEST_CURRENT_SEG_DATA_REF'
```

```
ENV_CURRENT_SEG_INITPOINT = 'WEST_CURRENT_SEG_INITPOINT_TYPE'
```

```
ENV_PARENT_SEG_ID = 'WEST_PARENT_ID'
```

```
ENV_PARENT_DATA_REF = 'WEST_PARENT_DATA_REF'
```

```
ENV_BSTATE_ID = 'WEST_BSTATE_ID'
```

```
ENV_BSTATE_DATA_REF = 'WEST_BSTATE_DATA_REF'
```

```
ENV_ISTATE_ID = 'WEST_ISTATE_ID'
```

```
ENV_ISTATE_DATA_REF = 'WEST_ISTATE_DATA_REF'
```

```
ENV_STRUCT_DATA_REF = 'WEST_STRUCT_DATA_REF'
```

```
ENV_RAND16 = 'WEST_RAND16'
```

```
ENV_RAND32 = 'WEST_RAND32'
```

```
ENV_RANDOM64 = 'WEST_RANDOM64'
```

```
ENV_RANDOM128 = 'WEST_RANDOM128'
```

```
ENV_RANDOMFLOAT = 'WEST_RANDOMFLOAT'
```

```
static makepath(template, template_args=None, expanduser=True, expandvars=True, abspath=False,
                realpath=False)
```

```
random_val_env_vars()
```

Return a set of environment variables containing random seeds. These are returned as a dictionary, suitable for use in `os.environ.update()` or as the `env` argument to `subprocess.Popen()`. Every child process executed by `exec_child()` gets these.

```
exec_child(executable, environ=None, stdin=None, stdout=None, stderr=None, cwd=None)
```

Execute a child process with the environment set from the current environment, the values of `self.addtl_child_environs`, the random numbers returned by `self.random_val_env_vars`, and the given `environ` (applied in that order). `stdin/stdout/stderr` are optionally redirected.

This function waits on the child process to finish, then returns `(rc, rusage)`, where `rc` is the child's return code and `rusage` is the resource usage tuple from `os.wait4()`

```
exec_child_from_child_info(child_info, template_args, environ)
```

```
update_args_env_basis_state(template_args, environ, basis_state)
```

```
update_args_env_initial_state(template_args, environ, initial_state)
```

```
update_args_env_iter(template_args, environ, n_iter)
```

```
update_args_env_segment(template_args, environ, segment)
```

```
template_args_for_segment(segment)
```

```
exec_for_segment(child_info, segment, addtl_env=None)
```

Execute a child process with environment and template expansion from the given segment.

```
exec_for_iteration(child_info, n_iter, addtl_env=None)
```

Execute a child process with environment and template expansion from the given iteration number.

```
exec_for_basis_state(child_info, basis_state, addtl_env=None)
```

Execute a child process with environment and template expansion from the given basis state

```
exec_for_initial_state(child_info, initial_state, addtl_env=None)
```

Execute a child process with environment and template expansion from the given initial state.

```
prepare_file_system(segment, environ)
```

```
setup_dataset_return(segment=None, subset_keys=None)
```

Set up temporary files and environment variables that point to them for segment runners to return data. `segment` is the `Segment` object that the return data is associated with. `subset_keys` specifies the names of a subset of data to be returned.

```
retrieve_dataset_return(state, return_files, del_return_files, single_point)
```

Retrieve returned data from the temporary locations directed by the environment variables. `state` is a `Segment`, `BasisState`, or `InitialState` object that the return data is associated with. `return_files` is a dict where the keys are the dataset names and the values are the paths to the temporarily files that contain the returned data. `del_return_files` is a dict where the keys are the names of datasets to be deleted (if the corresponding value is set to `True`) once the data is retrieved.

get_pcoord(*state*)

Get the progress coordinate of the given basis or initial state.

gen_istate(*basis_state*, *initial_state*)

Generate a new initial state from the given basis state.

prepare_iteration(*n_iter*, *segments*)

Perform any necessary per-iteration preparation. This is run by the work manager.

finalize_iteration(*n_iter*, *segments*)

Perform any necessary post-iteration cleanup. This is run by the work manager.

propagate(*segments*)

Propagate one or more segments, including any necessary per-iteration setup and teardown for this propagator.

6.2.5 westpa.core.reweight package

6.2.5.1 westpa.core.reweight module

Function(s) for the postanalysis toolkit

westpa.core.reweight.stats_process(*bin_assignments*, *weights*, *fluxes*, *populations*, *trans*, *mask*,
interval='timepoint')

westpa.core.reweight.reweight_for_c(*rows*, *cols*, *obs*, *flux*, *insert*, *indices*, *nstates*, *nbins*, *state_labels*,
state_map, *nfbins*, *istate*, *jstate*, *stride*, *bin_last_state_map*,
bin_state_map, *return_obs*, *obs_threshold*=1)

class westpa.core.reweight.FluxMatrix

Bases: object

w_postanalysis_matrix()

6.2.5.2 westpa.core.reweight.matrix module

westpa.core.reweight.matrix.weight_dtype

alias of float64

westpa.core.reweight.matrix.index_dtype

alias of uint16

westpa.core.reweight.matrix.stats_process(*bin_assignments*, *weights*, *fluxes*, *populations*, *trans*, *mask*,
interval='timepoint')

westpa.core.reweight.matrix.calc_stats(*bin_assignments*, *weights*, *fluxes*, *populations*, *trans*, *mask*,
sampling_frequency)

class westpa.core.reweight.matrix.FluxMatrix

Bases: object

w_postanalysis_matrix()

6.2.6 westpa.core modules

6.2.6.1 westpa.core module

6.2.6.2 westpa.core.data_manager module

HDF5 data manager for WEST.

Original HDF5 implementation: Joseph W. Kaus Current implementation: Matthew C. Zwier

WEST exclusively uses the cross-platform, self-describing file format HDF5 for data storage. This ensures that data is stored efficiently and portably in a manner that is relatively straightforward for other analysis tools (perhaps written in C/C++/Fortran) to access.

The data is laid out in HDF5 as follows:

- summary – overall summary data for the simulation
- /iterations/ – data for individual iterations, one group per iteration under /iterations
 - iter_00000001/ – data for iteration 1
 - * seg_index – overall information about segments in the iteration, including weight
 - * pcoord – progress coordinate data organized as [seg_id][time][dimension]
 - * wtg_parents – data used to reconstruct the split/merge history of trajectories
 - * recycling – flux and event count for recycled particles, on a per-target-state basis
 - * auxdata/ – auxiliary datasets (data stored on the ‘data’ field of Segment objects)

The file root object has an integer attribute ‘west_file_format_version’ which can be used to determine how to access data even as the file format (i.e. organization of data within HDF5 file) evolves.

Version history:

Version 9

- Basis states are now saved as iter_segid instead of just segid as a pointer label.
- Initial states are also saved in the iteration 0 file, with a negative sign.

Version 8

- Added external links to trajectory files in iterations/iter_* groups, if the HDF5 framework was used.
- Added an iter group for the iteration 0 to store conformations of basis states.

Version 7

- Removed bin_assignments, bin_populations, and bin_rates from iteration group.
- Added new_segments subgroup to iteration group

Version 6

- ???

Version 5

- moved iter_* groups into a top-level iterations/ group,
- added in-HDF5 storage for basis states, target states, and generated states


```
class westpa.core.data_manager.attrgetter(attr,/, *attrs)
```

Bases: object

Return a callable object that fetches the given attribute(s) from its operand. After `f = attrgetter('name')`, the call `f(r)` returns `r.name`. After `g = attrgetter('name', 'date')`, the call `g(r)` returns `(r.name, r.date)`. After `h = attrgetter('name.first', 'name.last')`, the call `h(r)` returns `(r.name.first, r.name.last)`.

```
westpa.core.data_manager.relpath(path, start=None)
```

Return a relative version of a path

```
westpa.core.data_manager.dirname(p)
```

Returns the directory component of a pathname

```
class westpa.core.data_manager.Segment(n_iter=None, seg_id=None, weight=None, endpoint_type=None,
                                     parent_id=None, wtg_parent_ids=None, pcoord=None,
                                     status=None, walltime=None, cputime=None, data=None)
```

Bases: object

A class wrapping segment data that must be passed through the work manager or data manager. Most fields are self-explanatory. One item worth noting is that a negative parent ID means that the segment starts from the initial state with ID `-(segment.parent_id+1)`

```
SEG_STATUS_UNSET = 0
```

```
SEG_STATUS_PREPARED = 1
```

```
SEG_STATUS_COMPLETE = 2
```

```
SEG_STATUS_FAILED = 3
```

```
SEG_INITPOINT_UNSET = 0
```

```
SEG_INITPOINT_CONTINUES = 1
```

```
SEG_INITPOINT_NEWTRAJ = 2
```

```
SEG_ENDPOINT_UNSET = 0
```

```
SEG_ENDPOINT_CONTINUES = 1
```

```
SEG_ENDPOINT_MERGED = 2
```

```
SEG_ENDPOINT_RECYCLED = 3
```

```
statuses = {'SEG_STATUS_COMPLETE': 2, 'SEG_STATUS_FAILED': 3, 'SEG_STATUS_PREPARED': 1, 'SEG_STATUS_UNSET': 0}
```

```
initpoint_types = {'SEG_INITPOINT_CONTINUES': 1, 'SEG_INITPOINT_NEWTRAJ': 2, 'SEG_INITPOINT_UNSET': 0}
```

```
endpoint_types = {'SEG_ENDPOINT_CONTINUES': 1, 'SEG_ENDPOINT_MERGED': 2, 'SEG_ENDPOINT_RECYCLED': 3, 'SEG_ENDPOINT_UNSET': 0}
```

```
status_names = {0: 'SEG_STATUS_UNSET', 1: 'SEG_STATUS_PREPARED', 2: 'SEG_STATUS_COMPLETE', 3: 'SEG_STATUS_FAILED'}
```

```
initpoint_type_names = {0: 'SEG_INITPOINT_UNSET', 1: 'SEG_INITPOINT_CONTINUES', 2: 'SEG_INITPOINT_NEWTRAJ'}
```

```
endpoint_type_names = {0: 'SEG_ENDPOINT_UNSET', 1: 'SEG_ENDPOINT_CONTINUES', 2:
'SEG_ENDPOINT_MERGED', 3: 'SEG_ENDPOINT_RECYCLED'}
```

```
static initial_pcoord(segment)
```

Return the initial progress coordinate point of this segment.

```
static final_pcoord(segment)
```

Return the final progress coordinate point of this segment.

```
property initpoint_type
```

```
property initial_state_id
```

```
property status_text
```

```
property endpoint_type_text
```

```
class westpa.core.data_manager.BasisState(label, probability, pcoord=None, auxref=None,
state_id=None)
```

Bases: object

Describes an basis (micro)state. These basis states are used to generate initial states for new trajectories, either at the beginning of the simulation (i.e. at `w_init`) or due to recycling.

Variables

- **state_id** – Integer identifier of this state, usually set by the data manager.
- **label** – A descriptive label for this microstate (may be empty)
- **probability** – Probability of this state to be selected when creating a new trajectory.
- **pcoord** – The representative progress coordinate of this state.
- **auxref** – A user-provided (string) reference for locating data associated with this state (usually a filesystem path).

```
classmethod states_to_file(states, fileobj)
```

Write a file defining basis states, which may then be read by `states_from_file()`.

```
classmethod states_from_file(statefile)
```

Read a file defining basis states. Each line defines a state, and contains a label, the probability, and optionally a data reference, separated by whitespace, as in:

```
unbound    1.0
```

or:

```
unbound_0   0.6    state0.pdb
unbound_1   0.4    state1.pdb
```

```
as_numpy_record()
```

Return the data for this state as a numpy record array.

```
class westpa.core.data_manager.TargetState(label, pcoord, state_id=None)
```

Bases: object

Describes a target state.

Variables

- **state_id** – Integer identifier of this state, usually set by the data manager.

- **label** – A descriptive label for this microstate (may be empty)
- **pcoord** – The representative progress coordinate of this state.

classmethod `states_to_file(states, fileobj)`

Write a file defining basis states, which may then be read by `states_from_file()`.

classmethod `states_from_file(statefile, dtype)`

Read a file defining target states. Each line defines a state, and contains a label followed by a representative progress coordinate value, separated by whitespace, as in:

```
bound      0.02
```

for a single target and one-dimensional progress coordinates or:

```
bound      2.7      0.0
drift      100      50.0
```

for two targets and a two-dimensional progress coordinate.

class `westpa.core.data_manager.InitialState(state_id, basis_state_id, iter_created, iter_used=None, istate_type=None, istate_status=None, pcoord=None, basis_state=None, basis_auxref=None)`

Bases: object

Describes an initial state for a new trajectory. These are generally constructed by appropriate modification of a basis state.

Variables

- **state_id** – Integer identifier of this state, usually set by the data manager.
- **basis_state_id** – Identifier of the basis state from which this state was generated, or None.
- **basis_state** – The *BasisState* from which this state was generated, or None.
- **iter_created** – Iteration in which this state was generated (0 for simulation initialization).
- **iter_used** – Iteration in which this state was used to initiate a trajectory (None for unused).
- **istate_type** – Integer describing the type of this initial state (ISTATE_TYPE_BASIS for direct use of a basis state, ISTATE_TYPE_GENERATED for a state generated from a basis state, ISTATE_TYPE_RESTART for a state corresponding to the endpoint of a segment in another simulation, or ISTATE_TYPE_START for a state generated from a start state).
- **istate_status** – Integer describing whether this initial state has been properly prepared.
- **pcoord** – The representative progress coordinate of this state.

ISTATE_TYPE_UNSET = 0

ISTATE_TYPE_BASIS = 1

ISTATE_TYPE_GENERATED = 2

ISTATE_TYPE_RESTART = 3

ISTATE_TYPE_START = 4

```
ISTATE_UNUSED = 0

ISTATE_STATUS_PENDING = 0

ISTATE_STATUS_PREPARED = 1

ISTATE_STATUS_FAILED = 2

istate_types = {'ISTATE_TYPE_BASIS': 1, 'ISTATE_TYPE_GENERATED': 2,
'ISTATE_TYPE_RESTART': 3, 'ISTATE_TYPE_START': 4, 'ISTATE_TYPE_UNSET': 0}

istate_type_names = {0: 'ISTATE_TYPE_UNSET', 1: 'ISTATE_TYPE_BASIS', 2:
'ISTATE_TYPE_GENERATED', 3: 'ISTATE_TYPE_RESTART', 4: 'ISTATE_TYPE_START'}

istate_statuses = {'ISTATE_STATUS_FAILED': 2, 'ISTATE_STATUS_PENDING': 0,
'ISTATE_STATUS_PREPARED': 1}

istate_status_names = {0: 'ISTATE_STATUS_PENDING', 1: 'ISTATE_STATUS_PREPARED', 2:
'ISTATE_STATUS_FAILED'}

as_numpy_record()

class westpa.core.data_manager.NewWeightEntry(source_type, weight, prev_seg_id=None,
                                              prev_init_pcoord=None, prev_final_pcoord=None,
                                              new_init_pcoord=None, target_state_id=None,
                                              initial_state_id=None)

    Bases: object

    NW_SOURCE_RECYCLED = 0

class westpa.core.data_manager.ExecutablePropagator(rc=None)
    Bases: WESTPropagator

    ENV_CURRENT_ITER = 'WEST_CURRENT_ITER'

    ENV_CURRENT_SEG_ID = 'WEST_CURRENT_SEG_ID'

    ENV_CURRENT_SEG_DATA_REF = 'WEST_CURRENT_SEG_DATA_REF'

    ENV_CURRENT_SEG_INITPOINT = 'WEST_CURRENT_SEG_INITPOINT_TYPE'

    ENV_PARENT_SEG_ID = 'WEST_PARENT_ID'

    ENV_PARENT_DATA_REF = 'WEST_PARENT_DATA_REF'

    ENV_BSTATE_ID = 'WEST_BSTATE_ID'

    ENV_BSTATE_DATA_REF = 'WEST_BSTATE_DATA_REF'

    ENV_ISTATE_ID = 'WEST_ISTATE_ID'

    ENV_ISTATE_DATA_REF = 'WEST_ISTATE_DATA_REF'

    ENV_STRUCT_DATA_REF = 'WEST_STRUCT_DATA_REF'

    ENV_RAND16 = 'WEST_RAND16'

    ENV_RAND32 = 'WEST_RAND32'
```

```
ENV_RAND64 = 'WEST_RAND64'
```

```
ENV_RAND128 = 'WEST_RAND128'
```

```
ENV_RANDFLOAT = 'WEST_RANDFLOAT'
```

```
static makepath(template, template_args=None, expanduser=True, expandvars=True, abspath=False,
                realpath=False)
```

```
random_val_env_vars()
```

Return a set of environment variables containing random seeds. These are returned as a dictionary, suitable for use in `os.environ.update()` or as the `env` argument to `subprocess.Popen()`. Every child process executed by `exec_child()` gets these.

```
exec_child(executable, environ=None, stdin=None, stdout=None, stderr=None, cwd=None)
```

Execute a child process with the environment set from the current environment, the values of `self.addtl_child_environs`, the random numbers returned by `self.random_val_env_vars`, and the given `environ` (applied in that order). `stdin/stdout/stderr` are optionally redirected.

This function waits on the child process to finish, then returns `(rc, rusage)`, where `rc` is the child's return code and `rusage` is the resource usage tuple from `os.wait4()`

```
exec_child_from_child_info(child_info, template_args, environ)
```

```
update_args_env_basis_state(template_args, environ, basis_state)
```

```
update_args_env_initial_state(template_args, environ, initial_state)
```

```
update_args_env_iter(template_args, environ, n_iter)
```

```
update_args_env_segment(template_args, environ, segment)
```

```
template_args_for_segment(segment)
```

```
exec_for_segment(child_info, segment, addtl_env=None)
```

Execute a child process with environment and template expansion from the given segment.

```
exec_for_iteration(child_info, n_iter, addtl_env=None)
```

Execute a child process with environment and template expansion from the given iteration number.

```
exec_for_basis_state(child_info, basis_state, addtl_env=None)
```

Execute a child process with environment and template expansion from the given basis state

```
exec_for_initial_state(child_info, initial_state, addtl_env=None)
```

Execute a child process with environment and template expansion from the given initial state.

```
prepare_file_system(segment, environ)
```

```
setup_dataset_return(segment=None, subset_keys=None)
```

Set up temporary files and environment variables that point to them for segment runners to return data. `segment` is the `Segment` object that the return data is associated with. `subset_keys` specifies the names of a subset of data to be returned.

```
retrieve_dataset_return(state, return_files, del_return_files, single_point)
```

Retrieve returned data from the temporary locations directed by the environment variables. `state` is a `Segment`, `BasisState`, or `InitialState` object that the return data is associated with. `return_files` is a dict where the keys are the dataset names and the values are the paths to the temporarily files that contain the returned data. `del_return_files` is a dict where the keys are the names of datasets to be deleted (if the corresponding value is set to `True`) once the data is retrieved.

get_pcoord(*state*)

Get the progress coordinate of the given basis or initial state.

gen_istate(*basis_state*, *initial_state*)

Generate a new initial state from the given basis state.

prepare_iteration(*n_iter*, *segments*)

Perform any necessary per-iteration preparation. This is run by the work manager.

finalize_iteration(*n_iter*, *segments*)

Perform any necessary post-iteration cleanup. This is run by the work manager.

propagate(*segments*)

Propagate one or more segments, including any necessary per-iteration setup and teardown for this propagator.

`westpa.core.data_manager.makepath(template, template_args=None, expanduser=True, expandvars=True, abspath=False, realpath=False)`

class `westpa.core.data_manager.flushing_lock(lock, fileobj)`

Bases: object

class `westpa.core.data_manager.expiring_flushing_lock(lock, flush_method, nextsync)`

Bases: object

`westpa.core.data_manager.seg_id_dtype`

alias of int64

`westpa.core.data_manager.n_iter_dtype`

alias of uint32

`westpa.core.data_manager.weight_dtype`

alias of float64

`westpa.core.data_manager.ptime_dtype`

alias of float64

`westpa.core.data_manager.seg_status_dtype`

alias of uint8

`westpa.core.data_manager.seg_initpoint_dtype`

alias of uint8

`westpa.core.data_manager.seg_endpoint_dtype`

alias of uint8

`westpa.core.data_manager.istate_type_dtype`

alias of uint8

`westpa.core.data_manager.istate_status_dtype`

alias of uint8

`westpa.core.data_manager.nw_source_dtype`

alias of uint8

class `westpa.core.data_manager.WESTDataManager(rc=None)`

Bases: object

Data manager for assisiting the reading and writing of WEST data from/to HDF5 files.

```

default_iter_prec = 8
default_we_h5filename = 'west.h5'
default_we_h5file_driver = None
default_flush_period = 60
default_aux_compression_threshold = 1048576
binning_hchunksize = 4096
table_scan_chunksize = 1024
flushing_lock()
expiring_flushing_lock()
process_config()
property system
property closed
iter_group_name(n_iter, absolute=True)
require_iter_group(n_iter)
    Get the group associated with n_iter, creating it if necessary.
del_iter_group(n_iter)
get_iter_group(n_iter)
get_seg_index(n_iter)
property current_iteration
open_backing(mode=None)
    Open the (already-created) HDF5 file named in self.west_h5filename.
prepare_backing()
    Create new HDF5 file
close_backing()
flush_backing()
save_target_states(tstates, n_iter=None)
    Save the given target states in the HDF5 file; they will be used for the next iteration to be propagated. A
    complete set is required, even if nominally appending to an existing set, which simplifies the mapping of
    IDs to the table.
find_tstate_group(n_iter)
find_ibstate_group(n_iter)
get_target_states(n_iter)
    Return a list of Target objects representing the target (sink) states that are in use for iteration n_iter. Future
    iterations are assumed to continue from the most recent set of states.

```

create_ibstate_group(*basis_states*, *n_iter=None*)

Create the group used to store basis states and initial states (whose definitions are always coupled). This group is hard-linked into all iteration groups that use these basis and initial states.

create_ibstate_iter_h5file(*basis_states*)

Create the per-iteration HDF5 file for the basis states (i.e., iteration 0). This special treatment is needed so that the analysis tools can access basis states more easily.

update_iter_h5file(*n_iter*, *segments*)

Write out the per-iteration HDF5 file with given segments and add an external link to it in the main HDF5 file (west.h5) if the link is not present.

get_basis_states(*n_iter=None*)

Return a list of BasisState objects representing the basis states that are in use for iteration *n_iter*.

create_initial_states(*n_states*, *n_iter=None*)

Create storage for *n_states* initial states associated with iteration *n_iter*, and return bare InitialState objects with only state_id set.

update_initial_states(*initial_states*, *n_iter=None*)

Save the given initial states in the HDF5 file

get_initial_states(*n_iter=None*)

get_segment_initial_states(*segments*, *n_iter=None*)

Retrieve all initial states referenced by the given segments.

get_unused_initial_states(*n_states=None*, *n_iter=None*)

Retrieve any prepared but unused initial states applicable to the given iteration. Up to *n_states* states are returned; if *n_states* is None, then all unused states are returned.

prepare_iteration(*n_iter*, *segments*)

Prepare for a new iteration by creating space to store the new iteration's data. The number of segments, their IDs, and their lineage must be determined and included in the set of segments passed in.

update_iter_group_links(*n_iter*)

Update the per-iteration hard links pointing to the tables of target and initial/basis states for the given iteration. These links are not used by this class, but are remarkably convenient for third-party analysis tools and hdfview.

get_iter_summary(*n_iter=None*)

update_iter_summary(*summary*, *n_iter=None*)

del_iter_summary(*min_iter*)

update_segments(*n_iter*, *segments*)

Update segment information in the HDF5 file; all prior information for each segment is overwritten, except for parent and weight transfer information.

get_segments(*n_iter=None*, *seg_ids=None*, *load_pcoords=True*)

Return the given (or all) segments from a given iteration.

If the optional parameter *load_auxdata* is true, then all auxiliary datasets available are loaded and mapped onto the data dictionary of each segment. If *load_auxdata* is None, then use the default *self.auto_load_auxdata*, which can be set by the option *load_auxdata* in the [data] section of *west.cfg*. This essentially requires as much RAM as there is per-iteration auxiliary data, so this behavior is not on by default.

prepare_segment_restarts(*segments*, *basis_states=None*, *initial_states=None*)

Prepare the necessary folder and files given the data stored in parent per-iteration HDF5 file for propagating the simulation. *basis_states* and *initial_states* should be provided if the segments are newly created

get_all_parent_ids(*n_iter*)

get_parent_ids(*n_iter*, *seg_ids=None*)

Return a sequence of the parent IDs of the given *seg_ids*.

get_weights(*n_iter*, *seg_ids*)

Return the weights associated with the given *seg_ids*

get_child_ids(*n_iter*, *seg_id*)

Return the *seg_ids* of segments who have the given segment as a parent.

get_children(*segment*)

Return all segments which have the given segment as a parent

prepare_run()

finalize_run()

save_new_weight_data(*n_iter*, *new_weights*)

Save a set of NewWeightEntry objects to HDF5. Note that this should be called for the iteration in which the weights appear in their new locations (e.g. for recycled walkers, the iteration following recycling).

get_new_weight_data(*n_iter*)

find_bin_mapper(*hashval*)

Check to see if the given has value is in the binning table. Returns the index in the bin data tables if found, or raises KeyError if not.

get_bin_mapper(*hashval*)

Look up the given hash value in the binning table, unpickling and returning the corresponding bin mapper if available, or raising KeyError if not.

save_bin_mapper(*hashval*, *pickle_data*)

Store the given mapper in the table of saved mappers. If the mapper cannot be stored, PickleError will be raised. Returns the index in the bin data tables where the mapper is stored.

save_iter_binning(*n_iter*, *hashval*, *pickled_mapper*, *target_counts*)

Save information about the binning used to generate segments for iteration *n_iter*.

`westpa.core.data_manager.normalize_dataset_options(dsopts, path_prefix="", n_iter=0)`

`westpa.core.data_manager.create_dataset_from_dsopts(group, dsopts, shape=None, dtype=None,
data=None, autocompress_threshold=None,
n_iter=None)`

`westpa.core.data_manager.require_dataset_from_dsopts(group, dsopts, shape=None, dtype=None,
data=None, autocompress_threshold=None,
n_iter=None)`

`westpa.core.data_manager.calc_chunksize(shape, dtype, max_chunksize=262144)`

Calculate a chunk size for HDF5 data, anticipating that access will slice along lower dimensions sooner than higher dimensions.

6.2.6.3 westpa.core.extloader module

`westpa.core.extloader.load_module(module_name, path=None)`

Load and return the given module, recursively loading containing packages as necessary.

`westpa.core.extloader.get_object(object_name, path=None)`

Attempt to load the given object, using additional path information if given.

6.2.6.4 westpa.core.h5io module

Miscellaneous routines to help with HDF5 input and output of WEST-related data.

class `westpa.core.h5io.Trajectory(xyz, topology, time=None, unitcell_lengths=None, unitcell_angles=None)`

Bases: `object`

Container object for a molecular dynamics trajectory

A Trajectory represents a collection of one or more molecular structures, generally (but not necessarily) from a molecular dynamics trajectory. The Trajectory stores a number of fields describing the system through time, including the cartesian coordinates of each atoms (`xyz`), the topology of the molecular system (`topology`), and information about the unitcell if appropriate (`unitcell_vectors`, `unitcell_length`, `unitcell_angles`).

A Trajectory should generally be constructed by loading a file from disk. Trajectories can be loaded from (and saved to) the PDB, XTC, TRR, DCD, binpos, NetCDF or MDTraj HDF5 formats.

Trajectory supports fancy indexing, so you can extract one or more frames from a Trajectory as a separate trajectory. For example, to form a trajectory with every other frame, you can slice with `traj[::2]`.

Trajectory uses the nanometer, degree & picosecond unit system.

Examples

```
>>> # loading a trajectory
>>> import mdtraj as md
>>> md.load('trajectory.xtc', top='native.pdb')
<mdtraj.Trajectory with 1000 frames, 22 atoms at 0x1058a73d0>
```

```
>>> # slicing a trajectory
>>> t = md.load('trajectory.h5')
>>> print(t)
<mdtraj.Trajectory with 100 frames, 22 atoms>
>>> print(t[::2])
<mdtraj.Trajectory with 50 frames, 22 atoms>
```

```
>>> # calculating the average distance between two atoms
>>> import mdtraj as md
>>> import numpy as np
>>> t = md.load('trajectory.h5')
>>> np.mean(np.sqrt(np.sum((t.xyz[:, 0, :] - t.xyz[:, 21, :])**2, axis=1)))
```

See also:

mdtraj.load

High-level function that loads files and returns an `md.Trajectory`

n_frames
 Type
 int

n_atoms
 Type
 int

n_residues
 Type
 int

time
 Type
 np.ndarray, shape=(n_frames,)

timestep
 Type
 float

topology
 Type
 md.Topology

top
 Type
 md.Topology

xyz
 Type
 np.ndarray, shape=(n_frames, n_atoms, 3)

unitcell_vectors
 Type
 {np.ndarray, shape=(n_frames, 3, 3), None}

unitcell_lengths
 Type
 {np.ndarray, shape=(n_frames, 3), None}

unitcell_angles
 Type
 {np.ndarray, shape=(n_frames, 3), None}

property n_frames
 Number of frames in the trajectory
Returns
n_frames – The number of frames in the trajectory
Return type
 int

property n_atoms
 Number of atoms in the trajectory
Returns
n_atoms – The number of atoms in the trajectory

Return type

int

property n_residues

Number of residues (amino acids) in the trajectory

Returns**n_residues** – The number of residues in the trajectory’s topology**Return type**

int

property n_chains

Number of chains in the trajectory

Returns**n_chains** – The number of chains in the trajectory’s topology**Return type**

int

property top

Alias for self.topology, describing the organization of atoms into residues, bonds, etc

Returns**topology** – The topology object, describing the organization of atoms into residues, bonds, etc**Return type**

md.Topology

property timestep

Timestep between frames, in picoseconds

Returns**timestep** – The timestep between frames, in picoseconds.**Return type**

float

property unitcell_vectors

The vectors that define the shape of the unit cell in each frame

Returns**vectors** – Vectors defining the shape of the unit cell in each frame. The semantics of this array are that the shape of the unit cell in frame *i* are given by the three vectors, `value[i, 0, :]`, `value[i, 1, :]`, and `value[i, 2, :]`.**Return type**

np.ndarray, shape(n_frames, 3, 3)

property unitcell_volumes

Volumes of unit cell for each frame.

Returns**volumes** – Volumes of the unit cell in each frame, in nanometers³, or None if the Trajectory contains no unitcell information.**Return type**

{np.ndarray, shape=(n_frames), None}

superpose(reference, frame=0, atom_indices=None, ref_atom_indices=None, parallel=True)

Superpose each conformation in this trajectory upon a reference

Parameters

- **reference** (*md.Trajectory*) – Align self to a particular frame in *reference*
- **frame** (*int*) – The index of the conformation in *reference* to align to.
- **atom_indices** (*array_like, or None*) – The indices of the atoms to superpose. If not supplied, all atoms will be used.
- **ref_atom_indices** (*array_like, or None*) – Use these atoms on the reference structure. If not supplied, the same atom indices will be used for this trajectory and the reference one.
- **parallel** (*bool*) – Use OpenMP to run the superposition in parallel over multiple cores

Return type

self

join(*other, check_topology=True, discard_overlapping_frames=False*)

Join two trajectories together along the time/frame axis.

This method joins trajectories along the time axis, giving a new trajectory of length equal to the sum of the lengths of *self* and *other*. It can also be called by using *self* + *other*

Parameters

- **other** (*Trajectory or list of Trajectory*) – One or more trajectories to join with this one. These trajectories are *appended* to the end of this trajectory.
- **check_topology** (*bool*) – Ensure that the topology of *self* and *other* are identical before joining them. If false, the resulting trajectory will have the topology of *self*.
- **discard_overlapping_frames** (*bool, optional*) – If True, compare coordinates at trajectory edges to discard overlapping frames. Default: False.

See also:

stack

join two trajectories along the atom axis

stack(*other, keep_resSeq=True*)

Stack two trajectories along the atom axis

This method joins trajectories along the atom axis, giving a new trajectory with a number of atoms equal to the sum of the number of atoms in *self* and *other*.

Notes

The resulting trajectory will have the unitcell and time information the left operand.

Examples

```
>>> t1 = md.load('traj1.h5')
>>> t2 = md.load('traj2.h5')
>>> # even when t2 contains no unitcell information
>>> t2.unitcell_vectors = None
>>> stacked = t1.stack(t2)
>>> # the stacked trajectory inherits the unitcell information
>>> # from the first trajectory
```

(continues on next page)

(continued from previous page)

```
>>> np.all(stacked.unitcell_vectors == t1.unitcell_vectors)
True
```

Parameters

- **other** (*Trajectory*) – The other trajectory to join
- **keep_resSeq** (*bool, optional, default=True*) – see ``mdtraj.core.topology.Topology.join`` method documentation

See also:***join***

join two trajectories along the time/frame axis.

slice(*key, copy=True*)

Slice trajectory, by extracting one or more frames into a separate object

This method can also be called using index bracket notation, i.e `traj[1] == traj.slice(1)`

Parameters

- **key** (*{int, np.ndarray, slice}*) – The slice to take. Can be either an int, a list of ints, or a slice object.
- **copy** (*bool, default=True*) – Copy the arrays after slicing. If you set this to false, then if you modify a slice, you'll modify the original array since they point to the same data.

property topology

Topology of the system, describing the organization of atoms into residues, bonds, etc

Returns

topology – The topology object, describing the organization of atoms into residues, bonds, etc

Return type

md.Topology

property xyz

Cartesian coordinates of each atom in each simulation frame

Returns

xyz – A three dimensional numpy array, with the cartesian coordinates of each atoms in each frame.

Return type

np.ndarray, shape=(n_frames, n_atoms, 3)

property unitcell_lengths

Lengths that define the shape of the unit cell in each frame.

Returns

lengths – Lengths of the unit cell in each frame, in nanometers, or None if the Trajectory contains no unitcell information.

Return type

{np.ndarray, shape=(n_frames, 3), None}

property unitcell_angles

Angles that define the shape of the unit cell in each frame.

Returns

lengths – The angles between the three unitcell vectors in each frame, alpha, beta, and gamma. alpha' gives the angle between vectors `b` and `c`, beta gives the angle between vectors `c` and `a`, and gamma gives the angle between vectors `a` and `b`. The angles are in degrees.

Return type

np.ndarray, shape=(n_frames, 3)

property time

The simulation time corresponding to each frame, in picoseconds

Returns

time – The simulation time corresponding to each frame, in picoseconds

Return type

np.ndarray, shape=(n_frames,)

openmm_positions(frame)

OpenMM-compatible positions of a single frame.

Examples

```
>>> t = md.load('trajectory.h5')
>>> context.setPositions(t.openmm_positions(0))
```

Parameters

frame (*int*) – The index of frame of the trajectory that you wish to extract

Returns

positions – The cartesian coordinates of specific trajectory frame, formatted for input to OpenMM

Return type

list

openmm_boxes(frame)

OpenMM-compatible box vectors of a single frame.

Examples

```
>>> t = md.load('trajectory.h5')
>>> context.setPeriodicBoxVectors(t.openmm_positions(0))
```

Parameters

frame (*int*) – Return box for this single frame.

Returns

box – The periodic box vectors for this frame, formatted for input to OpenMM.

Return type

tuple

static load(filenamees, **kwargs)

Load a trajectory from disk

Parameters

- **filenamees** (*{path-like, [path-like]}*) – Either a path or list of paths

- **extension** (As requested by the various load functions -- it depends on the) –

save(filename, **kwargs)

Save trajectory to disk, in a format determined by the filename extension

Parameters

- **filename** (path-like) – filesystem path in which to save the trajectory. The extension will be parsed and will control the format.
- **lossy** (bool) – For .h5 or .lh5, whether or not to use compression.
- **no_models** (bool) – For .pdb. TODO: Document this?
- **force_overwrite** (bool) – If filename already exists, overwrite it.

save_hdf5(filename, force_overwrite=True)

Save trajectory to MDTraj HDF5 format

Parameters

- **filename** (path-like) – filesystem path in which to save the trajectory
- **force_overwrite** (bool, default=True) – Overwrite anything that exists at filename, if its already there

save_lammpstrj(filename, force_overwrite=True)

Save trajectory to LAMMPS custom dump format

Parameters

- **filename** (path-like) – filesystem path in which to save the trajectory
- **force_overwrite** (bool, default=True) – Overwrite anything that exists at filename, if its already there

save_xyz(filename, force_overwrite=True)

Save trajectory to .xyz format.

Parameters

- **filename** (path-like) – filesystem path in which to save the trajectory
- **force_overwrite** (bool, default=True) – Overwrite anything that exists at filename, if its already there

save_pdb(filename, force_overwrite=True, bfactors=None)

Save trajectory to RCSB PDB format

Parameters

- **filename** (path-like) – filesystem path in which to save the trajectory
- **force_overwrite** (bool, default=True) – Overwrite anything that exists at filename, if its already there
- **bfactors** (array_like, default=None, shape=(n_frames, n_atoms) or (n_atoms,)) – Save bfactors with pdb file. If the array is two dimensional it should contain a bfactor for each atom in each frame of the trajectory. Otherwise, the same bfactor will be saved in each frame.

save_xtc(filename, force_overwrite=True)

Save trajectory to Gromacs XTC format

Parameters

- **filename** (path-like) – filesystem path in which to save the trajectory

- **force_overwrite** (*bool*, *default=True*) – Overwrite anything that exists at filename, if its already there

save_trr(*filename*, *force_overwrite=True*)

Save trajectory to Gromacs TRR format

Notes

Only the xyz coordinates and the time are saved, the velocities and forces in the trr will be zeros

Parameters

- **filename** (*path-like*) – filesystem path in which to save the trajectory
- **force_overwrite** (*bool*, *default=True*) – Overwrite anything that exists at filename, if its already there

save_dcd(*filename*, *force_overwrite=True*)

Save trajectory to CHARMM/NAMD DCD format

Parameters

- **filename** (*path-like*) – filesystem path in which to save the trajectory
- **force_overwrite** (*bool*, *default=True*) – Overwrite anything that exists at filenames, if its already there

save_dtr(*filename*, *force_overwrite=True*)

Save trajectory to DESMOND DTR format

Parameters

- **filename** (*path-like*) – filesystem path in which to save the trajectory
- **force_overwrite** (*bool*, *default=True*) – Overwrite anything that exists at filenames, if its already there

save_binpos(*filename*, *force_overwrite=True*)

Save trajectory to AMBER BINPOS format

Parameters

- **filename** (*path-like*) – filesystem path in which to save the trajectory
- **force_overwrite** (*bool*, *default=True*) – Overwrite anything that exists at filename, if its already there

save_mdcrd(*filename*, *force_overwrite=True*)

Save trajectory to AMBER mdcrd format

Parameters

- **filename** (*path-like*) – filesystem path in which to save the trajectory
- **force_overwrite** (*bool*, *default=True*) – Overwrite anything that exists at filename, if its already there

save_netcdf(*filename*, *force_overwrite=True*)

Save trajectory in AMBER NetCDF format

Parameters

- **filename** (*path-like*) – filesystem path in which to save the trajectory
- **force_overwrite** (*bool*, *default=True*) – Overwrite anything that exists at filename, if it's already there

save_netcdfirst(*filename, force_overwrite=True*)

Save trajectory in AMBER NetCDF restart format

Parameters

- **filename** (*path-like*) – filesystem path in which to save the restart
- **force_overwrite** (*bool, default=True*) – Overwrite anything that exists at filename, if it's already there

Notes

NetCDF restart files can only store a single frame. If only one frame exists, “filename” will be written. Otherwise, “filename.#” will be written, where # is a zero-padded number from 1 to the total number of frames in the trajectory

save_amberrst7(*filename, force_overwrite=True*)

Save trajectory in AMBER ASCII restart format

Parameters

- **filename** (*path-like*) – filesystem path in which to save the restart
- **force_overwrite** (*bool, default=True*) – Overwrite anything that exists at filename, if it's already there

Notes

Amber restart files can only store a single frame. If only one frame exists, “filename” will be written. Otherwise, “filename.#” will be written, where # is a zero-padded number from 1 to the total number of frames in the trajectory

save_lh5(*filename, force_overwrite=True*)

Save trajectory in deprecated MSMBuild2 LH5 (lossy HDF5) format.

Parameters

- **filename** (*path-like*) – filesystem path in which to save the trajectory
- **force_overwrite** (*bool, default=True*) – Overwrite anything that exists at filename, if it's already there

save_gro(*filename, force_overwrite=True, precision=3*)

Save trajectory in Gromacs .gro format

Parameters

- **filename** (*path-like*) – Path to save the trajectory
- **force_overwrite** (*bool, default=True*) – Overwrite anything that exists at that filename if it exists
- **precision** (*int, default=3*) – The number of decimal places to use for coordinates in GRO file

save_tng(*filename, force_overwrite=True*)

Save trajectory to Gromacs TNG format

Parameters

- **filename** (*path-like*) – filesystem path in which to save the trajectory
- **force_overwrite** (*bool, default=True*) – Overwrite anything that exists at filename, if its already there

save_gsd(*filename*, *force_overwrite*=True)

Save trajectory to HOOMD GSD format

Parameters

- **filename** (*path-like*) – filesystem path in which to save the trajectory
- **force_overwrite** (*bool*, *default*=True) – Overwrite anything that exists at filenames, if its already there

center_coordinates(*mass_weighted*=False)

Center each trajectory frame at the origin (0,0,0).

This method acts inplace on the trajectory. The centering can be either uniformly weighted (*mass_weighted*=False) or weighted by the mass of each atom (*mass_weighted*=True).

Parameters

mass_weighted (*bool*, *optional* (*default* = False)) – If True, weight atoms by mass when removing COM.

Return type

self

restrict_atoms(***kwargs*)

DEPRECATED: `restrict_atoms` was replaced by `atom_slice` and will be removed in 2.0

Retain only a subset of the atoms in a trajectory

Deletes atoms not in *atom_indices*, and re-indexes those that remain

atom_indices

[array-like, *dtype*=int, *shape*=(*n_atoms*)] List of atom indices to keep.

inplace

[*bool*, *default*=True] If True, the operation is done inplace, modifying *self*. Otherwise, a copy is returned with the restricted atoms, and *self* is not modified.

traj

[*md.Trajectory*] The return value is either *self*, or the new trajectory, depending on the value of *inplace*.

atom_slice(*atom_indices*, *inplace*=False)

Create a new trajectory from a subset of atoms

Parameters

- **atom_indices** (*array-like*, *dtype*=int, *shape*=(*n_atoms*)) – List of indices of atoms to retain in the new trajectory.
- **inplace** (*bool*, *default*=False) – If True, the operation is done inplace, modifying *self*. Otherwise, a copy is returned with the sliced atoms, and *self* is not modified.

Returns

traj – The return value is either *self*, or the new trajectory, depending on the value of *inplace*.

Return type

md.Trajectory

See also:

[*stack*](#)

stack multiple trajectories along the atom axis

remove_solvent(*exclude=None, inplace=False*)

Create a new trajectory without solvent atoms

Parameters

- **exclude** (*array-like, dtype=str, shape=(n_solvent_types)*) – List of solvent residue names to retain in the new trajectory.
- **inplace** (*bool, default=False*) – The return value is either `self`, or the new trajectory, depending on the value of `inplace`.

Returns

traj – The return value is either `self`, or the new trajectory, depending on the value of `inplace`.

Return type

`md.Trajectory`

smooth(*width, order=3, atom_indices=None, inplace=False*)

Smoothen a trajectory using a zero-delay Buttersworth filter. Please note that for optimal results the trajectory should be properly aligned prior to smoothing (see *md.Trajectory.superpose*).

Parameters

- **width** (*int*) – This acts very similar to the window size in a moving average smoother. In this implementation, the frequency of the low-pass filter is taken to be two over this width, so it's like "half the period" of the sinusoid where the filter starts to kick in. Must be an integer greater than one.
- **order** (*int, optional, default=3*) – The order of the filter. A small odd number is recommended. Higher order filters cutoff more quickly, but have worse numerical properties.
- **atom_indices** (*array-like, dtype=int, shape=(n_atoms), default=None*) – List of indices of atoms to retain in the new trajectory. Default is set to *None*, which applies smoothing to all atoms.
- **inplace** (*bool, default=False*) – The return value is either `self`, or the new trajectory, depending on the value of `inplace`.

Returns

traj – The return value is either `self`, or the new smoothed trajectory, depending on the value of `inplace`.

Return type

`md.Trajectory`

References

make_molecules_whole(*inplace=False, sorted_bonds=None*)

Only make molecules whole

Parameters

- **inplace** (*bool*) – If `False`, a new `Trajectory` is created and returned. If `True`, this `Trajectory` is modified directly.
- **sorted_bonds** (*array of shape (n_bonds, 2)*) – Pairs of atom indices that define bonds, in sorted order. If not specified, these will be determined from the trajectory's topology.

See also:

[*image_molecules*](#)

image_molecules(*inplace=False, anchor_molecules=None, other_molecules=None, sorted_bonds=None, make_whole=True*)

Recenter and apply periodic boundary conditions to the molecules in each frame of the trajectory.

This method is useful for visualizing a trajectory in which molecules were not wrapped to the periodic unit cell, or in which the macromolecules are not centered with respect to the solvent. It tries to be intelligent in deciding what molecules to center, so you can simply call it and trust that it will “do the right thing”.

Parameters

- **inplace** (*bool, default=False*) – If False, a new Trajectory is created and returned. If True, this Trajectory is modified directly.
- **anchor_molecules** (*list of atom sets, optional, default=None*) – Molecule that should be treated as an “anchor”. These molecules will be centered in the box and put near each other. If not specified, anchor molecules are guessed using a heuristic.
- **other_molecules** (*list of atom sets, optional, default=None*) – Molecules that are not anchors. If not specified, these will be molecules other than the anchor molecules
- **sorted_bonds** (*array of shape (n_bonds, 2)*) – Pairs of atom indices that define bonds, in sorted order. If not specified, these will be determined from the trajectory’s topology. Only relevant if **make_whole** is True.
- **make_whole** (*bool*) – Whether to make molecules whole.

Returns

traj – The return value is either **self** or the new trajectory, depending on the value of **inplace**.

Return type

md.Trajectory

See also:

Topology.guess_anchor_molecules

westpa.core.h5io.**join_traj**(*trajs, check_topology=True, discard_overlapping_frames=False*)

Concatenate multiple trajectories into one long trajectory

Parameters

- **trajs** (*iterable of trajectories*) – Combine these into one trajectory
- **check_topology** (*bool*) – Make sure topologies match before joining
- **discard_overlapping_frames** (*bool*) – Check for overlapping frames and discard

westpa.core.h5io.**in_units_of**(*quantity, units_in, units_out, inplace=False*)

Convert a numerical quantity between unit systems.

Parameters

- **quantity** (*{number, np.ndarray, openmm.unit.Quantity}*) – quantity can either be a unitless quantity – i.e. instance of openmm.unit.Quantity, or just a bare number or numpy array
- **units_in** (*str*) – If you supply a quantity that’s not a openmm.unit.Quantity, you should tell me what units it is in. If you don’t, i’m just going to echo you back your quantity without doing any unit checking.
- **units_out** (*str*) – A string description of the units you want out. This should look like “nanometers/picosecond” or “nanometers**3” or whatever

- **inplace** (*bool*) – Attempt to do the transformation inplace, by mutating the *quantity* argument and avoiding a copy. This is only possible if *quantity* is a writable numpy array.

Returns

rquantity – The resulting quantity, in the new unit system. If the function was called with *inplace=True* and *quantity* was a writable numpy array, *rquantity* will alias the same memory as the input *quantity*, which will have been changed inplace. Otherwise, if a copy was required, *rquantity* will point to new memory.

Return type

{number, np.ndarray}

Examples

```
>>> in_units_of(1, 'meter**2/second', 'nanometers**2/picosecond')
10000000.0
```

`westpa.core.h5io.import_(module)`

Import a module, and issue a nice message to stderr if the module isn't installed.

Currently, this function will print nice error messages for networkx, tables, netCDF4, and openmm.unit, which are optional MDTraj dependencies.

Parameters

module (*str*) – The module you'd like to import, as a string

Returns

module – The module object

Return type

{module, object}

Examples

```
>>> # the following two lines are equivalent. the difference is that the
>>> # second will check for an ImportError and print you a very nice
>>> # user-facing message about what's wrong (where you can install the
>>> # module from, etc) if the import fails
>>> import tables
>>> tables = import_('tables')
```

`westpa.core.h5io.ensure_type(val, dtype, ndim, name, length=None, can_be_none=False, shape=None, warn_on_cast=True, add_newaxis_on_deficient_ndim=False)`

Typecheck the size, shape and dtype of a numpy array, with optional casting.

Parameters

- **val** ({*np.ndarray*, *None*}) – The array to check
- **dtype** ({*nd.dtype*, *str*}) – The dtype you'd like the array to have
- **ndim** (*int*) – The number of dimensions you'd like the array to have
- **name** (*str*) – name of the array. This is used when throwing exceptions, so that we can describe to the user which array is messed up.
- **length** (*int*, *optional*) – How long should the array be?
- **can_be_none** (*bool*) – Is *val == None* acceptable?

- **shape** (*tuple, optional*) – What should be shape of the array be? If the provided tuple has Nones in it, those will be semantically interpreted as matching any length in that dimension. So, for example, using the shape spec (`None, None, 3`) will ensure that the last dimension is of length three without constraining the first two dimensions
- **warn_on_cast** (*bool, default=True*) – Raise a warning when the dtypes don't match and a cast is done.
- **add_newaxis_on_deficient_ndim** (*bool, default=True*) – Add a new axis to the beginning of the array if the number of dimensions is deficient by one compared to your specification. For instance, if you're trying to get out an array of `ndim == 3`, but the user provides an array of `shape == (10, 10)`, a new axis will be created with length 1 in front, so that the return value is of shape `(1, 10, 10)`.

Notes

The returned value will always be C-contiguous.

Returns

typechecked_val – If `val=None` and `can_be_none=True`, then this will return `None`. Otherwise, it will return `val` (or a copy of `val`). If the dtype wasn't right, it'll be casted to the right shape. If the array was not C-contiguous, it'll be copied as well.

Return type

`np.ndarray, None`

```
class westpa.core.h5io.HDF5TrajectoryFile(filename, mode='r', force_overwrite=True,
                                         compression='zlib')
```

Bases: `object`

Interface for reading and writing to a MDTraj HDF5 molecular dynamics trajectory file, whose format is described [here](#).

This is a file-like object, that both reading or writing depending on the *mode* flag. It implements the context manager protocol, so you can also use it with the python 'with' statement.

The format is extremely flexible and high performance. It can hold a wide variety of information about a trajectory, including fields like the temperature and energies. Because it's built on the fantastic HDF5 library, it's easily extensible too.

Parameters

- **filename** (*path-like*) – Path to the file to open
- **mode** (`{'r', 'w'}`) – Mode in which to open the file. 'r' is for reading and 'w' is for writing
- **force_overwrite** (*bool*) – In mode='w', how do you want to behave if a file by the name of *filename* already exists? if `force_overwrite=True`, it will be overwritten.
- **compression** (`{'zlib', None}`) – Apply compression to the file? This will save space, and does not cost too many cpu cycles, so it's recommended.

root

title

application

topology

randomState

forcefield

reference

constraints

See also:

mdtraj.load_hdf5

High-level wrapper that returns a `md.Trajectory`

distance_unit = 'nanometers'

property root

Direct access to the root group of the underlying Tables HDF5 file handle.

This can be used for random or specific access to the underlying arrays on disk

property title

User-defined title for the data represented in the file

property application

Suite of programs that created the file

property topology

Get the topology out from the file

Returns

topology – A topology object

Return type

`mdtraj.Topology`

property randomState

State of the creators internal random number generator at the start of the simulation

property forcefield

Description of the hamiltonian used. A short, human readable string, like AMBER99sbildn.

property reference

A published reference that documents the program or parameters used to generate the data

property constraints

Constraints applied to the bond lengths

Returns

constraints – A one dimensional array with the a int, int, float type giving the index of the two atoms involved in the constraints and the distance of the constraint. If no constraint information is in the file, the return value is None.

Return type

`{None, np.array, dtype=[('atom1', '<i4'), ('atom2', '<i4'), ('distance', '<f4')]}`

read_as_traj(*n_frames=None, stride=None, atom_indices=None*)

Read a trajectory from the HDF5 file

Parameters

- **n_frames** (*{int, None}*) – The number of frames to read. If not supplied, all of the remaining frames will be read.
- **stride** (*{int, None}*) – By default all of the frames will be read, but you can pass this flag to read a subset of of the data by grabbing only every *stride*-th frame from disk.

- **atom_indices** (*{int, None}*) – By default all of the atom will be read, but you can pass this flag to read only a subsets of the atoms for the *coordinates* and *velocities* fields. Note that you will have to carefully manage the indices and the offsets, since the *i*-th atom in the topology will not necessarily correspond to the *i*-th atom in your subset.

Returns

trajectory – A trajectory object containing the loaded portion of the file.

Return type

Trajectory

read(*n_frames=None, stride=None, atom_indices=None*)

Read one or more frames of data from the file

Parameters

- **n_frames** (*{int, None}*) – The number of frames to read. If not supplied, all of the remaining frames will be read.
- **stride** (*{int, None}*) – By default all of the frames will be read, but you can pass this flag to read a subset of of the data by grabbing only every *stride*-th frame from disk.
- **atom_indices** (*{int, None}*) – By default all of the atom will be read, but you can pass this flag to read only a subsets of the atoms for the *coordinates* and *velocities* fields. Note that you will have to carefully manage the indices and the offsets, since the *i*-th atom in the topology will not necessarily correspond to the *i*-th atom in your subset.

Notes

If you'd like more flexible access to the data, that is available by using the pytables group directly, which is accessible via the *root* property on this class.

Returns

frames – The returned namedtuple will have the fields “coordinates”, “time”, “cell_lengths”, “cell_angles”, “velocities”, “kineticEnergy”, “potentialEnergy”, “temperature” and “alchemicalLambda”. Each of the fields in the returned namedtuple will either be a numpy array or None, dependending on if that data was saved in the trajectory. All of the data shall be n units of “nanometers”, “picoseconds”, “kelvin”, “degrees” and “kilojoules_per_mole”.

Return type

namedtuple

write(*coordinates, time=None, cell_lengths=None, cell_angles=None, velocities=None, kineticEnergy=None, potentialEnergy=None, temperature=None, alchemicalLambda=None*)

Write one or more frames of data to the file

This method saves data that is associated with one or more simulation frames. Note that all of the arguments can either be raw numpy arrays or unitted arrays (with `openmm.unit.Quantity`). If the arrays are unitted, a unit conversion will be automatically done from the supplied units into the proper units for saving on disk. You won't have to worry about it.

Furthermore, if you wish to save a single frame of simulation data, you can do so naturally, for instance by supplying a 2d array for the coordinates and a single float for the time. This “shape deficiency” will be recognized, and handled appropriately.

Parameters

- **coordinates** (*np.ndarray, shape=(n_frames, n_atoms, 3)*) – The cartesian coordinates of the atoms to write. By convention, the lengths should be in units of nanometers.
- **time** (*np.ndarray, shape=(n_frames,), optional*) – You may optionally specify the simulation time, in picoseconds corresponding to each frame.
- **cell_lengths** (*np.ndarray, shape=(n_frames, 3), dtype=float32, optional*) – You may optionally specify the unitcell lengths. The length of the periodic box in each frame, in each direction, *a, b, c*. By convention the lengths should be in units of angstroms.
- **cell_angles** (*np.ndarray, shape=(n_frames, 3), dtype=float32, optional*) – You may optionally specify the unitcell angles in each frame. Organized analogously to *cell_lengths*. Gives the alpha, beta and gamma angles respectively. By convention, the angles should be in units of degrees.
- **velocities** (*np.ndarray, shape=(n_frames, n_atoms, 3), optional*) – You may optionally specify the cartesian components of the velocity for each atom in each frame. By convention, the velocities should be in units of nanometers / picosecond.
- **kineticEnergy** (*np.ndarray, shape=(n_frames,), optional*) – You may optionally specify the kinetic energy in each frame. By convention the kinetic energies should be in units of kilojoules per mole.
- **potentialEnergy** (*np.ndarray, shape=(n_frames,), optional*) – You may optionally specify the potential energy in each frame. By convention the kinetic energies should be in units of kilojoules per mole.
- **temperature** (*np.ndarray, shape=(n_frames,), optional*) – You may optionally specify the temperature in each frame. By convention the temperatures should be in units of Kelvin.
- **alchemicalLambda** (*np.ndarray, shape=(n_frames,), optional*) – You may optionally specify the alchemical lambda in each frame. These have no units, but are generally between zero and one.

seek(*offset, whence=0*)

Move to a new file position

Parameters

- **offset** (*int*) – A number of frames.
- **whence** (*{0, 1, 2}*) – 0: offset from start of file, offset should be ≥ 0 . 1: move relative to the current position, positive or negative 2: move relative to the end of file, offset should be ≤ 0 . Seeking beyond the end of a file is not supported

tell()

Current file position

Returns

offset – The current frame in the file.

Return type

int

close()

Close the HDF5 file handle

flush()

Write all buffered data in the to the disk file.

```
class westpa.core.h5io.Frames(coordinates, time, cell_lengths, cell_angles, velocities, kineticEnergy,  
                             potentialEnergy, temperature, alchemicalLambda)
```

Bases: `tuple`

Create new instance of Frames(coordinates, time, cell_lengths, cell_angles, velocities, kineticEnergy, potentialEnergy, temperature, alchemicalLambda)

alchemicalLambda

Alias for field number 8

cell_angles

Alias for field number 3

cell_lengths

Alias for field number 2

coordinates

Alias for field number 0

kineticEnergy

Alias for field number 5

potentialEnergy

Alias for field number 6

temperature

Alias for field number 7

time

Alias for field number 1

velocities

Alias for field number 4

```
class westpa.core.h5io.WESTTrajectory(coordinates, topology=None, time=None, iter_labels=None,  
                                     seg_labels=None, pcoords=None, parent_ids=None,  
                                     unitcell_lengths=None, unitcell_angles=None)
```

Bases: `Trajectory`

A subclass of `mdtraj.Trajectory` that contains the trajectory of atom coordinates with pointers denoting the iteration number and segment index of each frame.

iter_label_values()

seg_label_values(*iteration=None*)

property label_values**property iter_labels**

Iteration index corresponding to each frame

Returns

time – The iteration index corresponding to each frame

Return type

`np.ndarray`, shape=(n_frames,)

property seg_labels

Segment index corresponding to each frame

Returns

time – The segment index corresponding to each frame

Return type

np.ndarray, shape=(n_frames,)

property pcoords**property parent_ids****join**(*other*, *check_topology=True*, *discard_overlapping_frames=False*)

Join two Trajectory's. This overrides `mdtraj.Trajectory.join` so that it also handles WESTPA pointers. `mdtraj.Trajectory.join`'s documentation for more details.

slice(*key*, *copy=True*)

Slice the Trajectory. This overrides `mdtraj.Trajectory.slice` so that it also handles WESTPA pointers. Please see `mdtraj.Trajectory.slice`'s documentation for more details.

westpa.core.h5io.resolve_filepath(*path*, *constructor=<class 'h5py._hl.files.File'>*, *cargs=None*, *ckwargs=None*, ***additkwargs*)

Use a combined filesystem and HDF5 path to open an HDF5 file and return the appropriate object. Returns (h5file, h5object). The file is opened using `constructor(filename, *cargs, **kwargs)`.

westpa.core.h5io.calc_chunksize(*shape*, *dtype*, *max_chunksize=262144*)

Calculate a chunk size for HDF5 data, anticipating that access will slice along lower dimensions sooner than higher dimensions.

westpa.core.h5io.tostr(*b*)

Convert a nonstandard string object *b* to str with the handling of the case where *b* is bytes.

westpa.core.h5io.is_within_directory(*directory*, *target*)

westpa.core.h5io.safe_extract(*tar*, *path='.'*, *members=None*, **, numeric_owner=False*)

westpa.core.h5io.create_hdf5_group(*parent_group*, *groupname*, *replace=False*, *creating_program=None*)

Create (or delete and recreate) and HDF5 group named *groupname* within the enclosing Group (object) *parent_group*. If *replace* is True, then the group is replaced if present; if False, then an error is raised if the group is present. After the group is created, HDF5 attributes are set using *stamp_creator_data*.

westpa.core.h5io.stamp_creator_data(*h5group*, *creating_program=None*)

Mark the following on the HDF5 group *h5group*:

creation_program

The name of the program that created the group

creation_user

The username of the user who created the group

creation_hostname

The hostname of the machine on which the group was created

creation_time

The date and time at which the group was created, in the current locale.

creation_unix_time

The Unix time (seconds from the epoch, UTC) at which the group was created.

This is meant to facilitate tracking the flow of data, but should not be considered a secure paper trail (after all, anyone with write access to the HDF5 file can modify these attributes).

`westpa.core.h5io.get_creator_data(h5group)`

Read back creator data as written by `stamp_creator_data`, returning a dictionary with keys as described for `stamp_creator_data`. Missing fields are denoted with `None`. The `creation_time` field is returned as a string.

`westpa.core.h5io.load_west(filename)`

Load WESTPA trajectory files from disk.

Parameters

filename (*str*) – String filename of HDF Trajectory file.

`westpa.core.h5io.stamp_iter_range(h5object, start_iter, stop_iter)`

Mark that the HDF5 object `h5object` (dataset or group) contains data from iterations `start_iter` \leq `n_iter` $<$ `stop_iter`.

`westpa.core.h5io.get_iter_range(h5object)`

Read back iteration range data written by `stamp_iter_range`

`westpa.core.h5io.stamp_iter_step(h5group, iter_step)`

Mark that the HDF5 object `h5object` (dataset or group) contains data with an iteration step (stride) of `iter_step`.

`westpa.core.h5io.get_iter_step(h5group)`

Read back iteration step (stride) written by `stamp_iter_step`

`westpa.core.h5io.check_iter_range_least(h5object, iter_start, iter_stop)`

Return True if the iteration range [`iter_start`, `iter_stop`] is the same as or entirely contained within the iteration range stored on `h5object`.

`westpa.core.h5io.check_iter_range_equal(h5object, iter_start, iter_stop)`

Return True if the iteration range [`iter_start`, `iter_stop`] is the same as the iteration range stored on `h5object`.

`westpa.core.h5io.get_iteration_entry(h5object, n_iter)`

Create a slice for data corresponding to iteration `n_iter` in `h5object`.

`westpa.core.h5io.get_iteration_slice(h5object, iter_start, iter_stop=None, iter_stride=None)`

Create a slice for data corresponding to iterations [`iter_start`, `iter_stop`), with stride `iter_step`, in the given `h5object`.

`westpa.core.h5io.label_axes(h5object, labels, units=None)`

Stamp the given HDF5 object with axis labels. This stores the axis labels in an array of strings in an attribute called `axis_labels` on the given object. `units` if provided is a corresponding list of units.

class `westpa.core.h5io.WESTPAH5File(*args, **kwargs)`

Bases: `File`

Generalized input/output for WESTPA simulation (or analysis) data.

Create a new file object.

See the `h5py` user guide for a detailed explanation of the options.

name

Name of the file on disk, or file-like object. Note: for files created with the ‘core’ driver, HDF5 still requires this be non-empty.

mode

r Readonly, file must exist (default) r+ Read/write, file must exist w Create file, truncate if exists w- or x Create file, fail if exists a Read/write if exists, create otherwise

driver

Name of the driver to use. Legal values are `None` (default, recommended), ‘core’, ‘sec2’, ‘direct’, ‘stdio’, ‘mpio’, ‘ros3’.

libver

Library version bounds. Supported values: 'earliest', 'v108', 'v110', 'v112' and 'latest'. The 'v108', 'v110' and 'v112' options can only be specified with the HDF5 1.10.2 library or later.

userblock_size

Desired size of user block. Only allowed when creating a new file (mode w, w- or x).

swmr

Open the file in SWMR read mode. Only used when mode = 'r'.

rdcc_nbytes

Total size of the dataset chunk cache in bytes. The default size is 1024**2 (1 MiB) per dataset. Applies to all datasets unless individually changed.

rdcc_w0

The chunk preemption policy for all datasets. This must be between 0 and 1 inclusive and indicates the weighting according to which chunks which have been fully read or written are penalized when determining which chunks to flush from cache. A value of 0 means fully read or written chunks are treated no differently than other chunks (the preemption is strictly LRU) while a value of 1 means fully read or written chunks are always preempted before other chunks. If your application only reads or writes data once, this can be safely set to 1. Otherwise, this should be set lower depending on how often you re-read or re-write the same data. The default value is 0.75. Applies to all datasets unless individually changed.

rdcc_nslots

The number of chunk slots in the raw data chunk cache for this file. Increasing this value reduces the number of cache collisions, but slightly increases the memory used. Due to the hashing strategy, this value should ideally be a prime number. As a rule of thumb, this value should be at least 10 times the number of chunks that can fit in rdcc_nbytes bytes. For maximum performance, this value should be set approximately 100 times that number of chunks. The default value is 521. Applies to all datasets unless individually changed.

track_order

Track dataset/group/attribute creation order under root group if True. If None use global default h5.get_config().track_order.

fs_strategy

The file space handling strategy to be used. Only allowed when creating a new file (mode w, w- or x). Defined as: "fsm" FSM, Aggregators, VFD "page" Paged FSM, VFD "aggregate" Aggregators, VFD "none" VFD If None use HDF5 defaults.

fs_page_size

File space page size in bytes. Only used when fs_strategy="page". If None use the HDF5 default (4096 bytes).

fs_persist

A boolean value to indicate whether free space should be persistent or not. Only allowed when creating a new file. The default value is False.

fs_threshold

The smallest free-space section size that the free space manager will track. Only allowed when creating a new file. The default value is 1.

page_buf_size

Page buffer size in bytes. Only allowed for HDF5 files created with fs_strategy="page". Must be a power of two value and greater or equal than the file space page size when creating the file. It is not used by default.

min_meta_keep

Minimum percentage of metadata to keep in the page buffer before allowing pages containing metadata to be evicted. Applicable only if page_buf_size is set. Default value is zero.

min_raw_keep

Minimum percentage of raw data to keep in the page buffer before allowing pages containing raw data to be evicted. Applicable only if page_buf_size is set. Default value is zero.

locking

The file locking behavior. Defined as:

- False (or "false") – Disable file locking

- True (or “true”) – Enable file locking
- “best-effort” – Enable file locking but ignore some errors
- None – Use HDF5 defaults

Warning: The `HDF5_USE_FILE_LOCKING` environment variable can override this parameter.

Only available with HDF5 $\geq 1.12.1$ or $1.10.x \geq 1.10.7$.

alignment_threshold

Together with `alignment_interval`, this property ensures that any file object greater than or equal in size to the alignment threshold (in bytes) will be aligned on an address which is a multiple of alignment interval.

alignment_interval

This property should be used in conjunction with `alignment_threshold`. See the description above. For more details, see https://portal.hdfgroup.org/display/HDF5/H5P_SET_ALIGNMENT

meta_block_size

Set the current minimum size, in bytes, of new metadata block allocations. See https://portal.hdfgroup.org/display/HDF5/H5P_SET_META_BLOCK_SIZE

Additional keywords

Passed on to the selected file driver.

default_iter_prec = 8

replace_dataset(*args, **kwargs)

iter_object_name(n_iter, prefix="", suffix="")

Return a properly-formatted per-iteration name for iteration `n_iter`. (This is used in `create/require/get_iter_group`, but may also be useful for naming datasets on a per-iteration basis.)

create_iter_group(n_iter, group=None)

Create a per-iteration data storage group for iteration number `n_iter` in the group `group` (which is ‘iterations’ by default).

require_iter_group(n_iter, group=None)

Ensure that a per-iteration data storage group for iteration number `n_iter` is available in the group `group` (which is ‘iterations’ by default).

get_iter_group(n_iter, group=None)

Get the per-iteration data group for iteration number `n_iter` from within the group `group` (‘iterations’ by default).

class westpa.core.h5io.WESTIterationFile(file, mode='r', force_overwrite=True, compression='zlib', link=None)

Bases: [HDF5TrajectoryFile](#)

read(frame_indices=None, atom_indices=None)

Read one or more frames of data from the file

Parameters

- **n_frames** ({int, None}) – The number of frames to read. If not supplied, all of the remaining frames will be read.
- **stride** ({int, None}) – By default all of the frames will be read, but you can pass this flag to read a subset of of the data by grabbing only every *stride*-th frame from disk.
- **atom_indices** ({int, None}) – By default all of the atom will be read, but you can pass this flag to read only a subsets of the atoms for the *coordinates* and

velocities fields. Note that you will have to carefully manage the indices and the offsets, since the *i*-th atom in the topology will not necessarily correspond to the *i*-th atom in your subset.

Notes

If you'd like more flexible access to the data, that is available by using the pytables group directly, which is accessible via the *root* property on this class.

Returns

frames – The returned namedtuple will have the fields “coordinates”, “time”, “cell_lengths”, “cell_angles”, “velocities”, “kineticEnergy”, “potentialEnergy”, “temperature” and “alchemicalLambda”. Each of the fields in the returned namedtuple will either be a numpy array or None, depending on if that data was saved in the trajectory. All of the data shall be in units of “nanometers”, “picoseconds”, “kelvin”, “degrees” and “kilojoules_per_mole”.

Return type

namedtuple

has_topology()

has_pointer()

has_restart(*segment*)

write_data(*where, name, data*)

read_data(*where, name*)

read_as_traj(*iteration=None, segment=None, atom_indices=None*)

Read a trajectory from the HDF5 file

Parameters

- **n_frames** (*{int, None}*) – The number of frames to read. If not supplied, all of the remaining frames will be read.
- **stride** (*{int, None}*) – By default all of the frames will be read, but you can pass this flag to read a subset of the data by grabbing only every *stride*-th frame from disk.
- **atom_indices** (*{int, None}*) – By default all of the atom will be read, but you can pass this flag to read only a subsets of the atoms for the *coordinates* and *velocities* fields. Note that you will have to carefully manage the indices and the offsets, since the *i*-th atom in the topology will not necessarily correspond to the *i*-th atom in your subset.

Returns

trajectory – A trajectory object containing the loaded portion of the file.

Return type

Trajectory

read_restart(*segment*)

write_segment(*segment, pop=False*)


```
class westpa.core.h5io.DSSpec
```

Bases: object

Generalized WE dataset access

```
get_iter_data(n_iter, seg_slice=(slice(None, None, None),))
```

```
get_segment_data(n_iter, seg_id)
```

```
class westpa.core.h5io.FileLinkedDSSpec(h5file_or_name)
```

Bases: [DSSpec](#)

Provide facilities for accessing WESTPA HDF5 files, including auto-opening and the ability to pickle references to such files for transmission (through, e.g., the work manager), provided that the HDF5 file can be accessed by the same path on both the sender and receiver.

property h5file

Lazily open HDF5 file. This is required because allowing an open HDF5 file to cross a fork() boundary generally corrupts the internal state of the HDF5 library.

```
class westpa.core.h5io.SingleDSSpec(h5file_or_name, dsname, alias=None, slice=None)
```

Bases: [FileLinkedDSSpec](#)

```
classmethod from_string(dsspec_string, default_h5file)
```

```
class westpa.core.h5io.SingleIterDSSpec(h5file_or_name, dsname, alias=None, slice=None)
```

Bases: [SingleDSSpec](#)

```
get_iter_data(n_iter, seg_slice=(slice(None, None, None),))
```

```
class westpa.core.h5io.SingleSegmentDSSpec(h5file_or_name, dsname, alias=None, slice=None)
```

Bases: [SingleDSSpec](#)

```
get_iter_data(n_iter, seg_slice=(slice(None, None, None),))
```

```
get_segment_data(n_iter, seg_id)
```

```
class westpa.core.h5io.FnDSSpec(h5file_or_name, fn)
```

Bases: [FileLinkedDSSpec](#)

```
get_iter_data(n_iter, seg_slice=(slice(None, None, None),))
```

```
class westpa.core.h5io.MultiDSSpec(dsspecs)
```

Bases: [DSSpec](#)

```
get_iter_data(n_iter, seg_slice=(slice(None, None, None),))
```

```
class westpa.core.h5io.IterBlockedDataset(dataset_or_array, attrs=None)
```

Bases: object

```
classmethod empty_like(blocked_dataset)
```

```
cache_data(max_size=None)
```

Cache this dataset in RAM. If max_size is given, then only cache if the entire dataset fits in max_size bytes. If max_size is the string 'available', then only cache if the entire dataset fits in available RAM, as defined by the psutil module.

```
drop_cache()
```

`iter_entry(n_iter)`

`iter_slice(start=None, stop=None)`

6.2.6.5 westpa.core.progress module

`westpa.core.progress.linregress(x, y=None, alternative='two-sided')`

Calculate a linear least-squares regression for two sets of measurements.

Parameters

- **x** (*array_like*) – Two sets of measurements. Both arrays should have the same length. If only *x* is given (and *y=None*), then it must be a two-dimensional array where one dimension has length 2. The two sets of measurements are then found by splitting the array along the length-2 dimension. In the case where *y=None* and *x* is a 2x2 array, `linregress(x)` is equivalent to `linregress(x[0], x[1])`.
- **y** (*array_like*) – Two sets of measurements. Both arrays should have the same length. If only *x* is given (and *y=None*), then it must be a two-dimensional array where one dimension has length 2. The two sets of measurements are then found by splitting the array along the length-2 dimension. In the case where *y=None* and *x* is a 2x2 array, `linregress(x)` is equivalent to `linregress(x[0], x[1])`.
- **alternative** (*{'two-sided', 'less', 'greater'}, optional*) – Defines the alternative hypothesis. Default is 'two-sided'. The following options are available:
 - 'two-sided': the slope of the regression line is nonzero
 - 'less': the slope of the regression line is less than zero
 - 'greater': the slope of the regression line is greater than zero

New in version 1.7.0.

Returns

result – The return value is an object with the following attributes:

slope

[float] Slope of the regression line.

intercept

[float] Intercept of the regression line.

rvalue

[float] The Pearson correlation coefficient. The square of *rvalue* is equal to the coefficient of determination.

pvalue

[float] The p-value for a hypothesis test whose null hypothesis is that the slope is zero, using Wald Test with t-distribution of the test statistic. See *alternative* above for alternative hypotheses.

stderr

[float] Standard error of the estimated slope (gradient), under the assumption of residual normality.

intercept_stderr

[float] Standard error of the estimated intercept, under the assumption of residual normality.

Return type

`LinregressResult` instance

See also:

scipy.optimize.curve_fit

Use non-linear least squares to fit a function to data.

scipy.optimize.leastsq

Minimize the sum of squares of a set of equations.

Notes

Missing values are considered pair-wise: if a value is missing in x , the corresponding value in y is masked.

For compatibility with older versions of SciPy, the return value acts like a `namedtuple` of length 5, with fields `slope`, `intercept`, `rvalue`, `pvalue` and `stderr`, so one can continue to write:

```
slope, intercept, r, p, se = linregress(x, y)
```

With that style, however, the standard error of the intercept is not available. To have access to all the computed values, including the standard error of the intercept, use the return value as an object with attributes, e.g.:

```
result = linregress(x, y)
print(result.intercept, result.intercept_stderr)
```

Examples

```
>>> import numpy as np
>>> import matplotlib.pyplot as plt
>>> from scipy import stats
>>> rng = np.random.default_rng()
```

Generate some data:

```
>>> x = rng.random(10)
>>> y = 1.6*x + rng.random(10)
```

Perform the linear regression:

```
>>> res = stats.linregress(x, y)
```

Coefficient of determination (R-squared):

```
>>> print(f"R-squared: {res.rvalue**2:.6f}")
R-squared: 0.717533
```

Plot the data along with the fitted line:

```
>>> plt.plot(x, y, 'o', label='original data')
>>> plt.plot(x, res.intercept + res.slope*x, 'r', label='fitted line')
>>> plt.legend()
>>> plt.show()
```

Calculate 95% confidence interval on slope and intercept:

```
>>> # Two-sided inverse Students t-distribution
>>> # p - probability, df - degrees of freedom
>>> from scipy.stats import t
>>> tinv = lambda p, df: abs(t.ppf(p/2, df))
```

```
>>> ts = tinv(0.05, len(x)-2)
>>> print(f"slope (95%): {res.slope:.6f} +/- {ts*res.stderr:.6f}")
slope (95%): 1.453392 +/- 0.743465
>>> print(f"intercept (95%): {res.intercept:.6f}"
...      f" +/- {ts*res.intercept_stderr:.6f}")
intercept (95%): 0.616950 +/- 0.544475
```

```
westpa.core.progress.nop()
```

```
class westpa.core.progress.ProgressIndicator(stream=None, interval=1)
```

Bases: object

draw_fancy()

draw_simple()

draw()

clear()

property operation

property extent

property progress

new_operation(operation, extent=None, progress=0)

start()

stop()

6.2.6.6 westpa.core.segment module

```
class westpa.core.segment.Segment(n_iter=None, seg_id=None, weight=None, endpoint_type=None,
                                   parent_id=None, wtg_parent_ids=None, pcoord=None, status=None,
                                   walltime=None, cputime=None, data=None)
```

Bases: object

A class wrapping segment data that must be passed through the work manager or data manager. Most fields are self-explanatory. One item worth noting is that a negative parent ID means that the segment starts from the initial state with ID $-(\text{segment.parent_id}+1)$

SEG_STATUS_UNSET = 0

SEG_STATUS_PREPARED = 1

SEG_STATUS_COMPLETE = 2

SEG_STATUS_FAILED = 3

```

SEG_INITPOINT_UNSET = 0
SEG_INITPOINT_CONTINUES = 1
SEG_INITPOINT_NEWTRAJ = 2
SEG_ENDPOINT_UNSET = 0
SEG_ENDPOINT_CONTINUES = 1
SEG_ENDPOINT_MERGED = 2
SEG_ENDPOINT_RECYCLED = 3

statuses = {'SEG_STATUS_COMPLETE': 2, 'SEG_STATUS_FAILED': 3, 'SEG_STATUS_PREPARED':
1, 'SEG_STATUS_UNSET': 0}

initpoint_types = {'SEG_INITPOINT_CONTINUES': 1, 'SEG_INITPOINT_NEWTRAJ': 2,
'SEG_INITPOINT_UNSET': 0}

endpoint_types = {'SEG_ENDPOINT_CONTINUES': 1, 'SEG_ENDPOINT_MERGED': 2,
'SEG_ENDPOINT_RECYCLED': 3, 'SEG_ENDPOINT_UNSET': 0}

status_names = {0: 'SEG_STATUS_UNSET', 1: 'SEG_STATUS_PREPARED', 2:
'SEG_STATUS_COMPLETE', 3: 'SEG_STATUS_FAILED'}

initpoint_type_names = {0: 'SEG_INITPOINT_UNSET', 1: 'SEG_INITPOINT_CONTINUES', 2:
'SEG_INITPOINT_NEWTRAJ'}

endpoint_type_names = {0: 'SEG_ENDPOINT_UNSET', 1: 'SEG_ENDPOINT_CONTINUES', 2:
'SEG_ENDPOINT_MERGED', 3: 'SEG_ENDPOINT_RECYCLED'}

static initial_pcoord(segment)
    Return the initial progress coordinate point of this segment.

static final_pcoord(segment)
    Return the final progress coordinate point of this segment.

property initpoint_type
property initial_state_id
property status_text
property endpoint_type_text

```

6.2.6.7 westpa.core.sim_manager module

class westpa.core.sim_manager.timedelta

Bases: object

Difference between two datetime values.

timedelta(days=0, seconds=0, microseconds=0, milliseconds=0, minutes=0, hours=0, weeks=0)

All arguments are optional and default to 0. Arguments may be integers or floats, and may be positive or negative.

days

Number of days.

max = `datetime.timedelta(days=999999999, seconds=86399, microseconds=999999)`

microseconds

Number of microseconds (≥ 0 and less than 1 second).

min = `datetime.timedelta(days=-999999999)`

resolution = `datetime.timedelta(microseconds=1)`

seconds

Number of seconds (≥ 0 and less than 1 day).

total_seconds()

Total seconds in the duration.

class `westpa.core.sim_manager.zip_longest`

Bases: `object`

`zip_longest(iter1 [,iter2 [...]], [fillvalue=None])` -> `zip_longest` object

Return a `zip_longest` object whose `__next__()` method returns a tuple where the *i*-th element comes from the *i*-th iterable argument. The `__next__()` method continues until the longest iterable in the argument sequence is exhausted and then it raises `StopIteration`. When the shorter iterables are exhausted, the `fillvalue` is substituted in their place. The `fillvalue` defaults to `None` or can be specified by a keyword argument.

exception `westpa.core.sim_manager.PickleError`

Bases: `Exception`

`westpa.core.sim_manager.weight_dtype`

alias of `float64`

class `westpa.core.sim_manager.Segment`(*n_iter=None, seg_id=None, weight=None, endpoint_type=None, parent_id=None, wtg_parent_ids=None, pcoord=None, status=None, walltime=None, cputime=None, data=None*)

Bases: `object`

A class wrapping segment data that must be passed through the work manager or data manager. Most fields are self-explanatory. One item worth noting is that a negative parent ID means that the segment starts from the initial state with ID $-(\text{segment.parent_id}+1)$

SEG_STATUS_UNSET = 0

SEG_STATUS_PREPARED = 1

SEG_STATUS_COMPLETE = 2

SEG_STATUS_FAILED = 3

SEG_INITPOINT_UNSET = 0

SEG_INITPOINT_CONTINUES = 1

SEG_INITPOINT_NEWTRAJ = 2

SEG_ENDPOINT_UNSET = 0

```
SEG_ENDPOINT_CONTINUES = 1
```

```
SEG_ENDPOINT_MERGED = 2
```

```
SEG_ENDPOINT_RECYCLED = 3
```

```
statuses = {'SEG_STATUS_COMPLETE': 2, 'SEG_STATUS_FAILED': 3, 'SEG_STATUS_PREPARED': 1, 'SEG_STATUS_UNSET': 0}
```

```
initpoint_types = {'SEG_INITPOINT_CONTINUES': 1, 'SEG_INITPOINT_NEWTRAJ': 2, 'SEG_INITPOINT_UNSET': 0}
```

```
endpoint_types = {'SEG_ENDPOINT_CONTINUES': 1, 'SEG_ENDPOINT_MERGED': 2, 'SEG_ENDPOINT_RECYCLED': 3, 'SEG_ENDPOINT_UNSET': 0}
```

```
status_names = {0: 'SEG_STATUS_UNSET', 1: 'SEG_STATUS_PREPARED', 2: 'SEG_STATUS_COMPLETE', 3: 'SEG_STATUS_FAILED'}
```

```
initpoint_type_names = {0: 'SEG_INITPOINT_UNSET', 1: 'SEG_INITPOINT_CONTINUES', 2: 'SEG_INITPOINT_NEWTRAJ'}
```

```
endpoint_type_names = {0: 'SEG_ENDPOINT_UNSET', 1: 'SEG_ENDPOINT_CONTINUES', 2: 'SEG_ENDPOINT_MERGED', 3: 'SEG_ENDPOINT_RECYCLED'}
```

```
static initial_pcoord(segment)
```

Return the initial progress coordinate point of this segment.

```
static final_pcoord(segment)
```

Return the final progress coordinate point of this segment.

```
property initpoint_type
```

```
property initial_state_id
```

```
property status_text
```

```
property endpoint_type_text
```

```
class westpa.core.sim_manager.InitialState(state_id, basis_state_id, iter_created, iter_used=None,
                                           istate_type=None, istate_status=None, pcoord=None,
                                           basis_state=None, basis_auxref=None)
```

Bases: object

Describes an initial state for a new trajectory. These are generally constructed by appropriate modification of a basis state.

Variables

- **state_id** – Integer identifier of this state, usually set by the data manager.
- **basis_state_id** – Identifier of the basis state from which this state was generated, or None.
- **basis_state** – The *BasisState* from which this state was generated, or None.
- **iter_created** – Iteration in which this state was generated (0 for simulation initialization).
- **iter_used** – Iteration in which this state was used to initiate a trajectory (None for unused).

- **istate_type** – Integer describing the type of this initial state (ISTATE_TYPE_BASIS for direct use of a basis state, ISTATE_TYPE_GENERATED for a state generated from a basis state, ISTATE_TYPE_RESTART for a state corresponding to the endpoint of a segment in another simulation, or ISTATE_TYPE_START for a state generated from a start state).
- **istate_status** – Integer describing whether this initial state has been properly prepared.
- **pcoord** – The representative progress coordinate of this state.

ISTATE_TYPE_UNSET = 0

ISTATE_TYPE_BASIS = 1

ISTATE_TYPE_GENERATED = 2

ISTATE_TYPE_RESTART = 3

ISTATE_TYPE_START = 4

ISTATE_UNUSED = 0

ISTATE_STATUS_PENDING = 0

ISTATE_STATUS_PREPARED = 1

ISTATE_STATUS_FAILED = 2

```
istate_types = {'ISTATE_TYPE_BASIS': 1, 'ISTATE_TYPE_GENERATED': 2,  
'ISTATE_TYPE_RESTART': 3, 'ISTATE_TYPE_START': 4, 'ISTATE_TYPE_UNSET': 0}
```

```
istate_type_names = {0: 'ISTATE_TYPE_UNSET', 1: 'ISTATE_TYPE_BASIS', 2:  
'ISTATE_TYPE_GENERATED', 3: 'ISTATE_TYPE_RESTART', 4: 'ISTATE_TYPE_START'}
```

```
istate_statuses = {'ISTATE_STATUS_FAILED': 2, 'ISTATE_STATUS_PENDING': 0,  
'ISTATE_STATUS_PREPARED': 1}
```

```
istate_status_names = {0: 'ISTATE_STATUS_PENDING', 1: 'ISTATE_STATUS_PREPARED', 2:  
'ISTATE_STATUS_FAILED'}
```

as_numpy_record()

westpa.core.sim_manager.grouper(*n, iterable, fillvalue=None*)

Collect data into fixed-length chunks or blocks

exception westpa.core.sim_manager.PropagationError

Bases: RuntimeError

class westpa.core.sim_manager.WESimManager(*rc=None*)

Bases: object

process_config()

register_callback(*hook, function, priority=0*)

Registers a callback to execute during the given hook into the simulation loop. The optional priority is used to order when the function is called relative to other registered callbacks.

invoke_callbacks(*hook, *args, **kwargs*)

load_plugins(*plugins=None*)

report_bin_statistics(*bins, target_states, save_summary=False*)

get_bstate_pcoords(*basis_states, label='basis'*)

For each of the given *basis_states*, calculate progress coordinate values as necessary. The HDF5 file is not updated.

report_basis_states(*basis_states, label='basis'*)

report_target_states(*target_states*)

initialize_simulation(*basis_states, target_states, start_states, segs_per_state=1, suppress_we=False*)

Initialize a new weighted ensemble simulation, taking *segs_per_state* initial states from each of the given *basis_states*.

w_init is the forward-facing version of this function

prepare_iteration()

finalize_iteration()

Clean up after an iteration and prepare for the next.

get_istate_futures()

Add *n_states* initial states to the internal list of initial states assigned to recycled particles. Spare states are used if available, otherwise new states are created. If created new initial states requires generation, then a set of futures is returned representing work manager tasks corresponding to the necessary generation work.

propagate()

save_bin_data()

Calculate and write flux and transition count matrices to HDF5. Population and rate matrices are likely useless at the single-tau level and are no longer written.

check_propagation()

Check for failures in propagation or initial state generation, and raise an exception if any are found.

run_we()

Run the weighted ensemble algorithm based on the binning in *self.final_bins* and the recycled particles in *self.to_recycle*, creating and committing the next iteration's segments to storage as well.

prepare_new_iteration()

Commit data for the coming iteration to the HDF5 file.

run()

prepare_run()

Prepare a new run.

finalize_run()

Perform cleanup at the normal end of a run

pre_propagation()

post_propagation()

pre_we()

post_we()

6.2.6.8 westpa.core.states module

```
class westpa.core.states.Segment(n_iter=None, seg_id=None, weight=None, endpoint_type=None,  
                                parent_id=None, wtg_parent_ids=None, pcoord=None, status=None,  
                                walltime=None, cputime=None, data=None)
```

Bases: object

A class wrapping segment data that must be passed through the work manager or data manager. Most fields are self-explanatory. One item worth noting is that a negative parent ID means that the segment starts from the initial state with ID $-(\text{segment.parent_id}+1)$

SEG_STATUS_UNSET = 0

SEG_STATUS_PREPARED = 1

SEG_STATUS_COMPLETE = 2

SEG_STATUS_FAILED = 3

SEG_INITPOINT_UNSET = 0

SEG_INITPOINT_CONTINUES = 1

SEG_INITPOINT_NEWTRAJ = 2

SEG_ENDPOINT_UNSET = 0

SEG_ENDPOINT_CONTINUES = 1

SEG_ENDPOINT_MERGED = 2

SEG_ENDPOINT_RECYCLED = 3

statuses = {'SEG_STATUS_COMPLETE': 2, 'SEG_STATUS_FAILED': 3, 'SEG_STATUS_PREPARED': 1, 'SEG_STATUS_UNSET': 0}

initpoint_types = {'SEG_INITPOINT_CONTINUES': 1, 'SEG_INITPOINT_NEWTRAJ': 2, 'SEG_INITPOINT_UNSET': 0}

endpoint_types = {'SEG_ENDPOINT_CONTINUES': 1, 'SEG_ENDPOINT_MERGED': 2, 'SEG_ENDPOINT_RECYCLED': 3, 'SEG_ENDPOINT_UNSET': 0}

status_names = {0: 'SEG_STATUS_UNSET', 1: 'SEG_STATUS_PREPARED', 2: 'SEG_STATUS_COMPLETE', 3: 'SEG_STATUS_FAILED'}

initpoint_type_names = {0: 'SEG_INITPOINT_UNSET', 1: 'SEG_INITPOINT_CONTINUES', 2: 'SEG_INITPOINT_NEWTRAJ'}

endpoint_type_names = {0: 'SEG_ENDPOINT_UNSET', 1: 'SEG_ENDPOINT_CONTINUES', 2: 'SEG_ENDPOINT_MERGED', 3: 'SEG_ENDPOINT_RECYCLED'}

static initial_pcoord(*segment*)

Return the initial progress coordinate point of this segment.

static final_pcoord(*segment*)

Return the final progress coordinate point of this segment.

property initpoint_type

property `initial_state_id`

property `status_text`

property `endpoint_type_text`

class `westpa.core.states.BasisState(label, probability, pcoord=None, auxref=None, state_id=None)`

Bases: `object`

Describes an basis (micro)state. These basis states are used to generate initial states for new trajectories, either at the beginning of the simulation (i.e. at `w_init`) or due to recycling.

Variables

- **state_id** – Integer identifier of this state, usually set by the data manager.
- **label** – A descriptive label for this microstate (may be empty)
- **probability** – Probability of this state to be selected when creating a new trajectory.
- **pcoord** – The representative progress coordinate of this state.
- **auxref** – A user-provided (string) reference for locating data associated with this state (usually a filesystem path).

classmethod `states_to_file(states, fileobj)`

Write a file defining basis states, which may then be read by `states_from_file()`.

classmethod `states_from_file(statefile)`

Read a file defining basis states. Each line defines a state, and contains a label, the probability, and optionally a data reference, separated by whitespace, as in:

```
unbound      1.0
```

or:

```
unbound_0    0.6      state0.pdb
unbound_1    0.4      state1.pdb
```

as_numpy_record()

Return the data for this state as a numpy record array.

class `westpa.core.states.InitialState(state_id, basis_state_id, iter_created, iter_used=None, istate_type=None, istate_status=None, pcoord=None, basis_state=None, basis_auxref=None)`

Bases: `object`

Describes an initial state for a new trajectory. These are generally constructed by appropriate modification of a basis state.

Variables

- **state_id** – Integer identifier of this state, usually set by the data manager.
- **basis_state_id** – Identifier of the basis state from which this state was generated, or `None`.
- **basis_state** – The *BasisState* from which this state was generated, or `None`.
- **iter_created** – Iteration in which this state was generated (0 for simulation initialization).
- **iter_used** – Iteration in which this state was used to initiate a trajectory (`None` for unused).

- **istate_type** – Integer describing the type of this initial state (ISTATE_TYPE_BASIS for direct use of a basis state, ISTATE_TYPE_GENERATED for a state generated from a basis state, ISTATE_TYPE_RESTART for a state corresponding to the endpoint of a segment in another simulation, or ISTATE_TYPE_START for a state generated from a start state).
- **istate_status** – Integer describing whether this initial state has been properly prepared.
- **pcoord** – The representative progress coordinate of this state.

```
ISTATE_TYPE_UNSET = 0
```

```
ISTATE_TYPE_BASIS = 1
```

```
ISTATE_TYPE_GENERATED = 2
```

```
ISTATE_TYPE_RESTART = 3
```

```
ISTATE_TYPE_START = 4
```

```
ISTATE_UNUSED = 0
```

```
ISTATE_STATUS_PENDING = 0
```

```
ISTATE_STATUS_PREPARED = 1
```

```
ISTATE_STATUS_FAILED = 2
```

```
istate_types = {'ISTATE_TYPE_BASIS': 1, 'ISTATE_TYPE_GENERATED': 2,  
'ISTATE_TYPE_RESTART': 3, 'ISTATE_TYPE_START': 4, 'ISTATE_TYPE_UNSET': 0}
```

```
istate_type_names = {0: 'ISTATE_TYPE_UNSET', 1: 'ISTATE_TYPE_BASIS', 2:  
'ISTATE_TYPE_GENERATED', 3: 'ISTATE_TYPE_RESTART', 4: 'ISTATE_TYPE_START'}
```

```
istate_statuses = {'ISTATE_STATUS_FAILED': 2, 'ISTATE_STATUS_PENDING': 0,  
'ISTATE_STATUS_PREPARED': 1}
```

```
istate_status_names = {0: 'ISTATE_STATUS_PENDING', 1: 'ISTATE_STATUS_PREPARED', 2:  
'ISTATE_STATUS_FAILED'}
```

```
as_numpy_record()
```

```
class westpa.core.states.TargetState(label, pcoord, state_id=None)
```

Bases: object

Describes a target state.

Variables

- **state_id** – Integer identifier of this state, usually set by the data manager.
- **label** – A descriptive label for this microstate (may be empty)
- **pcoord** – The representative progress coordinate of this state.

```
classmethod states_to_file(states, fileobj)
```

Write a file defining basis states, which may then be read by *states_from_file()*.

```
classmethod states_from_file(statefile, dtype)
```

Read a file defining target states. Each line defines a state, and contains a label followed by a representative progress coordinate value, separated by whitespace, as in:

```
bound      0.02
```

for a single target and one-dimensional progress coordinates or:

```
bound      2.7      0.0
drift      100      50.0
```

for two targets and a two-dimensional progress coordinate.

`westpa.core.states.pare_basis_initial_states(basis_states, initial_states, segments=None)`

Given iterables of basis and initial states (and optionally segments that use them), return minimal sets (as in `__builtins__.set`) of states needed to describe the history of the given segments and initial states.

`westpa.core.states.return_state_type(state_obj)`

Convenience function for returning the state ID and type of the `state_obj` pointer

6.2.6.9 westpa.core.systems module

class `westpa.core.systems.NopMapper`

Bases: `BinMapper`

Put everything into one bin.

assign(*coords*, *mask=None*, *output=None*)

class `westpa.core.systems.WESTSystem(rc=None)`

Bases: `object`

A description of the system being simulated, including the dimensionality and data type of the progress coordinate, the number of progress coordinate entries expected from each segment, and binning. To construct a simulation, the user must subclass `WESTSystem` and set several instance variables.

At a minimum, the user must subclass `WESTSystem` and override **method: `initialize`** to set the data type and dimensionality of progress coordinate data and define a bin mapper.

Variables

- **pcoord_ndim** – The number of dimensions in the progress coordinate. Defaults to 1 (i.e. a one-dimensional progress coordinate).
- **pcoord_dtype** – The data type of the progress coordinate, which must be callable (e.g. `np.float32` and `long` will work, but `'<f4'` and `'<i8'` will not). Defaults to `np.float64`.
- **pcoord_len** – The length of the progress coordinate time series generated by each segment, including *both* the initial and final values. Defaults to 2 (i.e. only the initial and final progress coordinate values for a segment are returned from propagation).
- **bin_mapper** – A bin mapper describing the progress coordinate space.
- **bin_target_counts** – A vector of target counts, one per bin.

property `bin_target_counts`

initialize()

Prepare this system object for use in simulation or analysis, creating a bin space, setting replicas per bin, and so on. This function is called whenever a WEST tool creates an instance of the system driver.

prepare_run()

Prepare this system for use in a simulation run. Called by `w_run` in all worker processes.

finalize_run()

A hook for system-specific processing for the end of a simulation run (as defined by such things as maximum wallclock time, rather than perhaps more scientifically-significant definitions of “the end of a simulation run”)

new_pcoord_array(pcoord_len=None)

Return an appropriately-sized and -typed pcoord array for a timepoint, segment, or number of segments. If pcoord_len is not specified (or None), then a length appropriate for a segment is returned.

new_region_set()**6.2.6.10 westpa.core.textio module**

Miscellaneous routines to help with input and output of WEST-related data in text format

class westpa.core.textio.**NumericTextOutputFormatter**(*output_file, mode='wt', emit_header=None*)

Bases: object

comment_string = '# '

emit_header = True

close()

write(str)

writelines(sequence)

write_comment(line)

Writes a line beginning with the comment string

write_header(line)

Appends a line to those written when the file header is written. The appropriate comment string will be prepended, so line should not include a comment character.

6.2.6.11 westpa.core.we_driver module

class westpa.core.we_driver.**Segment**(*n_iter=None, seg_id=None, weight=None, endpoint_type=None, parent_id=None, wtg_parent_ids=None, pcoord=None, status=None, walltime=None, cputime=None, data=None*)

Bases: object

A class wrapping segment data that must be passed through the work manager or data manager. Most fields are self-explanatory. One item worth noting is that a negative parent ID means that the segment starts from the initial state with ID -(segment.parent_id+1)

SEG_STATUS_UNSET = 0

SEG_STATUS_PREPARED = 1

SEG_STATUS_COMPLETE = 2

SEG_STATUS_FAILED = 3

SEG_INITPOINT_UNSET = 0

```

SEG_INITPOINT_CONTINUES = 1

SEG_INITPOINT_NEWTRAJ = 2

SEG_ENDPOINT_UNSET = 0

SEG_ENDPOINT_CONTINUES = 1

SEG_ENDPOINT_MERGED = 2

SEG_ENDPOINT_RECYCLED = 3

statuses = {'SEG_STATUS_COMPLETE': 2, 'SEG_STATUS_FAILED': 3, 'SEG_STATUS_PREPARED':
1, 'SEG_STATUS_UNSET': 0}

initpoint_types = {'SEG_INITPOINT_CONTINUES': 1, 'SEG_INITPOINT_NEWTRAJ': 2,
'SEG_INITPOINT_UNSET': 0}

endpoint_types = {'SEG_ENDPOINT_CONTINUES': 1, 'SEG_ENDPOINT_MERGED': 2,
'SEG_ENDPOINT_RECYCLED': 3, 'SEG_ENDPOINT_UNSET': 0}

status_names = {0: 'SEG_STATUS_UNSET', 1: 'SEG_STATUS_PREPARED', 2:
'SEG_STATUS_COMPLETE', 3: 'SEG_STATUS_FAILED'}

initpoint_type_names = {0: 'SEG_INITPOINT_UNSET', 1: 'SEG_INITPOINT_CONTINUES', 2:
'SEG_INITPOINT_NEWTRAJ'}

endpoint_type_names = {0: 'SEG_ENDPOINT_UNSET', 1: 'SEG_ENDPOINT_CONTINUES', 2:
'SEG_ENDPOINT_MERGED', 3: 'SEG_ENDPOINT_RECYCLED'}

static initial_pcoord(segment)
    Return the initial progress coordinate point of this segment.

static final_pcoord(segment)
    Return the final progress coordinate point of this segment.

property initpoint_type

property initial_state_id

property status_text

property endpoint_type_text

class westpa.core.we_driver.InitialState(state_id, basis_state_id, iter_created, iter_used=None,
                                         istate_type=None, istate_status=None, pcoord=None,
                                         basis_state=None, basis_auxref=None)

```

Bases: object

Describes an initial state for a new trajectory. These are generally constructed by appropriate modification of a basis state.

Variables

- **state_id** – Integer identifier of this state, usually set by the data manager.
- **basis_state_id** – Identifier of the basis state from which this state was generated, or None.
- **basis_state** – The *BasisState* from which this state was generated, or None.

- **iter_created** – Iteration in which this state was generated (0 for simulation initialization).
- **iter_used** – Iteration in which this state was used to initiate a trajectory (None for unused).
- **istate_type** – Integer describing the type of this initial state (ISTATE_TYPE_BASIS for direct use of a basis state, ISTATE_TYPE_GENERATED for a state generated from a basis state, ISTATE_TYPE_RESTART for a state corresponding to the endpoint of a segment in another simulation, or ISTATE_TYPE_START for a state generated from a start state).
- **istate_status** – Integer describing whether this initial state has been properly prepared.
- **pcoord** – The representative progress coordinate of this state.

```
ISTATE_TYPE_UNSET = 0
```

```
ISTATE_TYPE_BASIS = 1
```

```
ISTATE_TYPE_GENERATED = 2
```

```
ISTATE_TYPE_RESTART = 3
```

```
ISTATE_TYPE_START = 4
```

```
ISTATE_UNUSED = 0
```

```
ISTATE_STATUS_PENDING = 0
```

```
ISTATE_STATUS_PREPARED = 1
```

```
ISTATE_STATUS_FAILED = 2
```

```
istate_types = {'ISTATE_TYPE_BASIS': 1, 'ISTATE_TYPE_GENERATED': 2,  
'ISTATE_TYPE_RESTART': 3, 'ISTATE_TYPE_START': 4, 'ISTATE_TYPE_UNSET': 0}
```

```
istate_type_names = {0: 'ISTATE_TYPE_UNSET', 1: 'ISTATE_TYPE_BASIS', 2:  
'ISTATE_TYPE_GENERATED', 3: 'ISTATE_TYPE_RESTART', 4: 'ISTATE_TYPE_START'}
```

```
istate_statuses = {'ISTATE_STATUS_FAILED': 2, 'ISTATE_STATUS_PENDING': 0,  
'ISTATE_STATUS_PREPARED': 1}
```

```
istate_status_names = {0: 'ISTATE_STATUS_PENDING', 1: 'ISTATE_STATUS_PREPARED', 2:  
'ISTATE_STATUS_FAILED'}
```

```
as_numpy_record()
```

```
exception westpa.core.we_driver.ConsistencyError
```

```
Bases: RuntimeError
```

```
exception westpa.core.we_driver.AccuracyError
```

```
Bases: RuntimeError
```

```
class westpa.core.we_driver.NewWeightEntry(source_type, weight, prev_seg_id=None,  
                                           prev_init_pcoord=None, prev_final_pcoord=None,  
                                           new_init_pcoord=None, target_state_id=None,  
                                           initial_state_id=None)
```

```
Bases: object
```


NW_SOURCE_RECYCLED = 0

class westpa.core.we_driver.WEDriver(*rc=None, system=None*)

Bases: object

A class implemented Huber & Kim's weighted ensemble algorithm over Segment objects. This class handles all binning, recycling, and preparation of new Segment objects for the next iteration. Binning is accomplished using `system.bin_mapper`, and per-bin target counts are from `system.bin_target_counts`.

The workflow is as follows:

- 1) Call `new_iteration()` every new iteration, providing any recycling targets that are in force and any available initial states for recycling.
- 2) Call `assign()` to assign segments to bins based on their initial and end points. This returns the number of walkers that were recycled.
- 3) Call `run_we()`, optionally providing a set of initial states that will be used to recycle walkers.

Note the presence of `flux_matrix`, `transition_matrix`, `current_iter_segments`, `next_iter_segments`, `recycling_segments`, `initial_binning`, `final_binning`, `next_iter_binning`, and `new_weights` (to be documented soon).

weight_split_threshold = 2.0

weight_merge_cutoff = 1.0

largest_allowed_weight = 1.0

smallest_allowed_weight = 1e-310

process_config()

property next_iter_segments

Newly-created segments for the next iteration

property current_iter_segments

Segments for the current iteration

property next_iter_assignments

Bin assignments (indices) for initial points of next iteration.

property current_iter_assignments

Bin assignments (indices) for endpoints of current iteration.

property recycling_segments

Segments designated for recycling

property n_recycled_segs

Number of segments recycled this iteration

property n_istates_needed

Number of initial states needed to support recycling for this iteration

check_threshold_configs()

Check to see if weight thresholds parameters are valid

clear()

Explicitly delete all Segment-related state.

new_iteration(*initial_states=None, target_states=None, new_weights=None, bin_mapper=None, bin_target_counts=None*)

Prepare for a new iteration. *initial_states* is a sequence of all *InitialState* objects valid for use in to generating new segments for the *next* iteration (after the one being begun with the call to *new_iteration*); that is, these are states available to recycle to. Target states which generate recycling events are specified in *target_states*, a sequence of *TargetState* objects. Both *initial_states* and *target_states* may be empty as required.

The optional *new_weights* is a sequence of *NewWeightEntry* objects which will be used to construct the initial flux matrix.

The given *bin_mapper* will be used for assignment, and *bin_target_counts* used for splitting/merging target counts; each will be obtained from the system object if omitted or *None*.

add_initial_states(*initial_states*)

Add newly-prepared initial states to the pool available for recycling.

property all_initial_states

Return an iterator over all initial states (available or used)

assign(*segments, initializing=False*)

Assign segments to initial and final bins, and update the (internal) lists of used and available initial states. If *initializing* is *True*, then the “final” bin assignments will be identical to the initial bin assignments, a condition required for seeding a new iteration from pre-existing segments.

populate_initial(*initial_states, weights, system=None*)

Create walkers for a new weighted ensemble simulation.

One segment is created for each provided initial state, then binned and split/merged as necessary. After this function is called, *next_iter_segments* will yield the new segments to create, *used_initial_states* will contain data about which of the provided initial states were used, and *avail_initial_states* will contain data about which initial states were unused (because their corresponding walkers were merged out of existence).

rebin_current(*parent_segments*)

Reconstruct walkers for the current iteration based on (presumably) new binning. The previous iteration’s segments must be provided (as *parent_segments*) in order to update endpoint types appropriately.

construct_next()

Construct walkers for the next iteration, by running weighted ensemble recycling and bin/split/merge on the segments previously assigned to bins using *assign*. Enough unused initial states must be present in *self.avail_initial_states* for every recycled walker to be assigned an initial state.

After this function completes, *self.flux_matrix* contains a valid flux matrix for this iteration (including any contributions from recycling from the previous iteration), and *self.next_iter_segments* contains a list of segments ready for the next iteration, with appropriate values set for weight, endpoint type, parent walkers, and so on.

6.2.6.12 westpa.core.wm_ops module

westpa.core.wm_ops.get_pcoord(*state*)

westpa.core.wm_ops.gen_istate(*basis_state, initial_state*)

westpa.core.wm_ops.prep_iter(*n_iter, segments*)

westpa.core.wm_ops.post_iter(*n_iter, segments*)

westpa.core.wm_ops.propagate(*basis_states, initial_states, segments*)

6.2.6.13 westpa.core.yamlcfg module

YAML-based configuration files for WESTPA

`westpa.core.yamlcfg.YLoader`

alias of `CLoader`

class `westpa.core.yamlcfg.NopMapper`

Bases: `BinMapper`

Put everything into one bin.

assign(*coords*, *mask=None*, *output=None*)

exception `westpa.core.yamlcfg.ConfigValueWarning`

Bases: `UserWarning`

`westpa.core.yamlcfg.warn_dubious_config_entry`(*entry*, *value*, *expected_type=None*, *category=<class 'westpa.core.yamlcfg.ConfigValueWarning'>*, *stacklevel=1*)

`westpa.core.yamlcfg.check_bool`(*value*, *action='warn'*)

Check that the given value is boolean in type. If not, either raise a warning (if *action*== 'warn') or an exception (if *action*== 'raise').

exception `westpa.core.yamlcfg.ConfigItemMissing`(*key*, *message=None*)

Bases: `KeyError`

exception `westpa.core.yamlcfg.ConfigItemTypeError`(*key*, *expected_type*, *message=None*)

Bases: `TypeError`

exception `westpa.core.yamlcfg.ConfigValueError`(*key*, *value*, *message=None*)

Bases: `ValueError`

class `westpa.core.yamlcfg.YAMLConfig`

Bases: `object`

preload_config_files = `['/etc/westpa/westrc', '/home/docs/.westrc']`

update_from_file(*file*, *required=True*)

require(*key*, *type_=None*)

Ensure that a configuration item with the given *key* is present. If the optional *type_* is given, additionally require that the item has that type.

require_type_if_present(*key*, *type_*)

Ensure that the configuration item with the given *key* has the given type.

coerce_type_if_present(*key*, *type_*)

get(*key*, *default=None*)

get_typed(*key*, *type_*, *default=<object object>*)

get_path(*key*, *default=<object object>*, *expandvars=True*, *expanduser=True*, *realpath=True*, *abspath=True*)

get_pathlist(*key*, *default=<object object>*, *sep=':'*, *expandvars=True*, *expanduser=True*, *realpath=True*, *abspath=True*)

`get_python_object(key, default=<object object>, path=None)`

`get_choice(key, choices, default=<object object>, value_transform=None)`

`class westpa.core.yamlcfg.YAMLSystem(rc=None)`

Bases: `object`

A description of the system being simulated, including the dimensionality and data type of the progress coordinate, the number of progress coordinate entries expected from each segment, and binning. To construct a simulation, the user must subclass `WESTSystem` and set several instance variables.

At a minimum, the user must subclass `WESTSystem` and override **:method: `initialize`** to set the data type and dimensionality of progress coordinate data and define a bin mapper.

Variables

- **pcoord_ndim** – The number of dimensions in the progress coordinate. Defaults to 1 (i.e. a one-dimensional progress coordinate).
- **pcoord_dtype** – The data type of the progress coordinate, which must be callable (e.g. `np.float32` and `long` will work, but `'<f4'` and `'<i8'` will not). Defaults to `np.float64`.
- **pcoord_len** – The length of the progress coordinate time series generated by each segment, including *both* the initial and final values. Defaults to 2 (i.e. only the initial and final progress coordinate values for a segment are returned from propagation).
- **bin_mapper** – A bin mapper describing the progress coordinate space.
- **bin_target_counts** – A vector of target counts, one per bin.

property bin_target_counts

initialize()

Prepare this system object for use in simulation or analysis, creating a bin space, setting replicas per bin, and so on. This function is called whenever a WEST tool creates an instance of the system driver.

prepare_run()

Prepare this system for use in a simulation run. Called by `w_run` in all worker processes.

finalize_run()

A hook for system-specific processing for the end of a simulation run (as defined by such things as maximum wallclock time, rather than perhaps more scientifically-significant definitions of “the end of a simulation run”)

new_pcoord_array(pcoord_len=None)

Return an appropriately-sized and -typed pcoord array for a timepoint, segment, or number of segments. If `pcoord_len` is not specified (or `None`), then a length appropriate for a segment is returned.

new_region_set()

6.3 westpa.work_managers package

6.3.1 westpa.work_managers package

6.3.1.1 westpa.work_managers module

A system for parallel, remote execution of multiple arbitrary tasks. Much of this, both in concept and execution, was inspired by (and in some cases based heavily on) the `concurrent.futures` package from Python 3.2, with some simplifications and adaptations (thanks to Brian Quinlan and his futures implementation).

class `westpa.work_managers.SerialWorkManager`

Bases: [`WorkManager`](#)

classmethod `from_environ(wmenv=None)`

submit(*fn*, *args=None*, *kwargs=None*)

Submit a task to the work manager, returning a *WMFuture* object representing the pending result. *fn(*args, **kwargs)* will be executed by a worker, and the return value assigned as the result of the returned future. The function *fn* and all arguments must be picklable; note particularly that off-path modules (like the system module and any active plugins) are not picklable unless pre-loaded in the worker process (i.e. prior to forking the master).

class `westpa.work_managers.ThreadsWorkManager(n_workers=None)`

Bases: [`WorkManager`](#)

A work manager using threads.

classmethod `from_environ(wmenv=None)`

runtask(*task_queue*)

submit(*fn*, *args=None*, *kwargs=None*)

Submit a task to the work manager, returning a *WMFuture* object representing the pending result. *fn(*args, **kwargs)* will be executed by a worker, and the return value assigned as the result of the returned future. The function *fn* and all arguments must be picklable; note particularly that off-path modules (like the system module and any active plugins) are not picklable unless pre-loaded in the worker process (i.e. prior to forking the master).

startup()

Perform any necessary startup work, such as spawning clients.

shutdown()

Cleanly shut down any active workers.

class `westpa.work_managers.ProcessWorkManager(n_workers=None, shutdown_timeout=1)`

Bases: [`WorkManager`](#)

A work manager using the `multiprocessing` module.

Notes

On MacOS, as of Python 3.8 the default start method for multiprocessing launching new processes was changed from fork to spawn. In general, spawn is more robust and efficient, however it requires serializability of everything being passed to the child process. In contrast, fork is much less memory efficient, as it makes a full copy of everything in the parent process. However, it does not require picklability.

So, on MacOS, the method for launching new processes is explicitly changed to fork from the (MacOS-specific) default of spawn. Unix should default to fork.

See <https://docs.python.org/3/library/multiprocessing.html#contexts-and-start-methods> and <https://docs.python.org/3/library/multiprocessing.html#the-spawn-and-forkserver-start-methods> for more details.

classmethod `from_environ(wmenv=None)`

task_loop()

results_loop()

submit(*fn*, *args=None*, *kwargs=None*)

Submit a task to the work manager, returning a *WMFuture* object representing the pending result. `fn(*args, **kwargs)` will be executed by a worker, and the return value assigned as the result of the returned future. The function `fn` and all arguments must be picklable; note particularly that off-path modules (like the system module and any active plugins) are not picklable unless pre-loaded in the worker process (i.e. prior to forking the master).

startup()

Perform any necessary startup work, such as spawning clients.

shutdown()

Cleanly shut down any active workers.

`westpa.work_managers.make_work_manager()`

Using cues from the environment, instantiate a pre-configured work manager.

6.3.1.2 `westpa.work_managers.core` module

class `westpa.work_managers.core.islice`

Bases: `object`

`islice(iterable, stop) -> islice object` `islice(iterable, start, stop[, step]) -> islice object`

Return an iterator whose `next()` method returns selected values from an iterable. If `start` is specified, will skip all preceding elements; otherwise, `start` defaults to zero. `Step` defaults to one. If specified as another value, `step` determines how many values are skipped between successive calls. Works like a `slice()` on a list but returns an iterator.

`westpa.work_managers.core.contextmanager(func)`

@contextmanager decorator.

Typical usage:

```
@contextmanager def some_generator(<arguments>):
    <setup> try:
        yield <value>

    finally:
        <cleanup>
```

This makes this:

```
with some_generator(<arguments>) as <variable>:
    <body>
```

equivalent to this:

```
<setup> try:
    <variable> = <value> <body>

finally:
    <cleanup>
```

class westpa.work_managers.core.**WorkManager**

Bases: object

Base class for all work managers. At a minimum, work managers must provide a `submit()` function and a `n_workers` attribute (which may be a property), though most will also override `startup()` and `shutdown()`.

classmethod `from_environ(wmenv=None)`

classmethod `add_wm_args(parser, wmenv=None)`

sigint_handler(*signum, frame*)

install_sigint_handler()

startup()

Perform any necessary startup work, such as spawning clients.

shutdown()

Cleanly shut down any active workers.

run()

Run the worker loop (in clients only).

submit(*fn, args=None, kwargs=None*)

Submit a task to the work manager, returning a *WMFuture* object representing the pending result. `fn(*args, **kwargs)` will be executed by a worker, and the return value assigned as the result of the returned future. The function `fn` and all arguments must be picklable; note particularly that off-path modules (like the system module and any active plugins) are not picklable unless pre-loaded in the worker process (i.e. prior to forking the master).

submit_many(*tasks*)

Submit a set of tasks to the work manager, returning a list of *WMFuture* objects representing pending results. Each entry in `tasks` should be a triple (`fn, args, kwargs`), which will result in `fn(*args, **kwargs)` being executed by a worker. The function `fn` and all arguments must be picklable; note particularly that off-path modules are not picklable unless pre-loaded in the worker process.

as_completed(*futures*)

Return a generator which yields results from the given `futures` as they become available.

submit_as_completed(*task_generator, queue_size=None*)

Return a generator which yields results from a set of `futures` as they become available. Futures are generated by the `task_generator`, which must return a triple of the form expected by `submit`. The method also accepts an int `queue_size` that dictates the maximum number of Futures that should be pending at any given time. The default value of `None` submits all of the tasks at once.

wait_any(*futures*)

Wait on any of the given `futures` and return the first one which has a result available. If more than one result is or becomes available simultaneously, any completed future may be returned.

wait_all(*futures*)

A convenience function which waits on all the given *futures* in order. This function returns the same *futures* as submitted to the function as a list, indicating the order in which waits occurred.

property is_master

True if this is the master process for task distribution. This is necessary, e.g., for MPI, where all processes start identically and then must branch depending on rank.

class westpa.work_managers.core.**FutureWatcher**(*futures*, *threshold=1*)

Bases: object

A device to wait on multiple results and/or exceptions with only one lock.

signal(*future*)

Signal this watcher that the given future has results available. If this brings the number of available futures above *signal_threshold*, this watcher's event object will be signalled as well.

wait()

Wait on one or more futures.

reset()

Reset this watcher's list of completed futures, returning the list of completed futures prior to resetting it.

add(*futures*)

Add watchers to all futures in the iterable of futures.

class westpa.work_managers.core.**WMFuture**(*task_id=None*)

Bases: object

A "future", representing work which has been dispatched for completion asynchronously.

static all_acquired(*futures*)

Context manager to acquire all locks on the given *futures*. Primarily for internal use.

get_result(*discard=True*)

Get the result associated with this future, blocking until it is available. If *discard* is true, then removes the reference to the result contained in this instance, so that a collection of futures need not turn into a cache of all associated results.

property result

wait()

Wait until this future has a result or exception available.

get_exception()

Get the exception associated with this future, blocking until it is available.

property exception

Get the exception associated with this future, blocking until it is available.

get_traceback()

Get the traceback object associated with this future, if any.

property traceback

Get the traceback object associated with this future, if any.

is_done()

Indicates whether this future is done executing (may block if this future is being updated).

property done

Indicates whether this future is done executing (may block if this future is being updated).

6.3.1.3 westpa.work_managers.environment module

Routines for configuring the work manager environment

```
class westpa.work_managers.environment.WMEnvironment(use_arg_prefixes=False,  
                                                    valid_work_managers=None)
```

Bases: object

A class to encapsulate the environment in which work managers are instantiated; this controls how environment variables and command-line arguments are used to set up work managers. This could be used to cleanly instantiate two work managers within one application, but is really more about providing facilities to make it easier for individual work managers to configure themselves according to precedence of configuration information:

1. command-line arguments
2. environment variables
3. defaults

```
group_title = 'parallelization options'
```

```
group_description = None
```

```
env_prefix = 'WM'
```

```
arg_prefix = 'wm'
```

```
default_work_manager = 'serial'
```

```
default_parallel_work_manager = 'processes'
```

```
valid_work_managers = ['serial', 'threads', 'processes', 'zmq', 'mpi']
```

```
env_name(name)
```

```
arg_name(name)
```

```
arg_flag(name)
```

```
get_val(name, default=None, type_=None)
```

```
add_wm_args(parser)
```

```
process_wm_args(args)
```

```
make_work_manager()
```

Using cues from the environment, instantiate a pre-configured work manager.

```
westpa.work_managers.environment.make_work_manager()
```

Using cues from the environment, instantiate a pre-configured work manager.

```
westpa.work_managers.environment.add_wm_args(parser)
```

```
westpa.work_managers.environment.process_wm_args(args)
```

6.3.1.4 westpa.work_managers.mpi module

A work manager which uses MPI to distribute tasks and collect results.

class westpa.work_managers.mpi.deque

Bases: object

deque([iterable[, maxlen]]) → deque object

A list-like sequence optimized for data accesses near its endpoints.

append()

Add an element to the right side of the deque.

appendleft()

Add an element to the left side of the deque.

clear()

Remove all elements from the deque.

copy()

Return a shallow copy of a deque.

count()

D.count(value) – return number of occurrences of value

extend()

Extend the right side of the deque with elements from the iterable

extendleft()

Extend the left side of the deque with elements from the iterable

index()

D.index(value, [start, [stop]]) – return first index of value. Raises ValueError if the value is not present.

insert()

D.insert(index, object) – insert object before index

maxlen

maximum size of a deque or None if unbounded

pop()

Remove and return the rightmost element.

popleft()

Remove and return the leftmost element.

remove()

D.remove(value) – remove first occurrence of value.

reverse()

D.reverse() – reverse *IN PLACE*

rotate()

Rotate the deque n steps to the right (default n=1). If n is negative, rotates left.

class westpa.work_managers.mpi.WorkManager

Bases: object

Base class for all work managers. At a minimum, work managers must provide a `submit()` function and a `n_workers` attribute (which may be a property), though most will also override `startup()` and `shutdown()`.

classmethod `from_environ(wmenv=None)`

classmethod `add_wm_args(parser, wmenv=None)`

sigint_handler(*signum, frame*)

install_sigint_handler()

startup()

Perform any necessary startup work, such as spawning clients.

shutdown()

Cleanly shut down any active workers.

run()

Run the worker loop (in clients only).

submit(*fn, args=None, kwargs=None*)

Submit a task to the work manager, returning a *WMFuture* object representing the pending result. `fn(*args, **kwargs)` will be executed by a worker, and the return value assigned as the result of the returned future. The function `fn` and all arguments must be picklable; note particularly that off-path modules (like the system module and any active plugins) are not picklable unless pre-loaded in the worker process (i.e. prior to forking the master).

submit_many(*tasks*)

Submit a set of tasks to the work manager, returning a list of *WMFuture* objects representing pending results. Each entry in `tasks` should be a triple (`fn, args, kwargs`), which will result in `fn(*args, **kwargs)` being executed by a worker. The function `fn` and all arguments must be picklable; note particularly that off-path modules are not picklable unless pre-loaded in the worker process.

as_completed(*futures*)

Return a generator which yields results from the given `futures` as they become available.

submit_as_completed(*task_generator, queue_size=None*)

Return a generator which yields results from a set of `futures` as they become available. Futures are generated by the `task_generator`, which must return a triple of the form expected by `submit`. The method also accepts an int `queue_size` that dictates the maximum number of Futures that should be pending at any given time. The default value of `None` submits all of the tasks at once.

wait_any(*futures*)

Wait on any of the given `futures` and return the first one which has a result available. If more than one result is or becomes available simultaneously, any completed future may be returned.

wait_all(*futures*)

A convenience function which waits on all the given `futures` in order. This function returns the same `futures` as submitted to the function as a list, indicating the order in which waits occurred.

property is_master

True if this is the master process for task distribution. This is necessary, e.g., for MPI, where all processes start identically and then must branch depending on rank.

class westpa.work_managers.mpi.WMFuture(*task_id=None*)

Bases: object

A “future”, representing work which has been dispatched for completion asynchronously.

static all_acquired(*futures*)

Context manager to acquire all locks on the given *futures*. Primarily for internal use.

get_result(*discard=True*)

Get the result associated with this future, blocking until it is available. If *discard* is true, then removes the reference to the result contained in this instance, so that a collection of futures need not turn into a cache of all associated results.

property result

wait()

Wait until this future has a result or exception available.

get_exception()

Get the exception associated with this future, blocking until it is available.

property exception

Get the exception associated with this future, blocking until it is available.

get_traceback()

Get the traceback object associated with this future, if any.

property traceback

Get the traceback object associated with this future, if any.

is_done()

Indicates whether this future is done executing (may block if this future is being updated).

property done

Indicates whether this future is done executing (may block if this future is being updated).

class westpa.work_managers.mpi.Task(*task_id, fn, args, kwargs*)

Bases: object

Tasks are tuples of (task_id, function, args, keyword args)

class westpa.work_managers.mpi.MPIWorkManager

Bases: [WorkManager](#)

MPIWorkManager factory.

Initialize info shared by Manager and Worker classes.

classmethod from_env(*wmenv=None*)

submit(*fn, args=None, kwargs=None*)

Adhere to WorkManager interface. This method should never be called.

class westpa.work_managers.mpi.Serial

Bases: [MPIWorkManager](#)

Replication of the serial work manager. This is a fallback for MPI runs that request only 1 (size=1) processor.

Initialize info shared by Manager and Worker classes.

submit(*fn*, *args=None*, *kwargs=None*)

Adhere to WorkManager interface. This method should never be called.

class westpa.work_managers.mpi.**Manager**

Bases: *MPIWorkManager*

Manager of the MPIWorkManage. Distributes tasks to Worker as they are received from the sim_manager. In addition to the main thread, this class spawns two threads, a receiver and a dispatcher.

Initialize different state variables used by Manager.

startup()

Spawns the dispatcher and receiver threads.

submit(*fn*, *args=None*, *kwargs=None*)

Receive task from simulation manager and add it to pending_futures.

shutdown()

Send shutdown tag to all worker processes, and set the shutdown sentinel to stop the receiver and dispatcher loops.

class westpa.work_managers.mpi.**Worker**

Bases: *MPIWorkManager*

Client class for executing tasks as distributed by the Manager in the MPI Work Manager

Initialize info shared by Manager and Worker classes.

startup()

Clock the worker in for work.

clockIn()

Do each task as it comes in. The completion of a task is notice to the manager that more work is welcome.

property is_master

Worker processes need to be marked as not manager. This ensures that the proper branching is followed in w_run.py.

6.3.1.5 westpa.work_managers.processes module

class westpa.work_managers.processes.**WorkManager**

Bases: object

Base class for all work managers. At a minimum, work managers must provide a **submit**() function and a **n_workers** attribute (which may be a property), though most will also override **startup**() and **shutdown**().

classmethod from_environ(*wmenv=None*)

classmethod add_wm_args(*parser*, *wmenv=None*)

sigint_handler(*signal*, *frame*)

install_sigint_handler()

startup()

Perform any necessary startup work, such as spawning clients.

shutdown()

Cleanly shut down any active workers.

run()

Run the worker loop (in clients only).

submit(*fn*, *args=None*, *kwargs=None*)

Submit a task to the work manager, returning a *WMFuture* object representing the pending result. *fn(*args, **kwargs)* will be executed by a worker, and the return value assigned as the result of the returned future. The function *fn* and all arguments must be picklable; note particularly that off-path modules (like the system module and any active plugins) are not picklable unless pre-loaded in the worker process (i.e. prior to forking the master).

submit_many(*tasks*)

Submit a set of tasks to the work manager, returning a list of *WMFuture* objects representing pending results. Each entry in *tasks* should be a triple (*fn*, *args*, *kwargs*), which will result in *fn(*args, **kwargs)* being executed by a worker. The function *fn* and all arguments must be picklable; note particularly that off-path modules are not picklable unless pre-loaded in the worker process.

as_completed(*futures*)

Return a generator which yields results from the given *futures* as they become available.

submit_as_completed(*task_generator*, *queue_size=None*)

Return a generator which yields results from a set of *futures* as they become available. Futures are generated by the *task_generator*, which must return a triple of the form expected by *submit*. The method also accepts an int *queue_size* that dictates the maximum number of Futures that should be pending at any given time. The default value of *None* submits all of the tasks at once.

wait_any(*futures*)

Wait on any of the given *futures* and return the first one which has a result available. If more than one result is or becomes available simultaneously, any completed future may be returned.

wait_all(*futures*)

A convenience function which waits on all the given *futures* in order. This function returns the same *futures* as submitted to the function as a list, indicating the order in which waits occurred.

property is_master

True if this is the master process for task distribution. This is necessary, e.g., for MPI, where all processes start identically and then must branch depending on rank.

class westpa.work_managers.processes.**WMFuture**(*task_id=None*)

Bases: object

A “future”, representing work which has been dispatched for completion asynchronously.

static all_acquired(*futures*)

Context manager to acquire all locks on the given *futures*. Primarily for internal use.

get_result(*discard=True*)

Get the result associated with this future, blocking until it is available. If *discard* is true, then removes the reference to the result contained in this instance, so that a collection of futures need not turn into a cache of all associated results.

property result**wait**()

Wait until this future has a result or exception available.

get_exception()

Get the exception associated with this future, blocking until it is available.

property exception

Get the exception associated with this future, blocking until it is available.

get_traceback()

Get the traceback object associated with this future, if any.

property traceback

Get the traceback object associated with this future, if any.

is_done()

Indicates whether this future is done executing (may block if this future is being updated).

property done

Indicates whether this future is done executing (may block if this future is being updated).

class westpa.work_managers.processes.**ProcessWorkManager**(*n_workers=None, shutdown_timeout=1*)

Bases: *WorkManager*

A work manager using the multiprocessing module.

Notes

On MacOS, as of Python 3.8 the default start method for multiprocessing launching new processes was changed from fork to spawn. In general, spawn is more robust and efficient, however it requires serializability of everything being passed to the child process. In contrast, fork is much less memory efficient, as it makes a full copy of everything in the parent process. However, it does not require picklability.

So, on MacOS, the method for launching new processes is explicitly changed to fork from the (MacOS-specific) default of spawn. Unix should default to fork.

See <https://docs.python.org/3/library/multiprocessing.html#contexts-and-start-methods> and <https://docs.python.org/3/library/multiprocessing.html#the-spawn-and-forkserver-start-methods> for more details.

classmethod **from_environ**(*wmenv=None*)

task_loop()**results_loop()**

submit(*fn, args=None, kwargs=None*)

Submit a task to the work manager, returning a *WMFuture* object representing the pending result. *fn(*args, **kwargs)* will be executed by a worker, and the return value assigned as the result of the returned future. The function *fn* and all arguments must be picklable; note particularly that off-path modules (like the system module and any active plugins) are not picklable unless pre-loaded in the worker process (i.e. prior to forking the master).

startup()

Perform any necessary startup work, such as spawning clients.

shutdown()

Cleanly shut down any active workers.

6.3.1.6 westpa.work_managers.serial module

class westpa.work_managers.serial.**WorkManager**

Bases: object

Base class for all work managers. At a minimum, work managers must provide a `submit()` function and a `n_workers` attribute (which may be a property), though most will also override `startup()` and `shutdown()`.

classmethod `from_environ(wmenv=None)`

classmethod `add_wm_args(parser, wmenv=None)`

sigint_handler(*signum, frame*)

install_sigint_handler()

startup()

Perform any necessary startup work, such as spawning clients.

shutdown()

Cleanly shut down any active workers.

run()

Run the worker loop (in clients only).

submit(*fn, args=None, kwargs=None*)

Submit a task to the work manager, returning a *WMFuture* object representing the pending result. `fn(*args, **kwargs)` will be executed by a worker, and the return value assigned as the result of the returned future. The function `fn` and all arguments must be pickleable; note particularly that off-path modules (like the system module and any active plugins) are not pickleable unless pre-loaded in the worker process (i.e. prior to forking the master).

submit_many(*tasks*)

Submit a set of tasks to the work manager, returning a list of *WMFuture* objects representing pending results. Each entry in `tasks` should be a triple (`fn, args, kwargs`), which will result in `fn(*args, **kwargs)` being executed by a worker. The function `fn` and all arguments must be pickleable; note particularly that off-path modules are not pickleable unless pre-loaded in the worker process.

as_completed(*futures*)

Return a generator which yields results from the given `futures` as they become available.

submit_as_completed(*task_generator, queue_size=None*)

Return a generator which yields results from a set of `futures` as they become available. Futures are generated by the `task_generator`, which must return a triple of the form expected by `submit`. The method also accepts an int `queue_size` that dictates the maximum number of Futures that should be pending at any given time. The default value of `None` submits all of the tasks at once.

wait_any(*futures*)

Wait on any of the given `futures` and return the first one which has a result available. If more than one result is or becomes available simultaneously, any completed future may be returned.

wait_all(*futures*)

A convenience function which waits on all the given `futures` in order. This function returns the same `futures` as submitted to the function as a list, indicating the order in which waits occurred.

property `is_master`

True if this is the master process for task distribution. This is necessary, e.g., for MPI, where all processes start identically and then must branch depending on rank.

class westpa.work_managers.serial.**WMFuture**(*task_id=None*)

Bases: `object`

A “future”, representing work which has been dispatched for completion asynchronously.

static all_acquired(*futures*)

Context manager to acquire all locks on the given *futures*. Primarily for internal use.

get_result(*discard=True*)

Get the result associated with this future, blocking until it is available. If *discard* is true, then removes the reference to the result contained in this instance, so that a collection of futures need not turn into a cache of all associated results.

property result

wait()

Wait until this future has a result or exception available.

get_exception()

Get the exception associated with this future, blocking until it is available.

property exception

Get the exception associated with this future, blocking until it is available.

get_traceback()

Get the traceback object associated with this future, if any.

property traceback

Get the traceback object associated with this future, if any.

is_done()

Indicates whether this future is done executing (may block if this future is being updated).

property done

Indicates whether this future is done executing (may block if this future is being updated).

class westpa.work_managers.serial.**SerialWorkManager**

Bases: `WorkManager`

classmethod from_environ(*wmenv=None*)

submit(*fn, args=None, kwargs=None*)

Submit a task to the work manager, returning a *WMFuture* object representing the pending result. *fn(*args, **kwargs)* will be executed by a worker, and the return value assigned as the result of the returned future. The function *fn* and all arguments must be picklable; note particularly that off-path modules (like the system module and any active plugins) are not picklable unless pre-loaded in the worker process (i.e. prior to forking the master).

6.3.1.7 westpa.work_managers.threads module

class westpa.work_managers.threads.**WorkManager**

Bases: object

Base class for all work managers. At a minimum, work managers must provide a `submit()` function and a `n_workers` attribute (which may be a property), though most will also override `startup()` and `shutdown()`.

classmethod `from_environ(wmenv=None)`

classmethod `add_wm_args(parser, wmenv=None)`

sigint_handler(*signum, frame*)

install_sigint_handler()

startup()

Perform any necessary startup work, such as spawning clients.

shutdown()

Cleanly shut down any active workers.

run()

Run the worker loop (in clients only).

submit(*fn, args=None, kwargs=None*)

Submit a task to the work manager, returning a *WMFuture* object representing the pending result. `fn(*args, **kwargs)` will be executed by a worker, and the return value assigned as the result of the returned future. The function `fn` and all arguments must be pickleable; note particularly that off-path modules (like the system module and any active plugins) are not pickleable unless pre-loaded in the worker process (i.e. prior to forking the master).

submit_many(*tasks*)

Submit a set of tasks to the work manager, returning a list of *WMFuture* objects representing pending results. Each entry in `tasks` should be a triple (`fn, args, kwargs`), which will result in `fn(*args, **kwargs)` being executed by a worker. The function `fn` and all arguments must be pickleable; note particularly that off-path modules are not pickleable unless pre-loaded in the worker process.

as_completed(*futures*)

Return a generator which yields results from the given `futures` as they become available.

submit_as_completed(*task_generator, queue_size=None*)

Return a generator which yields results from a set of `futures` as they become available. Futures are generated by the `task_generator`, which must return a triple of the form expected by `submit`. The method also accepts an int `queue_size` that dictates the maximum number of Futures that should be pending at any given time. The default value of `None` submits all of the tasks at once.

wait_any(*futures*)

Wait on any of the given `futures` and return the first one which has a result available. If more than one result is or becomes available simultaneously, any completed future may be returned.

wait_all(*futures*)

A convenience function which waits on all the given `futures` in order. This function returns the same `futures` as submitted to the function as a list, indicating the order in which waits occurred.

property `is_master`

True if this is the master process for task distribution. This is necessary, e.g., for MPI, where all processes start identically and then must branch depending on rank.

class westpa.work_managers.threads.**WMFuture**(*task_id=None*)

Bases: object

A “future”, representing work which has been dispatched for completion asynchronously.

static all_acquired(*futures*)

Context manager to acquire all locks on the given *futures*. Primarily for internal use.

get_result(*discard=True*)

Get the result associated with this future, blocking until it is available. If *discard* is true, then removes the reference to the result contained in this instance, so that a collection of futures need not turn into a cache of all associated results.

property result

wait()

Wait until this future has a result or exception available.

get_exception()

Get the exception associated with this future, blocking until it is available.

property exception

Get the exception associated with this future, blocking until it is available.

get_traceback()

Get the traceback object associated with this future, if any.

property traceback

Get the traceback object associated with this future, if any.

is_done()

Indicates whether this future is done executing (may block if this future is being updated).

property done

Indicates whether this future is done executing (may block if this future is being updated).

class westpa.work_managers.threads.**Task**(*fn, args, kwargs, future*)

Bases: object

run()

class westpa.work_managers.threads.**ThreadsWorkManager**(*n_workers=None*)

Bases: [WorkManager](#)

A work manager using threads.

classmethod from_environ(*wmenv=None*)

runtask(*task_queue*)

submit(*fn, args=None, kwargs=None*)

Submit a task to the work manager, returning a *WMFuture* object representing the pending result. *fn(*args, **kwargs)* will be executed by a worker, and the return value assigned as the result of the returned future. The function *fn* and all arguments must be picklable; note particularly that off-path modules (like the system module and any active plugins) are not picklable unless pre-loaded in the worker process (i.e. prior to forking the master).

startup()

Perform any necessary startup work, such as spawning clients.

shutdown()

Cleanly shut down any active workers.

6.3.2 westpa.work_managers.zeromq package

6.3.2.1 westpa.work_managers.zeromq module

exception westpa.work_managers.zeromq.ZMQWMEError

Bases: RuntimeError

Base class for errors related to the ZeroMQ work manager itself

exception westpa.work_managers.zeromq.ZMQWMTIMEOUT

Bases: ZMQWMEEnvironmentError

A timeout of a sort that indicates that a master or worker has failed or never started.

exception westpa.work_managers.zeromq.ZMQWMEEnvironmentError

Bases: ZMQWMEError

Class representing an error in the environment in which the ZeroMQ work manager is running. This includes such things as master/worker ID mismatches.

exception westpa.work_managers.zeromq.ZMQWorkerMissing

Bases: ZMQWMEError

Exception representing that a worker processing a task died or disappeared

class westpa.work_managers.zeromq.ZMQCore

Bases: object

PROTOCOL_MAJOR = 3

PROTOCOL_MINOR = 0

PROTOCOL_UPDATE = 0

PROTOCOL_VERSION = (3, 0, 0)

internal_transport = 'ipc'

default_comm_mode = 'ipc'

default_master_heartbeat = 20.0

default_worker_heartbeat = 20.0

default_timeout_factor = 5.0

default_startup_timeout = 120.0

default_shutdown_timeout = 5.0

classmethod make_ipc_endpoint()

classmethod remove_ipc_endpoints()

classmethod make_tcp_endpoint(address='127.0.0.1')

classmethod `make_internal_endpoint()`

get_identification()

validate_message(*message*)

Validate incoming message. Raises an exception if the message is improperly formatted (TypeError) or does not correspond to the appropriate master (ZMQWMEEnvironmentError).

message_validation(*msg*)

A context manager for message validation. The instance variable `validation_fail_action` controls the behavior of this context manager:

- ‘raise’: re-raise the exception that indicated failed validation. Useful for development.
- ‘exit’ (default): report the error and exit the program.
- ‘warn’: report the error and continue.

recv_message(*socket, flags=0, validate=True, timeout=None*)

Receive a message object from the given socket, using the given flags. Message validation is performed if `validate` is true. If `timeout` is given, then it is the number of milliseconds to wait prior to raising a ZMQWMTIMEOUT exception. `timeout` is ignored if `flags` includes `zmq.NOBLOCK`.

recv_all(*socket, flags=0, validate=True*)

Receive all messages currently available from the given socket.

recv_ack(*socket, flags=0, validate=True, timeout=None*)

send_message(*socket, message, payload=None, flags=0*)

Send a message object. Subclasses may override this to decorate the message with appropriate IDs, then delegate upward to actually send the message. `message` may either be a pre-constructed Message object or a message identifier, in which (latter) case `payload` will become the message payload. `payload` is ignored if `message` is a Message object.

send_reply(*socket, original_message, reply='ok', payload=None, flags=0*)

Send a reply to `original_message` on `socket`. The reply message is a Message object or a message identifier. The reply `master_id` and `worker_id` are set from `original_message`, unless `master_id` is not set, in which case it is set from `self.master_id`.

send_ack(*socket, original_message*)

Send an acknowledgement message, which is mostly just to respect REQ/REP recv/send patterns.

send_nak(*socket, original_message*)

Send a negative acknowledgement message.

send_inproc_message(*message, payload=None, flags=0*)

signal_shutdown()

shutdown_handler(*signal=None, frame=None*)

install_signal_handlers(*signals=None*)

install_sigint_handler()

startup()

shutdown()

join()

```
class westpa.work_managers.zeromq.ZMQNode(upstream_rr_endpoint, upstream_ann_endpoint,
                                           n_local_workers=None)
```

Bases: [ZMQCore](#), [IsNode](#)

run()

property is_master

comm_loop()

startup()

```
class westpa.work_managers.zeromq.ZMQWorker(rr_endpoint, ann_endpoint)
```

Bases: [ZMQCore](#)

This is the outward facing worker component of the ZMQ work manager. This forms the interface to the master. This process cannot hang or crash due to an error in tasks it executes, so tasks are isolated in ZMQExecutor, which communicates with ZMQWorker via (what else?) ZeroMQ.

property is_master

update_master_info(msg)

identify(rr_socket)

request_task(rr_socket, task_socket)

handle_reconfigure_timeout(msg, timers)

handle_result(result_socket, rr_socket)

comm_loop()

Master communication loop for the worker process.

shutdown_executor()

install_signal_handlers(signals=None)

startup(process_index=None)

```
class westpa.work_managers.zeromq.ZMQWorkManager(n_local_workers=1)
```

Bases: [ZMQCore](#), [WorkManager](#), [IsNode](#)

classmethod add_wm_args(parser, wmenv=None)

classmethod from_environ(wmenv=None)

classmethod read_host_info(filename)

classmethod canonicalize_endpoint(endpoint, allow_wildcard_host=True)

property n_workers

submit(fn, args=None, kwargs=None)

Submit a task to the work manager, returning a *WMFuture* object representing the pending result. *fn(*args, **kwargs)* will be executed by a worker, and the return value assigned as the result of the returned future. The function *fn* and all arguments must be picklable; note particularly that off-path modules (like the system module and any active plugins) are not picklable unless pre-loaded in the worker process (i.e. prior to forking the master).

submit_many(tasks)

Submit a set of tasks to the work manager, returning a list of *WMFuture* objects representing pending results. Each entry in *tasks* should be a triple (fn, args, kwargs), which will result in fn(*args, **kwargs) being executed by a worker. The function *fn* and all arguments must be picklable; note particularly that off-path modules are not picklable unless pre-loaded in the worker process.

send_message(socket, message, payload=None, flags=0)

Send a message object. Subclasses may override this to decorate the message with appropriate IDs, then delegate upward to actually send the message. *message* may either be a pre-constructed *Message* object or a message identifier, in which (latter) case *payload* will become the message payload. *payload* is ignored if *message* is a *Message* object.

handle_result(socket, msg)**handle_task_request(socket, msg)****update_worker_information(msg)****check_workers()****remove_worker(worker_id)****shutdown_clear_tasks()**

Abort pending tasks with error on shutdown.

comm_loop()**startup()**

Perform any necessary startup work, such as spawning clients.

shutdown()

Cleanly shut down any active workers.

6.3.2.2 westpa.work_managers.zeromq.core module

Created on May 29, 2015

@author: mzwier

westpa.work_managers.zeromq.core.randport(address='127.0.0.1')

Select a random unused TCP port number on the given address.

exception westpa.work_managers.zeromq.core.ZMQWError

Bases: *RuntimeError*

Base class for errors related to the ZeroMQ work manager itself

exception westpa.work_managers.zeromq.core.ZMQWorkerMissing

Bases: *ZMQWError*

Exception representing that a worker processing a task died or disappeared

exception westpa.work_managers.zeromq.core.ZMQWEnvironmentError

Bases: *ZMQWError*

Class representing an error in the environment in which the ZeroMQ work manager is running. This includes such things as master/worker ID mismatches.

exception westpa.work_managers.zeromq.core.ZMQWTimeout

Bases: *ZMQWEnvironmentError*

A timeout of a sort that indicatess that a master or worker has failed or never started.

class westpa.work_managers.zeromq.core.**Message**(*message=None, payload=None, master_id=None, src_id=None*)

Bases: object

SHUTDOWN = 'shutdown'

ACK = 'ok'

NAK = 'no'

IDENTIFY = 'identify'

TASKS_AVAILABLE = 'tasks_available'

TASK_REQUEST = 'task_request'

MASTER_BEACON = 'master_alive'

RECONFIGURE_TIMEOUT = 'reconfigure_timeout'

TASK = 'task'

RESULT = 'result'

idempotent_announcement_messages = {'master_alive', 'shutdown', 'tasks_available'}

classmethod **coalesce_announcements**(*messages*)

class westpa.work_managers.zeromq.core.**Task**(*fn, args, kwargs, task_id=None*)

Bases: object

execute()

Run this task, returning a Result object.

class westpa.work_managers.zeromq.core.**Result**(*task_id, result=None, exception=None, traceback=None*)

Bases: object

class westpa.work_managers.zeromq.core.**PassiveTimer**(*duration, started=None*)

Bases: object

started

duration

property **expired**

property **expires_in**

reset(*at=None*)

start(*at=None*)

class westpa.work_managers.zeromq.core.**PassiveMultiTimer**

Bases: object


```
add_timer(identifier, duration)  
remove_timer(identifier)  
change_duration(identifier, duration)  
reset(identifier=None, at=None)  
expired(identifier, at=None)  
next_expiration()  
next_expiration_in()  
which_expired(at=None)
```

```
class westpa.work_managers.zeromq.core.ZMQCore
```

```
    Bases: object
```

```
    PROTOCOL_MAJOR = 3
```

```
    PROTOCOL_MINOR = 0
```

```
    PROTOCOL_UPDATE = 0
```

```
    PROTOCOL_VERSION = (3, 0, 0)
```

```
    internal_transport = 'ipc'
```

```
    default_comm_mode = 'ipc'
```

```
    default_master_heartbeat = 20.0
```

```
    default_worker_heartbeat = 20.0
```

```
    default_timeout_factor = 5.0
```

```
    default_startup_timeout = 120.0
```

```
    default_shutdown_timeout = 5.0
```

```
    classmethod make_ipc_endpoint()
```

```
    classmethod remove_ipc_endpoints()
```

```
    classmethod make_tcp_endpoint(address='127.0.0.1')
```

```
    classmethod make_internal_endpoint()
```

```
    get_identification()
```

```
    validate_message(message)
```

Validate incoming message. Raises an exception if the message is improperly formatted (TypeError) or does not correspond to the appropriate master (ZMQWEnvironmentError).

```
    message_validation(msg)
```

A context manager for message validation. The instance variable `validation_fail_action` controls the behavior of this context manager:

- 'raise': re-raise the exception that indicated failed validation. Useful for development.
- 'exit' (default): report the error and exit the program.
- 'warn': report the error and continue.

recv_message(*socket*, *flags=0*, *validate=True*, *timeout=None*)

Receive a message object from the given socket, using the given flags. Message validation is performed if *validate* is true. If *timeout* is given, then it is the number of milliseconds to wait prior to raising a `ZMQWMTIMEOUT` exception. *timeout* is ignored if *flags* includes `zmq.NOBLOCK`.

recv_all(*socket*, *flags=0*, *validate=True*)

Receive all messages currently available from the given socket.

recv_ack(*socket*, *flags=0*, *validate=True*, *timeout=None*)

send_message(*socket*, *message*, *payload=None*, *flags=0*)

Send a message object. Subclasses may override this to decorate the message with appropriate IDs, then delegate upward to actually send the message. *message* may either be a pre-constructed `Message` object or a message identifier, in which (latter) case *payload* will become the message payload. *payload* is ignored if *message* is a `Message` object.

send_reply(*socket*, *original_message*, *reply='ok'*, *payload=None*, *flags=0*)

Send a reply to *original_message* on *socket*. The reply message is a `Message` object or a message identifier. The reply *master_id* and *worker_id* are set from *original_message*, unless *master_id* is not set, in which case it is set from *self.master_id*.

send_ack(*socket*, *original_message*)

Send an acknowledgement message, which is mostly just to respect REQ/REP recv/send patterns.

send_nak(*socket*, *original_message*)

Send a negative acknowledgement message.

send_inproc_message(*message*, *payload=None*, *flags=0*)

signal_shutdown()

shutdown_handler(*signal=None*, *frame=None*)

install_signal_handlers(*signals=None*)

install_sigint_handler()

startup()

shutdown()

join()

`westpa.work_managers.zeromq.core.shutdown_process`(*process*, *timeout=1.0*)

class `westpa.work_managers.zeromq.core.IsNode`(*n_local_workers=None*)

Bases: `object`

write_host_info(*filename=None*)

startup()

shutdown()

6.3.2.3 westpa.work_managers.zeromq.node module

Created on Jun 11, 2015

@author: mzwier

class westpa.work_managers.zeromq.node.ZMQCore

Bases: object

PROTOCOL_MAJOR = 3

PROTOCOL_MINOR = 0

PROTOCOL_UPDATE = 0

PROTOCOL_VERSION = (3, 0, 0)

internal_transport = 'ipc'

default_comm_mode = 'ipc'

default_master_heartbeat = 20.0

default_worker_heartbeat = 20.0

default_timeout_factor = 5.0

default_startup_timeout = 120.0

default_shutdown_timeout = 5.0

classmethod make_ipc_endpoint()

classmethod remove_ipc_endpoints()

classmethod make_tcp_endpoint(address='127.0.0.1')

classmethod make_internal_endpoint()

get_identification()

validate_message(message)

Validate incoming message. Raises an exception if the message is improperly formatted (TypeError) or does not correspond to the appropriate master (ZMQWMLEnvironmentError).

message_validation(msg)

A context manager for message validation. The instance variable `validation_fail_action` controls the behavior of this context manager:

- 'raise': re-raise the exception that indicated failed validation. Useful for development.
- 'exit' (default): report the error and exit the program.
- 'warn': report the error and continue.

recv_message(socket, flags=0, validate=True, timeout=None)

Receive a message object from the given socket, using the given flags. Message validation is performed if `validate` is true. If `timeout` is given, then it is the number of milliseconds to wait prior to raising a `ZMQWMLTimeout` exception. `timeout` is ignored if `flags` includes `zmq.NOBLOCK`.

recv_all(socket, flags=0, validate=True)

Receive all messages currently available from the given socket.

recv_ack(*socket*, *flags*=0, *validate*=True, *timeout*=None)

send_message(*socket*, *message*, *payload*=None, *flags*=0)

Send a message object. Subclasses may override this to decorate the message with appropriate IDs, then delegate upward to actually send the message. *message* may either be a pre-constructed **Message** object or a message identifier, in which (latter) case *payload* will become the message payload. *payload* is ignored if *message* is a **Message** object.

send_reply(*socket*, *original_message*, *reply*='ok', *payload*=None, *flags*=0)

Send a reply to *original_message* on *socket*. The reply message is a **Message** object or a message identifier. The reply *master_id* and *worker_id* are set from *original_message*, unless *master_id* is not set, in which case it is set from *self.master_id*.

send_ack(*socket*, *original_message*)

Send an acknowledgement message, which is mostly just to respect REQ/REP recv/send patterns.

send_nak(*socket*, *original_message*)

Send a negative acknowledgement message.

send_inproc_message(*message*, *payload*=None, *flags*=0)

signal_shutdown()

shutdown_handler(*signal*=None, *frame*=None)

install_signal_handlers(*signals*=None)

install_sigint_handler()

startup()

shutdown()

join()

class westpa.work_managers.zeromq.node.**Message**(*message*=None, *payload*=None, *master_id*=None, *src_id*=None)

Bases: object

SHUTDOWN = 'shutdown'

ACK = 'ok'

NAK = 'no'

IDENTIFY = 'identify'

TASKS_AVAILABLE = 'tasks_available'

TASK_REQUEST = 'task_request'

MASTER_BEACON = 'master_alive'

RECONFIGURE_TIMEOUT = 'reconfigure_timeout'

TASK = 'task'

RESULT = 'result'

```
    idempotent_announcement_messages = {'master_alive', 'shutdown', 'tasks_available'}

    classmethod coalesce_announcements(messages)

class westpa.work_managers.zeromq.node.PassiveMultiTimer
    Bases: object
    add_timer(identifier, duration)
    remove_timer(identifier)
    change_duration(identifier, duration)
    reset(identifier=None, at=None)
    expired(identifier, at=None)
    next_expiration()
    next_expiration_in()
    which_expired(at=None)

class westpa.work_managers.zeromq.node.IsNode(n_local_workers=None)
    Bases: object
    write_host_info(filename=None)
    startup()
    shutdown()

class westpa.work_managers.zeromq.node.ThreadProxy(in_type, out_type, mon_type=SocketType.PUB)
    Bases: ProxyBase, ThreadDevice
    Proxy in a Thread. See Proxy for more.

class westpa.work_managers.zeromq.node.ZMQNode(upstream_rr_endpoint, upstream_ann_endpoint,
                                                n_local_workers=None)
    Bases: ZMQCore, IsNode
    run()
    property is_master
    comm_loop()
    startup()
```

6.3.2.4 westpa.work_managers.zeromq.work_manager module

```
class westpa.work_managers.zeromq.work_manager.ZMQCore
    Bases: object
    PROTOCOL_MAJOR = 3
    PROTOCOL_MINOR = 0
    PROTOCOL_UPDATE = 0
```

```
PROTOCOL_VERSION = (3, 0, 0)
```

```
internal_transport = 'ipc'
```

```
default_comm_mode = 'ipc'
```

```
default_master_heartbeat = 20.0
```

```
default_worker_heartbeat = 20.0
```

```
default_timeout_factor = 5.0
```

```
default_startup_timeout = 120.0
```

```
default_shutdown_timeout = 5.0
```

```
classmethod make_ipc_endpoint()
```

```
classmethod remove_ipc_endpoints()
```

```
classmethod make_tcp_endpoint(address='127.0.0.1')
```

```
classmethod make_internal_endpoint()
```

```
get_identification()
```

```
validate_message(message)
```

Validate incoming message. Raises an exception if the message is improperly formatted (TypeError) or does not correspond to the appropriate master (ZMQWMEEnvironmentError).

```
message_validation(msg)
```

A context manager for message validation. The instance variable `validation_fail_action` controls the behavior of this context manager:

- 'raise': re-raise the exception that indicated failed validation. Useful for development.
- 'exit' (default): report the error and exit the program.
- 'warn': report the error and continue.

```
recv_message(socket, flags=0, validate=True, timeout=None)
```

Receive a message object from the given socket, using the given flags. Message validation is performed if `validate` is true. If `timeout` is given, then it is the number of milliseconds to wait prior to raising a `ZMQWMTIMEOUT` exception. `timeout` is ignored if `flags` includes `zmq.NOBLOCK`.

```
recv_all(socket, flags=0, validate=True)
```

Receive all messages currently available from the given socket.

```
recv_ack(socket, flags=0, validate=True, timeout=None)
```

```
send_message(socket, message, payload=None, flags=0)
```

Send a message object. Subclasses may override this to decorate the message with appropriate IDs, then delegate upward to actually send the message. `message` may either be a pre-constructed `Message` object or a message identifier, in which (latter) case `payload` will become the message payload. `payload` is ignored if `message` is a `Message` object.

```
send_reply(socket, original_message, reply='ok', payload=None, flags=0)
```

Send a reply to `original_message` on `socket`. The reply message is a `Message` object or a message identifier. The reply `master_id` and `worker_id` are set from `original_message`, unless `master_id` is not set, in which case it is set from `self.master_id`.

send_ack(*socket, original_message*)

Send an acknowledgement message, which is mostly just to respect REQ/REP recv/send patterns.

send_nak(*socket, original_message*)

Send a negative acknowledgement message.

send_inproc_message(*message, payload=None, flags=0*)

signal_shutdown()

shutdown_handler(*signal=None, frame=None*)

install_signal_handlers(*signals=None*)

install_sigint_handler()

startup()

shutdown()

join()

class westpa.work_managers.zeromq.work_manager.**Message**(*message=None, payload=None, master_id=None, src_id=None*)

Bases: object

SHUTDOWN = 'shutdown'

ACK = 'ok'

NAK = 'no'

IDENTIFY = 'identify'

TASKS_AVAILABLE = 'tasks_available'

TASK_REQUEST = 'task_request'

MASTER_BEACON = 'master_alive'

RECONFIGURE_TIMEOUT = 'reconfigure_timeout'

TASK = 'task'

RESULT = 'result'

idempotent_announcement_messages = {'master_alive', 'shutdown', 'tasks_available'}

classmethod **coalesce_announcements**(*messages*)

class westpa.work_managers.zeromq.work_manager.**Task**(*fn, args, kwargs, task_id=None*)

Bases: object

execute()

Run this task, returning a Result object.

class westpa.work_managers.zeromq.work_manager.**Result**(*task_id, result=None, exception=None, traceback=None*)

Bases: object

exception westpa.work_managers.zeromq.work_manager.ZMQWorkerMissing

Bases: [ZMQWMEError](#)

Exception representing that a worker processing a task died or disappeared

exception westpa.work_managers.zeromq.work_manager.ZMQWMEEnvironmentError

Bases: [ZMQWMEError](#)

Class representing an error in the environment in which the ZeroMQ work manager is running. This includes such things as master/worker ID mismatches.

class westpa.work_managers.zeromq.work_manager.IsNode(*n_local_workers=None*)

Bases: object

write_host_info(*filename=None*)

startup()

shutdown()

class westpa.work_managers.zeromq.work_manager.PassiveMultiTimer

Bases: object

add_timer(*identifier, duration*)

remove_timer(*identifier*)

change_duration(*identifier, duration*)

reset(*identifier=None, at=None*)

expired(*identifier, at=None*)

next_expiration()

next_expiration_in()

which_expired(*at=None*)

westpa.work_managers.zeromq.work_manager.**randport**(*address='127.0.0.1'*)

Select a random unused TCP port number on the given address.

class westpa.work_managers.zeromq.work_manager.ZMQWorker(*rr_endpoint, ann_endpoint*)

Bases: [ZMQCore](#)

This is the outward facing worker component of the ZMQ work manager. This forms the interface to the master. This process cannot hang or crash due to an error in tasks it executes, so tasks are isolated in ZMQExecutor, which communicates with ZMQWorker via (what else?) ZeroMQ.

property **is_master**

update_master_info(*msg*)

identify(*rr_socket*)

request_task(*rr_socket, task_socket*)

handle_reconfigure_timeout(*msg, timers*)

handle_result(*result_socket, rr_socket*)

comm_loop()

Master communication loop for the worker process.

shutdown_executor()

install_signal_handlers(*signals=None*)

startup(*process_index=None*)

class westpa.work_managers.zeromq.work_manager.**ZMQNode**(*upstream_rr_endpoint,*
upstream_ann_endpoint,
n_local_workers=None)

Bases: [ZMQCore](#), [IsNode](#)

run()

property **is_master**

comm_loop()

startup()

class westpa.work_managers.zeromq.work_manager.**WorkManager**

Bases: `object`

Base class for all work managers. At a minimum, work managers must provide a `submit()` function and a `n_workers` attribute (which may be a property), though most will also override `startup()` and `shutdown()`.

classmethod **from_environ**(*wmenv=None*)

classmethod **add_wm_args**(*parser, wmenv=None*)

sigint_handler(*signal, frame*)

install_sigint_handler()

startup()

Perform any necessary startup work, such as spawning clients.

shutdown()

Cleanly shut down any active workers.

run()

Run the worker loop (in clients only).

submit(*fn, args=None, kwargs=None*)

Submit a task to the work manager, returning a *WMFuture* object representing the pending result. `fn(*args, **kwargs)` will be executed by a worker, and the return value assigned as the result of the returned future. The function `fn` and all arguments must be picklable; note particularly that off-path modules (like the `system` module and any active plugins) are not picklable unless pre-loaded in the worker process (i.e. prior to forking the master).

submit_many(*tasks*)

Submit a set of tasks to the work manager, returning a list of *WMFuture* objects representing pending results. Each entry in `tasks` should be a triple (`fn, args, kwargs`), which will result in `fn(*args, **kwargs)` being executed by a worker. The function `fn` and all arguments must be picklable; note particularly that off-path modules are not picklable unless pre-loaded in the worker process.

as_completed(futures)

Return a generator which yields results from the given `futures` as they become available.

submit_as_completed(task_generator, queue_size=None)

Return a generator which yields results from a set of `futures` as they become available. Futures are generated by the `task_generator`, which must return a triple of the form expected by `submit`. The method also accepts an int `queue_size` that dictates the maximum number of Futures that should be pending at any given time. The default value of `None` submits all of the tasks at once.

wait_any(futures)

Wait on any of the given `futures` and return the first one which has a result available. If more than one result is or becomes available simultaneously, any completed future may be returned.

wait_all(futures)

A convenience function which waits on all the given `futures` in order. This function returns the same `futures` as submitted to the function as a list, indicating the order in which waits occurred.

property is_master

True if this is the master process for task distribution. This is necessary, e.g., for MPI, where all processes start identically and then must branch depending on rank.

class westpa.work_managers.zeromq.work_manager.WMFuture(task_id=None)

Bases: object

A “future”, representing work which has been dispatched for completion asynchronously.

static all_acquired(futures)

Context manager to acquire all locks on the given `futures`. Primarily for internal use.

get_result(discard=True)

Get the result associated with this future, blocking until it is available. If `discard` is true, then removes the reference to the result contained in this instance, so that a collection of futures need not turn into a cache of all associated results.

property result**wait()**

Wait until this future has a result or exception available.

get_exception()

Get the exception associated with this future, blocking until it is available.

property exception

Get the exception associated with this future, blocking until it is available.

get_traceback()

Get the traceback object associated with this future, if any.

property traceback

Get the traceback object associated with this future, if any.

is_done()

Indicates whether this future is done executing (may block if this future is being updated).

property done

Indicates whether this future is done executing (may block if this future is being updated).

class westpa.work_managers.zeromq.work_manager.deque

Bases: object

deque([iterable[, maxlen]]) → deque object

A list-like sequence optimized for data accesses near its endpoints.

append()

Add an element to the right side of the deque.

appendleft()

Add an element to the left side of the deque.

clear()

Remove all elements from the deque.

copy()

Return a shallow copy of a deque.

count()

D.count(value) – return number of occurrences of value

extend()

Extend the right side of the deque with elements from the iterable

extendleft()

Extend the left side of the deque with elements from the iterable

index()

D.index(value, [start, [stop]]) – return first index of value. Raises ValueError if the value is not present.

insert()

D.insert(index, object) – insert object before index

maxlen

maximum size of a deque or None if unbounded

pop()

Remove and return the rightmost element.

popleft()

Remove and return the leftmost element.

remove()

D.remove(value) – remove first occurrence of value.

reverse()

D.reverse() – reverse *IN PLACE*

rotate()

Rotate the deque n steps to the right (default n=1). If n is negative, rotates left.

class westpa.work_managers.zeromq.work_manager.ZMQWorkManager(*n_local_workers=1*)

Bases: [ZMQCore](#), [WorkManager](#), [IsNode](#)

classmethod add_wm_args(*parser, wmenv=None*)

classmethod from_envron(*wmenv=None*)

classmethod `read_host_info(filename)`

classmethod `canonicalize_endpoint(endpoint, allow_wildcard_host=True)`

property `n_workers`

submit(*fn, args=None, kwargs=None*)

Submit a task to the work manager, returning a *WMFuture* object representing the pending result. *fn(*args, **kwargs)* will be executed by a worker, and the return value assigned as the result of the returned future. The function *fn* and all arguments must be picklable; note particularly that off-path modules (like the system module and any active plugins) are not picklable unless pre-loaded in the worker process (i.e. prior to forking the master).

submit_many(*tasks*)

Submit a set of tasks to the work manager, returning a list of *WMFuture* objects representing pending results. Each entry in *tasks* should be a triple (*fn, args, kwargs*), which will result in *fn(*args, **kwargs)* being executed by a worker. The function *fn* and all arguments must be picklable; note particularly that off-path modules are not picklable unless pre-loaded in the worker process.

send_message(*socket, message, payload=None, flags=0*)

Send a message object. Subclasses may override this to decorate the message with appropriate IDs, then delegate upward to actually send the message. *message* may either be a pre-constructed *Message* object or a message identifier, in which (latter) case *payload* will become the message payload. *payload* is ignored if *message* is a *Message* object.

handle_result(*socket, msg*)

handle_task_request(*socket, msg*)

update_worker_information(*msg*)

check_workers()

remove_worker(*worker_id*)

shutdown_clear_tasks()

Abort pending tasks with error on shutdown.

comm_loop()

startup()

Perform any necessary startup work, such as spawning clients.

shutdown()

Cleanly shut down any active workers.

6.3.2.5 westpa.work_managers.zeromq.worker module

Created on May 29, 2015

@author: mzwier

class `westpa.work_managers.zeromq.worker.ZMQCore`

Bases: `object`

`PROTOCOL_MAJOR = 3`

```
PROTOCOL_MINOR = 0
```

```
PROTOCOL_UPDATE = 0
```

```
PROTOCOL_VERSION = (3, 0, 0)
```

```
internal_transport = 'ipc'
```

```
default_comm_mode = 'ipc'
```

```
default_master_heartbeat = 20.0
```

```
default_worker_heartbeat = 20.0
```

```
default_timeout_factor = 5.0
```

```
default_startup_timeout = 120.0
```

```
default_shutdown_timeout = 5.0
```

```
classmethod make_ipc_endpoint()
```

```
classmethod remove_ipc_endpoints()
```

```
classmethod make_tcp_endpoint(address='127.0.0.1')
```

```
classmethod make_internal_endpoint()
```

```
get_identification()
```

```
validate_message(message)
```

Validate incoming message. Raises an exception if the message is improperly formatted (TypeError) or does not correspond to the appropriate master (ZMQWMEEnvironmentError).

```
message_validation(msg)
```

A context manager for message validation. The instance variable `validation_fail_action` controls the behavior of this context manager:

- ‘raise’: re-raise the exception that indicated failed validation. Useful for development.
- ‘exit’ (default): report the error and exit the program.
- ‘warn’: report the error and continue.

```
recv_message(socket, flags=0, validate=True, timeout=None)
```

Receive a message object from the given socket, using the given flags. Message validation is performed if `validate` is true. If `timeout` is given, then it is the number of milliseconds to wait prior to raising a `ZMQWMTIMEOUT` exception. `timeout` is ignored if `flags` includes `zmq.NOBLOCK`.

```
recv_all(socket, flags=0, validate=True)
```

Receive all messages currently available from the given socket.

```
recv_ack(socket, flags=0, validate=True, timeout=None)
```

```
send_message(socket, message, payload=None, flags=0)
```

Send a message object. Subclasses may override this to decorate the message with appropriate IDs, then delegate upward to actually send the message. `message` may either be a pre-constructed `Message` object or a message identifier, in which (latter) case `payload` will become the message payload. `payload` is ignored if `message` is a `Message` object.

send_reply(*socket, original_message, reply='ok', payload=None, flags=0*)

Send a reply to *original_message* on *socket*. The reply message is a *Message* object or a message identifier. The reply *master_id* and *worker_id* are set from *original_message*, unless *master_id* is not set, in which case it is set from *self.master_id*.

send_ack(*socket, original_message*)

Send an acknowledgement message, which is mostly just to respect REQ/REP recv/send patterns.

send_nak(*socket, original_message*)

Send a negative acknowledgement message.

send_inproc_message(*message, payload=None, flags=0*)

signal_shutdown()

shutdown_handler(*signal=None, frame=None*)

install_signal_handlers(*signals=None*)

install_sigint_handler()

startup()

shutdown()

join()

class westpa.work_managers.zeromq.worker.**Message**(*message=None, payload=None, master_id=None, src_id=None*)

Bases: *object*

SHUTDOWN = 'shutdown'

ACK = 'ok'

NAK = 'no'

IDENTIFY = 'identify'

TASKS_AVAILABLE = 'tasks_available'

TASK_REQUEST = 'task_request'

MASTER_BEACON = 'master_alive'

RECONFIGURE_TIMEOUT = 'reconfigure_timeout'

TASK = 'task'

RESULT = 'result'

idempotent_announcement_messages = {'master_alive', 'shutdown', 'tasks_available'}

classmethod **coalesce_announcements**(*messages*)

exception westpa.work_managers.zeromq.worker.**ZMQWTimeout**

Bases: *ZMQWEnvironmentError*

A timeout of a sort that indicatess that a master or worker has failed or never started.

```
class westpa.work_managers.zeromq.worker.PassiveMultiTimer
```

Bases: object

```
add_timer(identifier, duration)
```

```
remove_timer(identifier)
```

```
change_duration(identifier, duration)
```

```
reset(identifier=None, at=None)
```

```
expired(identifier, at=None)
```

```
next_expiration()
```

```
next_expiration_in()
```

```
which_expired(at=None)
```

```
class westpa.work_managers.zeromq.worker.Task(fn, args, kwargs, task_id=None)
```

Bases: object

```
execute()
```

Run this task, returning a Result object.

```
class westpa.work_managers.zeromq.worker.Result(task_id, result=None, exception=None,  
                                              traceback=None)
```

Bases: object

```
class westpa.work_managers.zeromq.worker.ZMQWorker(rr_endpoint, ann_endpoint)
```

Bases: [ZMQCore](#)

This is the outward facing worker component of the ZMQ work manager. This forms the interface to the master. This process cannot hang or crash due to an error in tasks it executes, so tasks are isolated in ZMQExecutor, which communicates with ZMQWorker via (what else?) ZeroMQ.

```
property is_master
```

```
update_master_info(msg)
```

```
identify(rr_socket)
```

```
request_task(rr_socket, task_socket)
```

```
handle_reconfigure_timeout(msg, timers)
```

```
handle_result(result_socket, rr_socket)
```

```
comm_loop()
```

Master communication loop for the worker process.

```
shutdown_executor()
```

```
install_signal_handlers(signals=None)
```

```
startup(process_index=None)
```

class westpa.work_managers.zeromq.worker.ZMQExecutor(*task_endpoint, result_endpoint*)

Bases: [ZMQCore](#)

The is the component of the ZMQ WM worker that actually executes tasks. This is isolated in a separate process and controlled via ZMQ from the ZMQWorker.

comm_loop()

startup(*process_index=None*)

6.4 westpa.tools package

6.4.1 westpa.tools module

tools – classes for implementing command-line tools for WESTPA

class westpa.tools.WESTTool

Bases: [WESTToolComponent](#)

Base class for WEST command line tools

prog = None

usage = None

description = None

epilog = None

add_args(*parser*)

Add arguments specific to this tool to the given argparse parser.

process_args(*args*)

Take argparse-processed arguments associated with this tool and deal with them appropriately (setting instance variables, etc)

make_parser(*prog=None, usage=None, description=None, epilog=None, args=None*)

make_parser_and_process(*prog=None, usage=None, description=None, epilog=None, args=None*)

A convenience function to create a parser, call `add_all_args()`, and then call `process_all_args()`. The argument namespace is returned.

go()

Perform the analysis associated with this tool.

main()

A convenience function to make a parser, parse and process arguments, then call `self.go()`

class westpa.tools.WESTParallelTool(*wm_env=None*)

Bases: [WESTTool](#)

Base class for command-line tools parallelized with wwmgr. This automatically adds and processes wwmgr command-line arguments and creates a work manager at `self.work_manager`.

make_parser_and_process(*prog=None, usage=None, description=None, epilog=None, args=None*)

A convenience function to create a parser, call `add_all_args()`, and then call `process_all_args()`. The argument namespace is returned.

add_args(*parser*)

Add arguments specific to this tool to the given argparse parser.

process_args(*args*)

Take argparse-processed arguments associated with this tool and deal with them appropriately (setting instance variables, etc)

go()

Perform the analysis associated with this tool.

main()

A convenience function to make a parser, parse and process arguments, then run self.go() in the master process.

class westpa.tools.**WESTToolComponent**

Bases: object

Base class for WEST command line tools and components used in constructing tools

include_arg(*argname*)

exclude_arg(*argname*)

set_arg_default(*argname*, *value*)

add_args(*parser*)

Add arguments specific to this component to the given argparse parser.

process_args(*args*)

Take argparse-processed arguments associated with this component and deal with them appropriately (setting instance variables, etc)

add_all_args(*parser*)

Add arguments for all components from which this class derives to the given parser, starting with the class highest up the inheritance chain (most distant ancestor).

process_all_args(*args*)

class westpa.tools.**WESTSubcommand**(*parent*)

Bases: [WESTToolComponent](#)

Base class for command-line tool subcommands. A little sugar for making this more uniform.

subcommand = None

help_text = None

description = None

add_to_subparsers(*subparsers*)

go()

property **work_manager**

The work manager for this tool. Raises AttributeError if this is not a parallel tool.

class westpa.tools.**WESTMasterCommand**

Bases: [WESTTool](#)

Base class for command-line tools that employ subcommands

subparsers_title = None

subcommands = None

include_help_command = True

add_args(*parser*)

Add arguments specific to this tool to the given argparse parser.

process_args(*args*)

Take argparse-processed arguments associated with this tool and deal with them appropriately (setting instance variables, etc)

go()

Perform the analysis associated with this tool.

class westpa.tools.**WESTMultiTool**(*wm_env=None*)

Bases: [*WESTParallelTool*](#)

Base class for command-line tools which work with multiple simulations. Automatically parses for and gives commands to load multiple files.

make_parser_and_process(*prog=None, usage=None, description=None, epilog=None, args=None*)

A convenience function to create a parser, call `add_all_args()`, and then call `process_all_args()`. The argument namespace is returned.

parse_from_yaml(*yamlfilepath*)

Parse options from YAML input file. Command line arguments take precedence over options specified in the YAML hierarchy. TODO: add description on how YAML files should be constructed.

add_args(*parser*)

Add arguments specific to this tool to the given argparse parser.

exception NoSimulationsException

Bases: Exception

generate_file_list(*key_list*)

A convenience function which takes in a list of keys that are filenames, and returns a dictionary which contains all the individual files loaded inside of a dictionary keyed to the filename.

process_args(*args*)

Take argparse-processed arguments associated with this tool and deal with them appropriately (setting instance variables, etc)

go()

Perform the analysis associated with this tool.

main()

A convenience function to make a parser, parse and process arguments, then run `self.go()` in the master process.

class westpa.tools.**WESTDataReader**

Bases: [*WESTToolComponent*](#)

Tool for reading data from WEST-related HDF5 files. Coordinates finding the main HDF5 file from `west.cfg` or command line arguments, caching of certain kinds of data (eventually), and retrieving auxiliary data sets from various places.

add_args(*parser*)

Add arguments specific to this component to the given argparse parser.

process_args(*args*)

Take argparse-processed arguments associated with this component and deal with them appropriately (setting instance variables, etc)

open(*mode='r'*)

close()

property **weight_dsspec**

property **parent_id_dsspec**

class westpa.tools.**WESTDSSynthesizer**(*default_dsname=None, h5filename=None*)

Bases: [WESTToolComponent](#)

Tool for synthesizing a dataset for analysis from other datasets. This may be done using a custom function, or a list of “data set specifications”. It is anticipated that if several source datasets are required, then a tool will have multiple instances of this class.

group_name = 'input dataset options'

add_args(*parser*)

Add arguments specific to this component to the given argparse parser.

process_args(*args*)

Take argparse-processed arguments associated with this component and deal with them appropriately (setting instance variables, etc)

class westpa.tools.**WESTWDSynthesizer**(*default_dsname=None, h5filename=None*)

Bases: [WESTToolComponent](#)

group_name = 'weight dataset options'

add_args(*parser*)

Add arguments specific to this component to the given argparse parser.

process_args(*args*)

Take argparse-processed arguments associated with this component and deal with them appropriately (setting instance variables, etc)

class westpa.tools.**IterRangeSelection**(*data_manager=None*)

Bases: [WESTToolComponent](#)

Select and record limits on iterations used in analysis and/or reporting. This class provides both the user-facing command-line options and parsing, and the application-side API for recording limits in HDF5.

HDF5 datasets calculated based on a restricted set of iterations should be tagged with the following attributes:

first_iter

The first iteration included in the calculation.

last_iter

One past the last iteration included in the calculation.

iter_step

Blocking or sampling period for iterations included in the calculation.

add_args(*parser*)

Add arguments specific to this component to the given argparse parser.

process_args(args, override_iter_start=None, override_iter_stop=None, default_iter_step=1)

Take argparse-processed arguments associated with this component and deal with them appropriately (setting instance variables, etc)

iter_block_iter()

Return an iterable of (block_start,block_end) over the blocks of iterations selected by --first-iter/--last-iter/--step-iter.

n_iter_blocks()

Return the number of blocks of iterations (as returned by `iter_block_iter`) selected by --first-iter/--last-iter/--step-iter.

record_data_iter_range(h5object, iter_start=None, iter_stop=None)

Store attributes `iter_start` and `iter_stop` on the given HDF5 object (group/dataset)

record_data_iter_step(h5object, iter_step=None)

Store attribute `iter_step` on the given HDF5 object (group/dataset).

check_data_iter_range_least(h5object, iter_start=None, iter_stop=None)

Check that the given HDF5 object contains (as denoted by its `iter_start/iter_stop` attributes) data at least for the iteration range specified.

check_data_iter_range_equal(h5object, iter_start=None, iter_stop=None)

Check that the given HDF5 object contains (as denoted by its `iter_start/iter_stop` attributes) data exactly for the iteration range specified.

check_data_iter_step_conformant(h5object, iter_step=None)

Check that the given HDF5 object contains per-iteration data at an iteration stride suitable for extracting data with the given stride (in other words, the given `iter_step` is a multiple of the stride with which data was recorded).

check_data_iter_step_equal(h5object, iter_step=None)

Check that the given HDF5 object contains per-iteration data at an iteration stride the same as that specified.

slice_per_iter_data(dataset, iter_start=None, iter_stop=None, iter_step=None, axis=0)

Return the subset of the given dataset corresponding to the given iteration range and stride. Unless otherwise specified, the first dimension of the dataset is the one sliced.

iter_range(iter_start=None, iter_stop=None, iter_step=None, dtype=None)

Return a sequence for the given iteration numbers and stride, filling in missing values from those stored on self. The smallest data type capable of holding `iter_stop` is returned unless otherwise specified using the `dtype` argument.

class westpa.tools.SegSelector

Bases: [WESTToolComponent](#)

add_args(parser)

Add arguments specific to this component to the given argparse parser.

process_args(args)

Take argparse-processed arguments associated with this component and deal with them appropriately (setting instance variables, etc)

parse_segssel_file(filename)

class westpa.tools.BinMappingComponentBases: *WESTToolComponent*

Component for obtaining a bin mapper from one of several places based on command-line arguments. Such locations include an HDF5 file that contains pickled mappers (including the primary WEST HDF5 file), the system object, an external function, or (in the common case of rectilinear bins) a list of lists of bin boundaries.

Some configuration is necessary prior to calling `process_args()` if loading a mapper from HDF5. Specifically, either `set_we_h5file_info()` or `set_other_h5file_info()` must be called to describe where to find the appropriate mapper. In the case of `set_we_h5file_info()`, the mapper used for WE at the end of a given iteration will be loaded. In the case of `set_other_h5file_info()`, an arbitrary group and hash value are specified; the mapper corresponding to that hash in the given group will be returned.

In the absence of arguments, the mapper contained in an existing HDF5 file is preferred; if that is not available, the mapper from the system driver is used.

This component adds the following arguments to argument parsers:

--bins-from-system Obtain bins from the system driver

—bins-from-expr=EXPR Construct rectilinear bins by parsing EXPR and calling `RectilinearBinMapper()` with the result. EXPR must therefore be a list of lists.

-bins-from-function=[PATH:]MODULE.FUNC

Call an external function FUNC in module MODULE (optionally adding PATH to the search path when loading MODULE) which, when called, returns a fully-constructed bin mapper.

—bins-from-file Load bin definitions from a YAML configuration file.

--bins-from-h5file Load bins from the file being considered; this is intended to mean the master WEST HDF5 file or results of other binning calculations, as appropriate.

add_args(*parser*, *description*='binning options', *suppress*=[])

Add arguments specific to this component to the given argparse parser.

add_target_count_args(*parser*, *description*='bin target count options')

Add options to the given parser corresponding to target counts.

process_args(*args*)

Take argparse-processed arguments associated with this component and deal with them appropriately (setting instance variables, etc)

set_we_h5file_info(*n_iter*=None, *data_manager*=None, *required*=False)

Set up to load a bin mapper from the master WEST HDF5 file. The mapper is actually loaded from the file when `self.load_bin_mapper()` is called, if and only if command line arguments direct this. If `required` is true, then a mapper must be available at iteration `n_iter`, or else an exception will be raised.

set_other_h5file_info(*topology_group*, *hashval*)

Set up to load a bin mapper from (any) open HDF5 file, where bin topologies are stored in `topology_group` (an `h5py Group` object) and the desired mapper has hash value `hashval`. The mapper itself is loaded when `self.load_bin_mapper()` is called.

`westpa.tools.mapper_from_dict`(*ybins*)

class westpa.tools.ProgressIndicatorComponentBases: *WESTToolComponent*

add_args(*parser*)

Add arguments specific to this component to the given argparse parser.

process_args(*args*)

Take argparse-processed arguments associated with this component and deal with them appropriately (setting instance variables, etc)

class westpa.tools.**Plotter**(*h5file, h5key, iteration=-1, interface='matplotlib'*)

Bases: object

This is a semi-generic plotting interface that has a built in curses based terminal plotter. It's fairly specific to what we're using it for here, but we could (and maybe should) build it out into a little library that we can use via the command line to plot things. Might be useful for looking at data later. That would also cut the size of this tool down by a good bit.

plot(*i=0, j=1, tau=1, iteration=None, dim=0, interface=None*)

class westpa.tools.**WIPIDataset**(*raw, key*)

Bases: object

keys()

class westpa.tools.**KineticsIteration**(*kin_h5file, index, assign, iteration=-1*)

Bases: object

keys()

class westpa.tools.**WIPIScheme**(*scheme, name, parent, settings*)

Bases: object

property scheme

property list_schemes

Lists what schemes are configured in west.cfg file. Schemes should be structured as follows, in west.cfg:

west:

system:

analysis:

directory: analysis analysis_schemes:

scheme.1:

enabled: True states:

- label: unbound coords: [[7.0]]
- label: bound coords: [[2.7]]

bins:

- type: RectilinearBinMapper boundaries: [[0.0, 2.80, 7, 10000]]

property iteration

property assign

property direct

The output from w_direct.py from the current scheme.

property state_labels

property bin_labels

property west

property reweight

property current

The current iteration. See help for `__get_data_for_iteration__`

property past

The previous iteration. See help for `__get_data_for_iteration__`

6.4.2 westpa.tools.binning module

class `westpa.tools.binning.count`(*start=0, step=1*)

Bases: `object`

Return a count object whose `__next__()` method returns consecutive values.

Equivalent to:

```
def count(firstval=0, step=1):
```

```
    x = firstval while 1:
```

```
        yield x
        x += step
```

exception `westpa.tools.binning.PickleError`

Bases: `Exception`

class `westpa.tools.binning.RectilinearBinMapper`(*boundaries*)

Bases: `BinMapper`

Bin into a rectangular grid based on tuples of float values

property boundaries

assign(*coords, mask=None, output=None*)

`westpa.tools.binning.weight_dtype`

alias of `float64`

`westpa.tools.binning.get_object`(*object_name, path=None*)

Attempt to load the given object, using additional path information if given.

class `westpa.tools.binning.WESTToolComponent`

Bases: `object`

Base class for WEST command line tools and components used in constructing tools

include_arg(*argname*)

exclude_arg(*argname*)

set_arg_default(*argname, value*)

add_args(*parser*)

Add arguments specific to this component to the given argparse parser.

process_args(*args*)

Take argparse-processed arguments associated with this component and deal with them appropriately (setting instance variables, etc)

add_all_args(*parser*)

Add arguments for all components from which this class derives to the given parser, starting with the class highest up the inheritance chain (most distant ancestor).

process_all_args(*args*)

westpa.tools.binning.mapper_from_expr(*expr*)

westpa.tools.binning.mapper_from_system()

westpa.tools.binning.mapper_from_function(*funcspec*)

Return a mapper constructed by calling a function in a named module. *funcspec* should be formatted as [PATH]:MODULE.FUNC. This function loads MODULE, optionally adding PATH to the search path, then returns MODULE.FUNC()

westpa.tools.binning.mapper_from_hdf5(*topol_group*, *hashval*)

Retrieve the mapper identified by *hashval* from the given bin topology group *topol_group*. Returns (mapper, pickle, hashval)

westpa.tools.binning.mapper_from_yaml(*yamlfilename*)

westpa.tools.binning.mapper_from_dict(*ybins*)

westpa.tools.binning.write_bin_info(*mapper*, *assignments*, *weights*, *n_target_states*,
outfile=<_io.TextIOWrapper name='<stdout>' mode='w'
encoding='utf-8'>, *detailed*=False)

Write information about binning to *outfile*, given a mapper (*mapper*) and the weights (*weights*) and bin assignments (*assignments*) of a set of segments, along with a target state count (*n_target_states*). If *detailed* is true, then per-bin information is written as well as summary information about all bins.

westpa.tools.binning.write_bin_labels(*mapper*, *dest*, *header*='# bin labels:\n', *fmt*='# bin
{index:{max_iwidth}d} -- {label!s}\n')

Print labels for all bins in *mapper* to the file-like object ``dest``.

If provided, *header* is printed prior to any labels. A number of expansions are available in *header*:

- *mapper* – the mapper itself (from which most of the following can be obtained)
- *classname* – the class name of the mapper
- *nbins* – number of bins in the mapper

The *fmt* string specifies how bin labels are to be printed. A number of expansions are available in *fmt*:

- *index* – the zero-based index of the bin
- *label* – the label of the bin
- *max_iwidth* – the maximum width (in characters) of the bin index, for pretty alignment

class westpa.tools.binning.BinMappingComponent

Bases: [WESTToolComponent](#)

Component for obtaining a bin mapper from one of several places based on command-line arguments. Such locations include an HDF5 file that contains pickled mappers (including the primary WEST HDF5 file), the system object, an external function, or (in the common case of rectilinear bins) a list of lists of bin boundaries.

Some configuration is necessary prior to calling *process_args*() if loading a mapper from HDF5. Specifically, either *set_we_h5file_info*() or *set_other_h5file_info*() must be called to describe where to find the appropriate mapper. In the case of *set_we_h5file_info*(), the mapper used for WE at the end of a given iteration will be loaded. In the case of *set_other_h5file_info*(), an arbitrary group and hash value are specified; the mapper corresponding to that hash in the given group will be returned.

In the absence of arguments, the mapper contained in an existing HDF5 file is preferred; if that is not available, the mapper from the system driver is used.

This component adds the following arguments to argument parsers:

--bins-from-system Obtain bins from the system driver

—bins-from-expr=EXPR Construct rectilinear bins by parsing EXPR and calling RectilinearBinMapper() with the result. EXPR must therefore be a list of lists.

-bins-from-function=[PATH:]MODULE.FUNC

Call an external function FUNC in module MODULE (optionally adding PATH to the search path when loading MODULE) which, when called, returns a fully-constructed bin mapper.

—bins-from-file Load bin definitions from a YAML configuration file.

--bins-from-h5file Load bins from the file being considered; this is intended to mean the master WEST HDF5 file or results of other binning calculations, as appropriate.

add_args(*parser*, *description*='binning options', *suppress*=[])

Add arguments specific to this component to the given argparse parser.

add_target_count_args(*parser*, *description*='bin target count options')

Add options to the given parser corresponding to target counts.

process_args(*args*)

Take argparse-processed arguments associated with this component and deal with them appropriately (setting instance variables, etc)

set_we_h5file_info(*n_iter*=None, *data_manager*=None, *required*=False)

Set up to load a bin mapper from the master WEST HDF5 file. The mapper is actually loaded from the file when `self.load_bin_mapper()` is called, if and only if command line arguments direct this. If `required` is true, then a mapper must be available at iteration `n_iter`, or else an exception will be raised.

set_other_h5file_info(*topology_group*, *hashval*)

Set up to load a bin mapper from (any) open HDF5 file, where bin topologies are stored in `topology_group` (an `h5py Group` object) and the desired mapper has hash value `hashval`. The mapper itself is loaded when `self.load_bin_mapper()` is called.

6.4.3 westpa.tools.core module

Core classes for creating WESTPA command-line tools

class `westpa.tools.core.WESTToolComponent`

Bases: `object`

Base class for WEST command line tools and components used in constructing tools

include_arg(*argname*)

exclude_arg(*argname*)

set_arg_default(*argname*, *value*)

add_args(*parser*)

Add arguments specific to this component to the given argparse parser.

process_args(*args*)

Take argparse-processed arguments associated with this component and deal with them appropriately (setting instance variables, etc)

add_all_args(*parser*)

Add arguments for all components from which this class derives to the given parser, starting with the class highest up the inheritance chain (most distant ancestor).

process_all_args(*args*)

class westpa.tools.core.**WESTTool**

Bases: [*WESTToolComponent*](#)

Base class for WEST command line tools

prog = None

usage = None

description = None

epilog = None

add_args(*parser*)

Add arguments specific to this tool to the given argparse parser.

process_args(*args*)

Take argparse-processed arguments associated with this tool and deal with them appropriately (setting instance variables, etc)

make_parser(*prog=None, usage=None, description=None, epilog=None, args=None*)

make_parser_and_process(*prog=None, usage=None, description=None, epilog=None, args=None*)

A convenience function to create a parser, call `add_all_args()`, and then call `process_all_args()`. The argument namespace is returned.

go()

Perform the analysis associated with this tool.

main()

A convenience function to make a parser, parse and process arguments, then call `self.go()`

class westpa.tools.core.**WESTParallelTool**(*wm_env=None*)

Bases: [*WESTTool*](#)

Base class for command-line tools parallelized with wwmgr. This automatically adds and processes wwmgr command-line arguments and creates a work manager at `self.work_manager`.

make_parser_and_process(*prog=None, usage=None, description=None, epilog=None, args=None*)

A convenience function to create a parser, call `add_all_args()`, and then call `process_all_args()`. The argument namespace is returned.

add_args(*parser*)

Add arguments specific to this tool to the given argparse parser.

process_args(*args*)

Take argparse-processed arguments associated with this tool and deal with them appropriately (setting instance variables, etc)

go()

Perform the analysis associated with this tool.

main()

A convenience function to make a parser, parse and process arguments, then run `self.go()` in the master process.

class `westpa.tools.core.WESTMultiTool(wm_env=None)`

Bases: [`WESTParallelTool`](#)

Base class for command-line tools which work with multiple simulations. Automatically parses for and gives commands to load multiple files.

make_parser_and_process(*prog=None, usage=None, description=None, epilog=None, args=None*)

A convenience function to create a parser, call `add_all_args()`, and then call `process_all_args()`. The argument namespace is returned.

parse_from_yaml(*yamlfilepath*)

Parse options from YAML input file. Command line arguments take precedence over options specified in the YAML hierarchy. TODO: add description on how YAML files should be constructed.

add_args(*parser*)

Add arguments specific to this tool to the given argparse parser.

exception NoSimulationsException

Bases: `Exception`

generate_file_list(*key_list*)

A convenience function which takes in a list of keys that are filenames, and returns a dictionary which contains all the individual files loaded inside of a dictionary keyed to the filename.

process_args(*args*)

Take argparse-processed arguments associated with this tool and deal with them appropriately (setting instance variables, etc)

go()

Perform the analysis associated with this tool.

main()

A convenience function to make a parser, parse and process arguments, then run `self.go()` in the master process.

class `westpa.tools.core.WESTSubcommand(parent)`

Bases: [`WESTToolComponent`](#)

Base class for command-line tool subcommands. A little sugar for making this more uniform.

subcommand = `None`

help_text = `None`

description = `None`

add_to_subparsers(*subparsers*)

go()

property work_manager

The work manager for this tool. Raises `AttributeError` if this is not a parallel tool.

```
class westpa.tools.core.WESTMasterCommand
```

Bases: [WESTTool](#)

Base class for command-line tools that employ subcommands

```
subparsers_title = None
```

```
subcommands = None
```

```
include_help_command = True
```

```
add_args(parser)
```

Add arguments specific to this tool to the given argparse parser.

```
process_args(args)
```

Take argparse-processed arguments associated with this tool and deal with them appropriately (setting instance variables, etc)

```
go()
```

Perform the analysis associated with this tool.

6.4.4 westpa.tools.data_reader module

```
class westpa.tools.data_reader.WESTToolComponent
```

Bases: `object`

Base class for WEST command line tools and components used in constructing tools

```
include_arg(argname)
```

```
exclude_arg(argname)
```

```
set_arg_default(argname, value)
```

```
add_args(parser)
```

Add arguments specific to this component to the given argparse parser.

```
process_args(args)
```

Take argparse-processed arguments associated with this component and deal with them appropriately (setting instance variables, etc)

```
add_all_args(parser)
```

Add arguments for all components from which this class derives to the given parser, starting with the class highest up the inheritance chain (most distant ancestor).

```
process_all_args(args)
```

```
westpa.tools.data_reader.get_object(object_name, path=None)
```

Attempt to load the given object, using additional path information if given.

```
class westpa.tools.data_reader.FnDSSpec(h5file_or_name, fn)
```

Bases: [FileLinkedDSSpec](#)

```
get_iter_data(n_iter, seg_slice=(slice(None, None, None),))
```

```
class westpa.tools.data_reader.MultiDSSpec(dsspecs)
```

Bases: [DSSpec](#)

```
get_iter_data(n_iter, seg_slice=(slice(None, None, None),))
```

```
class westpa.tools.data_reader.SingleSegmentDSSpec(h5file_or_name, dsname, alias=None,
                                                    slice=None)
```

Bases: [SingleDSSpec](#)

```
get_iter_data(n_iter, seg_slice=(slice(None, None, None),))
```

```
get_segment_data(n_iter, seg_id)
```

```
class westpa.tools.data_reader.SingleIterDSSpec(h5file_or_name, dsname, alias=None, slice=None)
```

Bases: [SingleDSSpec](#)

```
get_iter_data(n_iter, seg_slice=(slice(None, None, None),))
```

```
class westpa.tools.data_reader.WESTDataReader
```

Bases: [WESTToolComponent](#)

Tool for reading data from WEST-related HDF5 files. Coordinates finding the main HDF5 file from west.cfg or command line arguments, caching of certain kinds of data (eventually), and retrieving auxiliary data sets from various places.

```
add_args(parser)
```

Add arguments specific to this component to the given argparse parser.

```
process_args(args)
```

Take argparse-processed arguments associated with this component and deal with them appropriately (setting instance variables, etc)

```
open(mode='r')
```

```
close()
```

```
property weight_dsspec
```

```
property parent_id_dsspec
```

```
class westpa.tools.data_reader.WESTDSSynthesizer(default_dsname=None, h5filename=None)
```

Bases: [WESTToolComponent](#)

Tool for synthesizing a dataset for analysis from other datasets. This may be done using a custom function, or a list of “data set specifications”. It is anticipated that if several source datasets are required, then a tool will have multiple instances of this class.

```
group_name = 'input dataset options'
```

```
add_args(parser)
```

Add arguments specific to this component to the given argparse parser.

```
process_args(args)
```

Take argparse-processed arguments associated with this component and deal with them appropriately (setting instance variables, etc)

```
class westpa.tools.data_reader.WESTWDSSynthesizer(default_dsname=None, h5filename=None)
```

Bases: [WESTToolComponent](#)

```
group_name = 'weight dataset options'
```

add_args(*parser*)

Add arguments specific to this component to the given argparse parser.

process_args(*args*)

Take argparse-processed arguments associated with this component and deal with them appropriately (setting instance variables, etc)

6.4.5 westpa.tools.dtypes module

Numpy/HDF5 data types shared among several WESTPA tools

westpa.tools.dtypes.n_iter_dtype

alias of uint32

westpa.tools.dtypes.seg_id_dtype

alias of int64

westpa.tools.dtypes.weight_dtype

alias of float64

6.4.6 westpa.tools.iter_range module

class westpa.tools.iter_range.WESTToolComponent

Bases: object

Base class for WEST command line tools and components used in constructing tools

include_arg(*argname*)

exclude_arg(*argname*)

set_arg_default(*argname*, *value*)

add_args(*parser*)

Add arguments specific to this component to the given argparse parser.

process_args(*args*)

Take argparse-processed arguments associated with this component and deal with them appropriately (setting instance variables, etc)

add_all_args(*parser*)

Add arguments for all components from which this class derives to the given parser, starting with the class highest up the inheritance chain (most distant ancestor).

process_all_args(*args*)

class westpa.tools.iter_range.IterRangeSelection(*data_manager=None*)

Bases: *WESTToolComponent*

Select and record limits on iterations used in analysis and/or reporting. This class provides both the user-facing command-line options and parsing, and the application-side API for recording limits in HDF5.

HDF5 datasets calculated based on a restricted set of iterations should be tagged with the following attributes:

first_iter

The first iteration included in the calculation.

last_iter
One past the last iteration included in the calculation.

iter_step
Blocking or sampling period for iterations included in the calculation.

add_args(*parser*)
Add arguments specific to this component to the given argparse parser.

process_args(*args*, *override_iter_start=None*, *override_iter_stop=None*, *default_iter_step=1*)
Take argparse-processed arguments associated with this component and deal with them appropriately (setting instance variables, etc)

iter_block_iter()
Return an iterable of (block_start,block_end) over the blocks of iterations selected by `--first-iter/--last-iter/--step-iter`.

n_iter_blocks()
Return the number of blocks of iterations (as returned by `iter_block_iter`) selected by `--first-iter/--last-iter/--step-iter`.

record_data_iter_range(*h5object*, *iter_start=None*, *iter_stop=None*)
Store attributes `iter_start` and `iter_stop` on the given HDF5 object (group/dataset)

record_data_iter_step(*h5object*, *iter_step=None*)
Store attribute `iter_step` on the given HDF5 object (group/dataset).

check_data_iter_range_least(*h5object*, *iter_start=None*, *iter_stop=None*)
Check that the given HDF5 object contains (as denoted by its `iter_start/iter_stop` attributes) data at least for the iteration range specified.

check_data_iter_range_equal(*h5object*, *iter_start=None*, *iter_stop=None*)
Check that the given HDF5 object contains (as denoted by its `iter_start/iter_stop` attributes) data exactly for the iteration range specified.

check_data_iter_step_conformant(*h5object*, *iter_step=None*)
Check that the given HDF5 object contains per-iteration data at an iteration stride suitable for extracting data with the given stride (in other words, the given `iter_step` is a multiple of the stride with which data was recorded).

check_data_iter_step_equal(*h5object*, *iter_step=None*)
Check that the given HDF5 object contains per-iteration data at an iteration stride the same as that specified.

slice_per_iter_data(*dataset*, *iter_start=None*, *iter_stop=None*, *iter_step=None*, *axis=0*)
Return the subset of the given dataset corresponding to the given iteration range and stride. Unless otherwise specified, the first dimension of the dataset is the one sliced.

iter_range(*iter_start=None*, *iter_stop=None*, *iter_step=None*, *dtype=None*)
Return a sequence for the given iteration numbers and stride, filling in missing values from those stored on `self`. The smallest data type capable of holding `iter_stop` is returned unless otherwise specified using the `dtype` argument.

6.4.7 westpa.tools.kinetics_tool module

class westpa.tools.kinetics_tool.WESTDataReader

Bases: *WESTToolComponent*

Tool for reading data from WEST-related HDF5 files. Coordinates finding the main HDF5 file from west.cfg or command line arguments, caching of certain kinds of data (eventually), and retrieving auxiliary data sets from various places.

add_args(*parser*)

Add arguments specific to this component to the given argparse parser.

process_args(*args*)

Take argparse-processed arguments associated with this component and deal with them appropriately (setting instance variables, etc)

open(*mode='r'*)

close()

property weight_dsspec

property parent_id_dsspec

class westpa.tools.kinetics_tool.IterRangeSelection(*data_manager=None*)

Bases: *WESTToolComponent*

Select and record limits on iterations used in analysis and/or reporting. This class provides both the user-facing command-line options and parsing, and the application-side API for recording limits in HDF5.

HDF5 datasets calculated based on a restricted set of iterations should be tagged with the following attributes:

first_iter

The first iteration included in the calculation.

last_iter

One past the last iteration included in the calculation.

iter_step

Blocking or sampling period for iterations included in the calculation.

add_args(*parser*)

Add arguments specific to this component to the given argparse parser.

process_args(*args, override_iter_start=None, override_iter_stop=None, default_iter_step=1*)

Take argparse-processed arguments associated with this component and deal with them appropriately (setting instance variables, etc)

iter_block_iter()

Return an iterable of (block_start,block_end) over the blocks of iterations selected by --first-iter/--last-iter/--step-iter.

n_iter_blocks()

Return the number of blocks of iterations (as returned by iter_block_iter) selected by --first-iter/--last-iter/--step-iter.

record_data_iter_range(*h5object, iter_start=None, iter_stop=None*)

Store attributes iter_start and iter_stop on the given HDF5 object (group/dataset)

record_data_iter_step(*h5object, iter_step=None*)

Store attribute iter_step on the given HDF5 object (group/dataset).

check_data_iter_range_least(*h5object*, *iter_start=None*, *iter_stop=None*)

Check that the given HDF5 object contains (as denoted by its *iter_start/iter_stop* attributes) data at least for the iteration range specified.

check_data_iter_range_equal(*h5object*, *iter_start=None*, *iter_stop=None*)

Check that the given HDF5 object contains (as denoted by its *iter_start/iter_stop* attributes) data exactly for the iteration range specified.

check_data_iter_step_conformant(*h5object*, *iter_step=None*)

Check that the given HDF5 object contains per-iteration data at an iteration stride suitable for extracting data with the given stride (in other words, the given *iter_step* is a multiple of the stride with which data was recorded).

check_data_iter_step_equal(*h5object*, *iter_step=None*)

Check that the given HDF5 object contains per-iteration data at an iteration stride the same as that specified.

slice_per_iter_data(*dataset*, *iter_start=None*, *iter_stop=None*, *iter_step=None*, *axis=0*)

Return the subset of the given dataset corresponding to the given iteration range and stride. Unless otherwise specified, the first dimension of the dataset is the one sliced.

iter_range(*iter_start=None*, *iter_stop=None*, *iter_step=None*, *dtype=None*)

Return a sequence for the given iteration numbers and stride, filling in missing values from those stored on self. The smallest data type capable of holding *iter_stop* is returned unless otherwise specified using the *dtype* argument.

class westpa.tools.kinetics_tool.WESTSubcommand(*parent*)

Bases: [WESTToolComponent](#)

Base class for command-line tool subcommands. A little sugar for making this more uniform.

subcommand = None

help_text = None

description = None

add_to_subparsers(*subparsers*)

go()

property work_manager

The work manager for this tool. Raises AttributeError if this is not a parallel tool.

class westpa.tools.kinetics_tool.ProgressIndicatorComponent

Bases: [WESTToolComponent](#)

add_args(*parser*)

Add arguments specific to this component to the given argparse parser.

process_args(*args*)

Take argparse-processed arguments associated with this component and deal with them appropriately (setting instance variables, etc)

westpa.tools.kinetics_tool.**generate_future**(*work_manager*, *name*, *eval_block*, *kwargs*)

class westpa.tools.kinetics_tool.WESTKineticsBase(*parent*)

Bases: [WESTSubcommand](#)

Common argument processing for *w_direct/w_reweight* subcommands. Mostly limited to handling input and output from *w_assign*.

add_args(*parser*)

Add arguments specific to this component to the given argparse parser.

process_args(*args*)

Take argparse-processed arguments associated with this component and deal with them appropriately (setting instance variables, etc)

class westpa.tools.kinetics_tool.**AverageCommands**(*parent*)

Bases: [WESTKineticsBase](#)

default_output_file = 'direct.h5'

add_args(*parser*)

Add arguments specific to this component to the given argparse parser.

process_args(*args*)

Take argparse-processed arguments associated with this component and deal with them appropriately (setting instance variables, etc)

stamp_mcbs_info(*dataset*)

open_files()

open_assignments()

print_averages(*dataset, header, dim=1*)

run_calculation(*pi, nstates, start_iter, stop_iter, step_iter, dataset, eval_block, name, dim, do_averages=False, **extra*)

6.4.8 westpa.tools.plot module

class westpa.tools.plot.**Plotter**(*h5file, h5key, iteration=-1, interface='matplotlib'*)

Bases: `object`

This is a semi-generic plotting interface that has a built in curses based terminal plotter. It's fairly specific to what we're using it for here, but we could (and maybe should) build it out into a little library that we can use via the command line to plot things. Might be useful for looking at data later. That would also cut the size of this tool down by a good bit.

plot(*i=0, j=1, tau=1, iteration=None, dim=0, interface=None*)

6.4.9 westpa.tools.progress module

class westpa.tools.progress.**ProgressIndicator**(*stream=None, interval=1*)

Bases: `object`

draw_fancy()

draw_simple()

draw()

clear()

property operation

property extent

property progress

new_operation(*operation*, *extent=None*, *progress=0*)

start()

stop()

class westpa.tools.progress.**WESTToolComponent**

Bases: object

Base class for WEST command line tools and components used in constructing tools

include_arg(*argname*)

exclude_arg(*argname*)

set_arg_default(*argname*, *value*)

add_args(*parser*)

Add arguments specific to this component to the given argparse parser.

process_args(*args*)

Take argparse-processed arguments associated with this component and deal with them appropriately (setting instance variables, etc)

add_all_args(*parser*)

Add arguments for all components from which this class derives to the given parser, starting with the class highest up the inheritance chain (most distant ancestor).

process_all_args(*args*)

class westpa.tools.progress.**ProgressIndicatorComponent**

Bases: [*WESTToolComponent*](#)

add_args(*parser*)

Add arguments specific to this component to the given argparse parser.

process_args(*args*)

Take argparse-processed arguments associated with this component and deal with them appropriately (setting instance variables, etc)

6.4.10 westpa.tools.selected_segs module

class westpa.tools.selected_segs.**WESTToolComponent**

Bases: object

Base class for WEST command line tools and components used in constructing tools

include_arg(*argname*)

exclude_arg(*argname*)

set_arg_default(*argname*, *value*)

add_args(*parser*)

Add arguments specific to this component to the given argparse parser.

process_args(*args*)

Take argparse-processed arguments associated with this component and deal with them appropriately (setting instance variables, etc)

add_all_args(*parser*)

Add arguments for all components from which this class derives to the given parser, starting with the class highest up the inheritance chain (most distant ancestor).

process_all_args(*args*)

westpa.tools.selected_segs.seg_id_dtype

alias of int64

class westpa.tools.selected_segs.SegmentSelection(*iterable=None*)

Bases: `object`

Initialize this segment selection from an iterable of (n_iter,seg_id) pairs.

add(*pair*)

from_iter(*n_iter*)

property start_iter

property stop_iter

classmethod from_text(*filename*)

class westpa.tools.selected_segs.AllSegmentSelection(*start_iter=None, stop_iter=None, data_manager=None*)

Bases: [`SegmentSelection`](#)

Initialize this segment selection from an iterable of (n_iter,seg_id) pairs.

add(*pair*)

from_iter(*n_iter*)

class westpa.tools.selected_segs.SegSelector

Bases: [`WESTToolComponent`](#)

add_args(*parser*)

Add arguments specific to this component to the given argparse parser.

process_args(*args*)

Take argparse-processed arguments associated with this component and deal with them appropriately (setting instance variables, etc)

parse_segsel_file(*filename*)

6.4.11 westpa.tools.wipi module

```
class westpa.tools.wipi.Plotter(h5file, h5key, iteration=-1, interface='matplotlib')
```

Bases: object

This is a semi-generic plotting interface that has a built in curses based terminal plotter. It's fairly specific to what we're using it for here, but we could (and maybe should) build it out into a little library that we can use via the command line to plot things. Might be useful for looking at data later. That would also cut the size of this tool down by a good bit.

```
plot(i=0, j=1, tau=1, iteration=None, dim=0, interface=None)
```

```
class westpa.tools.wipi.WIPIDataset(raw, key)
```

Bases: object

```
keys()
```

```
class westpa.tools.wipi.KineticsIteration(kin_h5file, index, assign, iteration=-1)
```

Bases: object

```
keys()
```

```
class westpa.tools.wipi.WIPIScheme(scheme, name, parent, settings)
```

Bases: object

```
property scheme
```

```
property list_schemes
```

Lists what schemes are configured in west.cfg file. Schemes should be structured as follows, in west.cfg:

```
west:
    system:
        analysis:
            directory: analysis analysis_schemes:
                scheme.1:
                    enabled: True states:
                        • label: unbound coords: [[7.0]]
                        • label: bound coords: [[2.7]]
                    bins:
                        • type: RectilinearBinMapper boundaries: [[0.0, 2.80, 7, 10000]]
```

```
property iteration
```

```
property assign
```

```
property direct
```

The output from w_direct.py from the current scheme.

```
property state_labels
```

```
property bin_labels
```

```
property west
```

property reweight

property current

The current iteration. See help for `__get_data_for_iteration__`

property past

The previous iteration. See help for `__get_data_for_iteration__`

6.5 Other Packages

6.5.1 westpa.fasthist package

6.5.1.1 Module contents

`westpa.fasthist.histnd(values, binbounds, weights=1.0, out=None, binbound_check=True, ignore_out_of_range=False)`

Generate an N-dimensional PDF (or contribution to a PDF) from the given values. `binbounds` is a list of arrays of boundary values, with one entry for each dimension (`values` must have as many columns as there are entries in `binbounds`) `weight`, if provided, specifies the weight each value contributes to the histogram; this may be a scalar (for equal weights for all values) or a vector of the same length as `values` (for unequal weights). If `binbound_check` is `True`, then the boundaries are checked for strict positive monotonicity; set to `False` to shave a few microseconds if you know your bin boundaries to be monotonically increasing.

`westpa.fasthist.normhistnd(hist, binbounds)`

Normalize the N-dimensional histogram `hist` with corresponding bin boundaries `binbounds`. Modifies `hist` in place and returns the normalization factor used.

6.5.2 westpa.mclib package

6.5.2.1 Module contents

A package for performing Monte Carlo bootstrap estimates of statistics.

`westpa.mclib.mcbs_correltime(dataset, alpha, n_sets=None)`

Calculate the correlation time of the given `dataset`, significant to the $(1-\alpha)$ level, using the method described in Huber & Kim, “Weighted-ensemble Brownian dynamics simulations for protein association reactions” (1996), doi:10.1016/S0006-3495(96)79552-8. An appropriate balance between space and speed is chosen based on the size of the input data.

Returns 0 for data statistically uncorrelated with $(1-\alpha)$ confidence, otherwise the correlation length. (Thus, the appropriate stride for blocking is the result of this function plus one.)

`westpa.mclib.get_bssize(alpha)`

Return a bootstrap data set size appropriate for the given confidence level.

`westpa.mclib.mcbs_ci(dataset, estimator, alpha, dlen, n_sets=None, args=None, kwargs=None, sort=<function msort>)`

Perform a Monte Carlo bootstrap estimate for the $(1-\alpha)$ confidence interval on the given `dataset` with the given `estimator`. This routine is not appropriate for time-correlated data.

Returns (`estimate`, `ci_lb`, `ci_ub`) where `estimate` is the application of the given `estimator` to the input `dataset`, and `ci_lb` and `ci_ub` are the lower and upper limits, respectively, of the $(1-\alpha)$ confidence interval on `estimate`.

estimator is called as **estimator(dataset, *args, **kwargs)**. Common estimators include:

- `numpy.mean` – calculate the confidence interval on the mean of dataset
- `numpy.median` – calculate a confidence interval on the median of dataset
- `numpy.std` – calculate a confidence interval on the standard deviation of dataset.

`n_sets` is the number of synthetic data sets to generate using the given estimator, which will be chosen using ``get_bssize()`` if `n_sets` is not given.

`sort` can be used to override the sorting routine used to calculate the confidence interval, which should only be necessary for estimators returning vectors rather than scalars.

```
westpa.mclib.mcbs_ci_correl(estimator_datasets, estimator, alpha, n_sets=None, args=None,
                           autocorrel_alpha=None, autocorrel_n_sets=None, subsample=None,
                           do_correl=True, mcbs_enable=None, estimator_kwargs={})
```

Perform a Monte Carlo bootstrap estimate for the $(1-\alpha)$ confidence interval on the given dataset with the given estimator. This routine is appropriate for time-correlated data, using the method described in Huber & Kim, “Weighted-ensemble Brownian dynamics simulations for protein association reactions” (1996), doi:10.1016/S0006-3495(96)79552-8 to determine a statistically-significant correlation time and then reducing the dataset by a factor of that correlation time before running a “classic” Monte Carlo bootstrap.

Returns (estimate, ci_lb, ci_ub, correl_time) where `estimate` is the application of the given estimator to the input dataset, `ci_lb` and `ci_ub` are the lower and upper limits, respectively, of the $(1-\alpha)$ confidence interval on estimate, and `correl_time` is the correlation time of the dataset, significant to $(1-\text{autocorrel_alpha})$.

estimator is called as **estimator(dataset, *args, **kwargs)**. Common estimators include:

- `np.mean` – calculate the confidence interval on the mean of dataset
- `np.median` – calculate a confidence interval on the median of dataset
- `np.std` – calculate a confidence interval on the standard deviation of dataset.

`n_sets` is the number of synthetic data sets to generate using the given estimator, which will be chosen using ``get_bssize()`` if `n_sets` is not given.

`autocorrel_alpha` (which defaults to `alpha`) can be used to adjust the significance level of the autocorrelation calculation. Note that too high a significance level (too low an `alpha`) for evaluating the significance of autocorrelation values can result in a failure to detect correlation if the autocorrelation function is noisy.

The given `subsample` function is used, if provided, to subsample the dataset prior to running the full Monte Carlo bootstrap. If none is provided, then a random entry from each correlated block is used as the value for that block. Other reasonable choices include `np.mean`, `np.median`, `(lambda x: x[0])` or `(lambda x: x[-1])`. In particular, using `subsample=np.mean` will converge to the block averaged mean and standard error, while accounting for any non-normality in the distribution of the mean.

6.5.3 westpa.trajtree package

6.5.3.1 westpa.trajtree module

```
class westpa.trajtree.TrajTreeSet(segssel=None, data_manager=None)
```

Bases: `_trajtree_base`

`get_roots()`

`get_root_indices()`

`trace_trajectories(visit, get_visitor_state=None, set_visitor_state=None, vargs=None, vkargs=None)`

6.5.3.2 westpa.trajtree.trajtree module

class westpa.trajtree.trajtree.**AllSegmentSelection**(*start_iter=None, stop_iter=None, data_manager=None*)

Bases: *SegmentSelection*

Initialize this segment selection from an iterable of (n_iter,seg_id) pairs.

add(*pair*)

from_iter(*n_iter*)

class westpa.trajtree.trajtree.**trajnode**(*n_iter, seg_id*)

Bases: tuple

Create new instance of trajnode(n_iter, seg_id)

n_iter

Alias for field number 0

seg_id

Alias for field number 1

class westpa.trajtree.trajtree.**TrajTreeSet**(*segssel=None, data_manager=None*)

Bases: *_trajtree_base*

get_roots()

get_root_indices()

trace_trajectories(*visit, get_visitor_state=None, set_visitor_state=None, vargs=None, vkwargs=None*)

class westpa.trajtree.trajtree.**FakeTrajTreeSet**

Bases: *TrajTreeSet*

6.5.4 WESTPA Old Tools

6.5.4.1 westpa.oldtools package

6.5.4.1.1 westpa.oldtools module

6.5.4.1.2 westpa.oldtools.files module

westpa.oldtools.files.**load_npy_or_text**(*filename*)

Load an array from an existing .npz file, or read a text file and convert to a NumPy array. In either case, return a NumPy array. If a pickled NumPy dataset is found, memory-map it read-only. If the specified file does not contain a pickled NumPy array, attempt to read the file using `numpy.loadtxt(filename, **kwargs)`.

6.5.4.1.3 westpa.oldtools.miscfn module

Miscellaneous support functions for WEST and WEST tools

`westpa.oldtools.miscfn.parse_int_list(list_string)`

Parse a simple list consisting of integers or ranges of integers separated by commas. Ranges are specified as min:max, and include the maximum value (unlike Python's `range`). Duplicate values are ignored. Returns the result as a sorted list. Raises `ValueError` if the list cannot be parsed.

6.5.4.2 westpa.oldtools.aframe package

6.5.4.2.1 westpa.oldtools.aframe

WEST Analysis framework – an unholy mess of classes exploiting each other

class `westpa.oldtools.aframe.AnalysisMixin`

Bases: `object`

add_args(*parser*, *upcall=True*)

process_args(*args*, *upcall=True*)

exception `westpa.oldtools.aframe.ArgumentError(*args, **kwargs)`

Bases: `RuntimeError`

class `westpa.oldtools.aframe.WESTAnalysisTool`

Bases: `object`

add_args(*parser*, *upcall=True*)

Add arguments to a parser common to all analyses of this type.

process_args(*args*, *upcall=True*)

open_analysis_backing()

close_analysis_backing()

require_analysis_group(*groupname*, *replace=False*)

class `westpa.oldtools.aframe.IterRangeMixin`

Bases: `AnalysisMixin`

A mixin for limiting the range of data considered for a given analysis. This should go after `DataManagerMixin`

add_args(*parser*, *upcall=True*)

process_args(*args*, *upcall=True*)

check_iter_range()

iter_block_iter()

Return an iterable of (block_first, block_last+1) over the blocks of iterations selected by `–first/–last/–step`. NOTE WELL that the second of the pair follows Python iterator conventions and returns one past the last element of the block.

n_iter_blocks()

Return the number of blocks of iterations (as returned by `iter_block_iter`) selected by `–first/–last/–step`.

record_data_iter_range(*h5object*, *first_iter=None*, *last_iter=None*)

Store attributes *first_iter* and *last_iter* on the given HDF5 object (group/dataset)

record_data_iter_step(*h5object*, *iter_step=None*)

Store attribute *iter_step* on the given HDF5 object (group/dataset).

check_data_iter_range_least(*h5object*, *first_iter=None*, *last_iter=None*)

Check that the given HDF5 object contains (as denoted by its *first_iter*/*last_iter* attributes) at least the data range specified.

check_data_iter_range_equal(*h5object*, *first_iter=None*, *last_iter=None*)

Check that the given HDF5 object contains per-iteration data for exactly the specified iterations (as denoted by the object's *first_iter* and *last_iter* attributes

check_data_iter_step_conformant(*h5object*, *iter_step=None*)

Check that the given HDF5 object contains per-iteration data at an iteration stride suitable for extracting data with the given stride. (In other words, is the given *iter_step* a multiple of the stride with which data was recorded.)

check_data_iter_step_equal(*h5object*, *iter_step=None*)

Check that the given HDF5 object contains per-iteration data at an iteration stride the same as that specified.

slice_per_iter_data(*dataset*, *first_iter=None*, *last_iter=None*, *iter_step=None*, *axis=0*)

Return the subset of the given dataset corresponding to the given iteration range and stride. Unless otherwise specified, the first dimension of the dataset is the one sliced.

iter_range(*first_iter=None*, *last_iter=None*, *iter_step=None*)

class westpa.oldtools.iframe.**WESTDataReaderMixin**

Bases: [*AnalysisMixin*](#)

A mixin for analysis requiring access to the HDF5 files generated during a WEST run.

add_args(*parser*, *upcall=True*)

process_args(*args*, *upcall=True*)

clear_run_cache()

property **cache_pcoords**

Whether or not to cache progress coordinate data. While caching this data can significantly speed up some analysis operations, this requires copious RAM.

Setting this to False when it was formerly True will release any cached data.

get_summary_table()

get_iter_group(*n_iter*)

Return the HDF5 group corresponding to *n_iter*

get_segments(*n_iter*, *include_pcoords=True*)

Return all segments present in iteration *n_iter*

get_segments_by_id(*n_iter*, *seg_ids*, *include_pcoords=True*)

Get segments from the data manager, employing caching where possible

get_children(*segment*, *include_pcoords=True*)

get_seg_index(*n_iter*)

get_wtg_parent_array(*n_iter*)

get_parent_array(*n_iter*)

get_pcoord_array(*n_iter*)

get_pcoord_dataset(*n_iter*)

get_pcoords(*n_iter*, *seg_ids*)

get_seg_ids(*n_iter*, *bool_array*=None)

get_created_seg_ids(*n_iter*)

Return a list of *seg_ids* corresponding to segments which were created for the given iteration (are not continuations).

max_iter_segs_in_range(*first_iter*, *last_iter*)

Return the maximum number of segments present in any iteration in the range selected

total_segs_in_range(*first_iter*, *last_iter*)

Return the total number of segments present in all iterations in the range selected

get_pcoord_len(*n_iter*)

Get the length of the progress coordinate array for the given iteration.

get_total_time(*first_iter*=None, *last_iter*=None, *dt*=None)

Return the total amount of simulation time spanned between *first_iter* and *last_iter* (inclusive).

class westpa.oldtools.iframe.**ExtDataReaderMixin**

Bases: [AnalysisMixin](#)

An external data reader, primarily designed for reading brute force data, but also suitable for any auxiliary datasets required for analysis.

default_chunksize = 8192

add_args(*parser*, *upcall*=True)

process_args(*args*, *upcall*=True)

is_npy(*filename*)

load_npy_or_text(*filename*)

Load an array from an existing .npz file, or read a text file and convert to a NumPy array. In either case, return a NumPy array. If a pickled NumPy dataset is found, memory-map it read-only. If the specified file does not contain a pickled NumPy array, attempt to read the file using `numpy.loadtxt(filename)`.

text_to_h5dataset(*fileobj*, *group*, *dsname*, *dtype*=<class 'numpy.float64'>, *skiprows*=0, *usecols*=None, *chunksize*=None)

Read text-format data from the given filename or file-like object *fileobj* and write to a newly-created dataset called *dsname* in the HDF5 group *group*. The data is stored as type *dtype*. By default, the shape is taken as (number of lines, number of columns); columns can be omitted by specifying a list for *usecols*, and lines can be skipped by using *skiprows*. Data is read in chunks of *chunksize* rows.

npz_to_h5dataset(*array*, *group*, *dsname*, *usecols*=None, *chunksize*=None)

Store the given array into a newly-created dataset named *dsname* in the HDF5 group *group*, optionally only storing a subset of columns. Data is written *chunksize* rows at a time, allowing very large memory-mapped arrays to be copied.

class westpa.oldtools.iframe.BFDataManager

Bases: [AnalysisMixin](#)

A class to manage brute force trajectory data. The primary purpose is to read in and manage brute force progress coordinate data for one or more trajectories. The trajectories need not be the same length, but they do need to have the same time spacing for progress coordinate values.

traj_index_dtype = dtype([('pcoord_len', '<u8'), ('source_data', 'O')])

add_args(parser, upcall=True)

process_args(args, upcall=True)

update_traj_index(traj_id, pcoord_len, source_data)

get_traj_group(traj_id)

create_traj_group()

get_n_trajs()

get_traj_len(traj_id)

get_max_traj_len()

get_pcoord_array(traj_id)

get_pcoord_dataset(traj_id)

require_bf_h5file()

close_bf_h5file()

class westpa.oldtools.iframe.BinningMixin

Bases: [AnalysisMixin](#)

A mixin for performing binning on WEST data.

add_args(parser, upcall=True)

process_args(args, upcall=True)

mapper_from_expr(expr)

write_bin_labels(dest, header='# bin labels:\n', format='# bin {bin_index:{max_iwidth}d} -- {label!s}\n')

Print labels for all bins in self.mapper to dest. If provided, header is printed before any labels. The format string specifies how bin labels are to be printed. Valid entries are:

- `bin_index` – the zero-based index of the bin
- `label` – the label, as obtained by `bin.label`
- `max_iwidth` – the maximum width (in characters) of the bin index, for pretty alignment

require_binning_group()

delete_binning_group()

record_data_binhash(h5object)

Record the identity hash for self.mapper as an attribute on the given HDF5 object (group or dataset)

check_data_binhash(*h5object*)

Check whether the recorded bin identity hash on the given HDF5 object matches the identity hash for self.mapper

assign_to_bins()

Assign WEST segment data to bins. Requires the DataReader mixin to be in the inheritance tree

require_bin_assignments()

get_bin_assignments(*first_iter=None, last_iter=None*)

get_bin_populations(*first_iter=None, last_iter=None*)

class westpa.oldtools.iframe.MCBSMixin

Bases: [AnalysisMixin](#)

add_args(*parser, upcall=True*)

process_args(*args, upcall=True*)

calc_mcbs_nsets(*alpha=None*)

calc_ci_bound_indices(*n_sets=None, alpha=None*)

class westpa.oldtools.iframe.TrajWalker(*data_reader, history_chunksize=100*)

Bases: object

A class to perform analysis by walking the trajectory tree. A stack is used rather than recursion, or else the highest number of iterations capable of being considered would be the same as the Python recursion limit.

trace_to_root(*n_iter, seg_id*)

Trace the given segment back to its starting point, returning a list of Segment objects describing the entire trajectory.

get_trajectory_roots(*first_iter, last_iter, include_pcoords=True*)

Get segments which start new trajectories. If min_iter or max_iter is specified, restrict the set of iterations within which the search is conducted.

get_initial_nodes(*first_iter, last_iter, include_pcoords=True*)

Get segments with which to begin a tree walk – those alive or created within [first_iter,last_iter].

trace_trajectories(*first_iter, last_iter, callable, include_pcoords=True, cargs=None, ckwards=None, get_state=None, set_state=None*)

Walk the trajectory tree depth-first, calling

callable(segment, children, history, *cargs, **ckwards) for each segment visited. segment is the segment being visited, children is that segment's children, history is the chain of segments leading to segment (not including segment). get_state and set_state are used to record and reset, respectively, any state specific to callable when a new branch is traversed.

class westpa.oldtools.iframe.TransitionAnalysisMixin

Bases: [AnalysisMixin](#)

require_transitions_group()

delete_transitions_group()

get_transitions_ds()

```
    add_args(parser, upcall=True)
    process_args(args, upcall=True)
    require_transitions()
    find_transitions()

class westpa.oldtools.iframe.TransitionEventAccumulator(n_bins, output_group, calc_fpts=True)
    Bases: object
    index_dtype
        alias of uint64
    count_dtype
        alias of uint64
    weight_dtype
        alias of float64
    output_tdat_chunksize = 4096
    tdat_buffersize = 524288
    max_acc = 32768
    clear()
    clear_state()
    get_state()
    set_state(state_dict)
    record_transition_data(tdat)
        Update running statistics and write transition data to HDF5 (with buffering)
    flush_transition_data()
        Flush any unwritten output that may be present
    start_accumulation(assignments, weights, bin_pops, traj=0, n_iter=0)
    continue_accumulation(assignments, weights, bin_pops, traj=0, n_iter=0)

class westpa.oldtools.iframe.BFTTransitionAnalysisMixin
    Bases: TransitionAnalysisMixin
    require_transitions()
    find_transitions(chunksize=65536)

class westpa.oldtools.iframe.KineticsAnalysisMixin
    Bases: AnalysisMixin
    add_args(parser, upcall=True)
    process_args(args, upcall=True)
    parse_bin_range(range_string)
```

check_bin_selection(*n_bins=None*)

Check to see that the bin ranges selected by the user conform to the available bins (i.e., bin indices are within the permissible range). Also assigns the complete bin range if the user has not explicitly limited the bins to be considered.

property **selected_bin_pair_iter**

class westpa.oldtools.iframe.**CommonOutputMixin**

Bases: [AnalysisMixin](#)

add_common_output_args(*parser_or_group*)

process_common_output_args(*args*)

class westpa.oldtools.iframe.**PlottingMixin**

Bases: [AnalysisMixin](#)

require_matplotlib()

6.5.4.2.2 westpa.oldtools.iframe.atool module

class westpa.oldtools.iframe.atool.**WESTAnalysisTool**

Bases: object

add_args(*parser, upcall=True*)

Add arguments to a parser common to all analyses of this type.

process_args(*args, upcall=True*)

open_analysis_backing()

close_analysis_backing()

require_analysis_group(*groupname, replace=False*)

6.5.4.2.3 westpa.oldtools.iframe.base_mixin module

exception westpa.oldtools.iframe.base_mixin.**ArgumentError**(**args*, ***kwargs*)

Bases: RuntimeError

class westpa.oldtools.iframe.base_mixin.**AnalysisMixin**

Bases: object

add_args(*parser, upcall=True*)

process_args(*args, upcall=True*)

6.5.4.2.4 westpa.oldtools.iframe.binning module

class westpa.oldtools.iframe.binning.**AnalysisMixin**

Bases: object

add_args(*parser*, *upcall=True*)

process_args(*args*, *upcall=True*)

class westpa.oldtools.iframe.binning.**BinningMixin**

Bases: [AnalysisMixin](#)

A mixin for performing binning on WEST data.

add_args(*parser*, *upcall=True*)

process_args(*args*, *upcall=True*)

mapper_from_expr(*expr*)

write_bin_labels(*dest*, *header='# bin labels:\n'*, *format='# bin {bin_index:{max_iwidth}d} -- {label!s}\n'*)

Print labels for all bins in `self.mapper` to `dest`. If provided, `header` is printed before any labels. The `format` string specifies how bin labels are to be printed. Valid entries are:

- `bin_index` – the zero-based index of the bin
- `label` – the label, as obtained by `bin.label`
- `max_iwidth` – the maximum width (in characters) of the bin index, for pretty alignment

require_binning_group()

delete_binning_group()

record_data_binhash(*h5object*)

Record the identity hash for `self.mapper` as an attribute on the given HDF5 object (group or dataset)

check_data_binhash(*h5object*)

Check whether the recorded bin identity hash on the given HDF5 object matches the identity hash for `self.mapper`

assign_to_bins()

Assign WEST segment data to bins. Requires the `DataReader` mixin to be in the inheritance tree

require_bin_assignments()

get_bin_assignments(*first_iter=None*, *last_iter=None*)

get_bin_populations(*first_iter=None*, *last_iter=None*)

6.5.4.2.5 westpa.oldtools.iframe.data_reader module

class westpa.oldtools.iframe.data_reader.**Segment**(*n_iter=None*, *seg_id=None*, *weight=None*,
endpoint_type=None, *parent_id=None*,
wtg_parent_ids=None, *pcoord=None*, *status=None*,
walltime=None, *cputime=None*, *data=None*)

Bases: object

A class wrapping segment data that must be passed through the work manager or data manager. Most fields are self-explanatory. One item worth noting is that a negative parent ID means that the segment starts from the initial state with ID `-(segment.parent_id+1)`


```

SEG_STATUS_UNSET = 0

SEG_STATUS_PREPARED = 1

SEG_STATUS_COMPLETE = 2

SEG_STATUS_FAILED = 3

SEG_INITPOINT_UNSET = 0

SEG_INITPOINT_CONTINUES = 1

SEG_INITPOINT_NEWTRAJ = 2

SEG_ENDPOINT_UNSET = 0

SEG_ENDPOINT_CONTINUES = 1

SEG_ENDPOINT_MERGED = 2

SEG_ENDPOINT_RECYCLED = 3

statuses = {'SEG_STATUS_COMPLETE': 2, 'SEG_STATUS_FAILED': 3, 'SEG_STATUS_PREPARED':
1, 'SEG_STATUS_UNSET': 0}

initpoint_types = {'SEG_INITPOINT_CONTINUES': 1, 'SEG_INITPOINT_NEWTRAJ': 2,
'SEG_INITPOINT_UNSET': 0}

endpoint_types = {'SEG_ENDPOINT_CONTINUES': 1, 'SEG_ENDPOINT_MERGED': 2,
'SEG_ENDPOINT_RECYCLED': 3, 'SEG_ENDPOINT_UNSET': 0}

status_names = {0: 'SEG_STATUS_UNSET', 1: 'SEG_STATUS_PREPARED', 2:
'SEG_STATUS_COMPLETE', 3: 'SEG_STATUS_FAILED'}

initpoint_type_names = {0: 'SEG_INITPOINT_UNSET', 1: 'SEG_INITPOINT_CONTINUES', 2:
'SEG_INITPOINT_NEWTRAJ'}

endpoint_type_names = {0: 'SEG_ENDPOINT_UNSET', 1: 'SEG_ENDPOINT_CONTINUES', 2:
'SEG_ENDPOINT_MERGED', 3: 'SEG_ENDPOINT_RECYCLED'}

static initial_pcoord(segment)
    Return the initial progress coordinate point of this segment.

static final_pcoord(segment)
    Return the final progress coordinate point of this segment.

property initpoint_type

property initial_state_id

property status_text

property endpoint_type_text

class westpa.oldtools.iframe.data_reader.AnalysisMixin
    Bases: object
    add_args(parser, upcall=True)

```

process_args(args, upcall=True)

westpa.oldtools.iframe.data_reader.parse_int_list(list_string)

Parse a simple list consisting of integers or ranges of integers separated by commas. Ranges are specified as min:max, and include the maximum value (unlike Python's `range`). Duplicate values are ignored. Returns the result as a sorted list. Raises `ValueError` if the list cannot be parsed.

class westpa.oldtools.iframe.data_reader.WESTDataReaderMixin

Bases: [*AnalysisMixin*](#)

A mixin for analysis requiring access to the HDF5 files generated during a WEST run.

add_args(parser, upcall=True)

process_args(args, upcall=True)

clear_run_cache()

property cache_pcoords

Whether or not to cache progress coordinate data. While caching this data can significantly speed up some analysis operations, this requires copious RAM.

Setting this to False when it was formerly True will release any cached data.

get_summary_table()

get_iter_group(n_iter)

Return the HDF5 group corresponding to `n_iter`

get_segments(n_iter, include_pcoords=True)

Return all segments present in iteration `n_iter`

get_segments_by_id(n_iter, seg_ids, include_pcoords=True)

Get segments from the data manager, employing caching where possible

get_children(segment, include_pcoords=True)

get_seg_index(n_iter)

get_wtg_parent_array(n_iter)

get_parent_array(n_iter)

get_pcoord_array(n_iter)

get_pcoord_dataset(n_iter)

get_pcoords(n_iter, seg_ids)

get_seg_ids(n_iter, bool_array=None)

get_created_seg_ids(n_iter)

Return a list of `seg_ids` corresponding to segments which were created for the given iteration (are not continuations).

max_iter_segs_in_range(first_iter, last_iter)

Return the maximum number of segments present in any iteration in the range selected

total_segs_in_range(first_iter, last_iter)

Return the total number of segments present in all iterations in the range selected

get_pcoord_len(*n_iter*)

Get the length of the progress coordinate array for the given iteration.

get_total_time(*first_iter=None, last_iter=None, dt=None*)

Return the total amount of simulation time spanned between *first_iter* and *last_iter* (inclusive).

class westpa.oldtools.iframe.data_reader.**ExtDataReaderMixin**

Bases: [AnalysisMixin](#)

An external data reader, primarily designed for reading brute force data, but also suitable for any auxiliary datasets required for analysis.

default_chunksize = 8192

add_args(*parser, upcall=True*)

process_args(*args, upcall=True*)

is_numpy(*filename*)

load_numpy_or_text(*filename*)

Load an array from an existing .npy file, or read a text file and convert to a NumPy array. In either case, return a NumPy array. If a pickled NumPy dataset is found, memory-map it read-only. If the specified file does not contain a pickled NumPy array, attempt to read the file using `numpy.loadtxt(filename)`.

text_to_h5dataset(*fileobj, group, dsname, dtype=<class 'numpy.float64'>, skiprows=0, usecols=None, chunksize=None*)

Read text-format data from the given filename or file-like object *fileobj* and write to a newly-created dataset called *dsname* in the HDF5 group *group*. The data is stored as type *dtype*. By default, the shape is taken as (number of lines, number of columns); columns can be omitted by specifying a list for *usecols*, and lines can be skipped by using *skiprows*. Data is read in chunks of *chunksize* rows.

numpy_to_h5dataset(*array, group, dsname, usecols=None, chunksize=None*)

Store the given array into a newly-created dataset named *dsname* in the HDF5 group *group*, optionally only storing a subset of columns. Data is written *chunksize* rows at a time, allowing very large memory-mapped arrays to be copied.

class westpa.oldtools.iframe.data_reader.**BFDDataManager**

Bases: [AnalysisMixin](#)

A class to manage brute force trajectory data. The primary purpose is to read in and manage brute force progress coordinate data for one or more trajectories. The trajectories need not be the same length, but they do need to have the same time spacing for progress coordinate values.

traj_index_dtype = dtype([('pcoord_len', '<u8'), ('source_data', 'O')])

add_args(*parser, upcall=True*)

process_args(*args, upcall=True*)

update_traj_index(*traj_id, pcoord_len, source_data*)

get_traj_group(*traj_id*)

create_traj_group()

get_n_trajs()

get_traj_len(*traj_id*)

```
get_max_traj_len()
get_pcoord_array(traj_id)
get_pcoord_dataset(traj_id)
require_bf_h5file()
close_bf_h5file()
```

6.5.4.2.6 westpa.oldtools.iframe.iter_range module

```
class westpa.oldtools.iframe.iter_range.AnalysisMixin
```

Bases: object

```
add_args(parser, upcall=True)
```

```
process_args(args, upcall=True)
```

```
exception westpa.oldtools.iframe.iter_range.ArgumentError(*args, **kwargs)
```

Bases: RuntimeError

```
class westpa.oldtools.iframe.iter_range.IterRangeMixin
```

Bases: [AnalysisMixin](#)

A mixin for limiting the range of data considered for a given analysis. This should go after DataManagerMixin

```
add_args(parser, upcall=True)
```

```
process_args(args, upcall=True)
```

```
check_iter_range()
```

```
iter_block_iter()
```

Return an iterable of (block_first, block_last+1) over the blocks of iterations selected by –first/–last/–step. NOTE WELL that the second of the pair follows Python iterator conventions and returns one past the last element of the block.

```
n_iter_blocks()
```

Return the number of blocks of iterations (as returned by iter_block_iter) selected by –first/–last/–step.

```
record_data_iter_range(h5object, first_iter=None, last_iter=None)
```

Store attributes first_iter and last_iter on the given HDF5 object (group/dataset)

```
record_data_iter_step(h5object, iter_step=None)
```

Store attribute iter_step on the given HDF5 object (group/dataset).

```
check_data_iter_range_least(h5object, first_iter=None, last_iter=None)
```

Check that the given HDF5 object contains (as denoted by its first_iter/last_iter attributes) at least the data range specified.

```
check_data_iter_range_equal(h5object, first_iter=None, last_iter=None)
```

Check that the given HDF5 object contains per-iteration data for exactly the specified iterations (as denoted by the object's first_iter and last_iter attributes)

check_data_iter_step_conformant(*h5object*, *iter_step=None*)

Check that the given HDF5 object contains per-iteration data at an iteration stride suitable for extracting data with the given stride. (In other words, is the given *iter_step* a multiple of the stride with which data was recorded.)

check_data_iter_step_equal(*h5object*, *iter_step=None*)

Check that the given HDF5 object contains per-iteration data at an iteration stride the same as that specified.

slice_per_iter_data(*dataset*, *first_iter=None*, *last_iter=None*, *iter_step=None*, *axis=0*)

Return the subset of the given dataset corresponding to the given iteration range and stride. Unless otherwise specified, the first dimension of the dataset is the one sliced.

iter_range(*first_iter=None*, *last_iter=None*, *iter_step=None*)

6.5.4.2.7 westpa.oldtools.iframe.kinetics module

class westpa.oldtools.iframe.kinetics.**AnalysisMixin**

Bases: object

add_args(*parser*, *upcall=True*)

process_args(*args*, *upcall=True*)

class westpa.oldtools.iframe.kinetics.**KineticsAnalysisMixin**

Bases: [AnalysisMixin](#)

add_args(*parser*, *upcall=True*)

process_args(*args*, *upcall=True*)

parse_bin_range(*range_string*)

check_bin_selection(*n_bins=None*)

Check to see that the bin ranges selected by the user conform to the available bins (i.e., bin indices are within the permissible range). Also assigns the complete bin range if the user has not explicitly limited the bins to be considered.

property selected_bin_pair_iter

6.5.4.2.8 westpa.oldtools.iframe.mcbs module

Tools for Monte Carlo bootstrap error analysis

class westpa.oldtools.iframe.mcbs.**AnalysisMixin**

Bases: object

add_args(*parser*, *upcall=True*)

process_args(*args*, *upcall=True*)

class westpa.oldtools.iframe.mcbs.**MCBSMixin**

Bases: [AnalysisMixin](#)

add_args(*parser*, *upcall=True*)

```
process_args(args, upcall=True)
```

```
calc_mcbs_nsets(alpha=None)
```

```
calc_ci_bound_indices(n_sets=None, alpha=None)
```

```
westpa.oldtools.iframe.mcbs.calc_mcbs_nsets(alpha)
```

Return a bootstrap data set size appropriate for the given confidence level.

```
westpa.oldtools.iframe.mcbs.calc_ci_bound_indices(n_sets, alpha)
```

```
westpa.oldtools.iframe.mcbs.bootstrap_ci_ll(estimator, data, alpha, n_sets, storage, sort, eargs=(),  
                                             ekwargs={}, fhat=None)
```

Low-level routine for calculating bootstrap error estimates. Arguments and return values are as those for `bootstrap_ci`, except that no argument is optional except additional arguments for the estimator (`eargs`, `ekwargs`). `data` must be an array (or subclass), and an additional array `storage` must be provided, which must be appropriately shaped and typed to hold `n_sets` results from `estimator`. Further, if the value `fhat` of the estimator must be pre-calculated to allocate `storage`, then its value may be passed; otherwise, `estimator(data, *eargs, **kwargs)` will be called to calculate it.

```
westpa.oldtools.iframe.mcbs.bootstrap_ci(estimator, data, alpha, n_sets=None, sort=<function msort>,  
                                           eargs=(), ekwargs={})
```

Perform a Monte Carlo bootstrap of a $(1-\alpha)$ confidence interval for the given estimator. Returns `(fhat, ci_lower, ci_upper)`, where `fhat` is the result of `estimator(data, *eargs, **kwargs)`, and `ci_lower` and `ci_upper` are the lower and upper bounds of the surrounding confidence interval, calculated by calling `estimator(syndata, *eargs, **kwargs)` on each synthetic data set `syndata`. If `n_sets` is provided, that is the number of synthetic data sets generated, otherwise an appropriate size is selected automatically (see `calc_mcbs_nsets()`).

`sort`, if given, is applied to sort the results of calling `estimator` on each synthetic data set prior to obtaining the confidence interval. This function must sort on the last index.

Individual entries in synthetic data sets are selected by the first index of `data`, allowing this function to be used on arrays of multidimensional data.

Returns `(fhat, lb, ub, ub-lb, abs((ub-lb)/fhat), and max(ub-fhat, fhat-lb))` (that is, the estimated value, the lower and upper bounds of the confidence interval, the width of the confidence interval, the relative width of the confidence interval, and the symmetrized error bar of the confidence interval).

6.5.4.2.9 westpa.oldtools.iframe.output module

```
class westpa.oldtools.iframe.output.AnalysisMixin
```

Bases: object

```
add_args(parser, upcall=True)
```

```
process_args(args, upcall=True)
```

```
class westpa.oldtools.iframe.output.CommonOutputMixin
```

Bases: [AnalysisMixin](#)

```
add_common_output_args(parser_or_group)
```

```
process_common_output_args(args)
```

6.5.4.2.10 westpa.oldtools.iframe.plotting module

```
class westpa.oldtools.iframe.plotting.AnalysisMixin
```

Bases: object

```
add_args(parser, upcall=True)
```

```
process_args(args, upcall=True)
```

```
class westpa.oldtools.iframe.plotting.PlottingMixin
```

Bases: [AnalysisMixin](#)

```
require_matplotlib()
```

6.5.4.2.11 westpa.oldtools.iframe.trajwalker module

```
class westpa.oldtools.iframe.trajwalker.TrajWalker(data_reader, history_chunksize=100)
```

Bases: object

A class to perform analysis by walking the trajectory tree. A stack is used rather than recursion, or else the highest number of iterations capable of being considered would be the same as the Python recursion limit.

```
trace_to_root(n_iter, seg_id)
```

Trace the given segment back to its starting point, returning a list of Segment objects describing the entire trajectory.

```
get_trajectory_roots(first_iter, last_iter, include_pcoords=True)
```

Get segments which start new trajectories. If min_iter or max_iter is specified, restrict the set of iterations within which the search is conducted.

```
get_initial_nodes(first_iter, last_iter, include_pcoords=True)
```

Get segments with which to begin a tree walk – those alive or created within [first_iter,last_iter].

```
trace_trajectories(first_iter, last_iter, callable, include_pcoords=True, cargs=None, ckwards=None,
                  get_state=None, set_state=None)
```

Walk the trajectory tree depth-first, calling

callable(segment, children, history, *cargs, **ckwards) for each segment visited. segment is the segment being visited, children is that segment's children, history is the chain of segments leading to segment (not including segment). get_state and set_state are used to record and reset, respectively, any state specific to callable when a new branch is traversed.

6.5.4.2.12 westpa.oldtools.iframe.transitions module

```
class westpa.oldtools.iframe.transitions.AnalysisMixin
```

Bases: object

```
add_args(parser, upcall=True)
```

```
process_args(args, upcall=True)
```

```
class westpa.oldtools.iframe.transitions.TrajWalker(data_reader, history_chunksize=100)
```

Bases: object

A class to perform analysis by walking the trajectory tree. A stack is used rather than recursion, or else the highest number of iterations capable of being considered would be the same as the Python recursion limit.

trace_to_root(*n_iter*, *seg_id*)

Trace the given segment back to its starting point, returning a list of Segment objects describing the entire trajectory.

get_trajectory_roots(*first_iter*, *last_iter*, *include_pcoords=True*)

Get segments which start new trajectories. If *min_iter* or *max_iter* is specified, restrict the set of iterations within which the search is conducted.

get_initial_nodes(*first_iter*, *last_iter*, *include_pcoords=True*)

Get segments with which to begin a tree walk – those alive or created within [*first_iter*,*last_iter*].

trace_trajectories(*first_iter*, *last_iter*, *callable*, *include_pcoords=True*, *cargs=None*, *ckwargs=None*,
get_state=None, *set_state=None*)

Walk the trajectory tree depth-first, calling

callable(*segment*, *children*, *history*, **cargs*, ***kwargs*) for each segment visited. *segment* is the segment being visited, *children* is that segment's children, *history* is the chain of segments leading to *segment* (not including *segment*). *get_state* and *set_state* are used to record and reset, respectively, any state specific to *callable* when a new branch is traversed.

class westpa.oldtools.iframe.transitions.**TransitionEventAccumulator**(*n_bins*, *output_group*,
calc_fpts=True)

Bases: object

index_dtype

alias of uint64

count_dtype

alias of uint64

weight_dtype

alias of float64

output_tdat_chunksize = 4096

tdat_buffersize = 524288

max_acc = 32768

clear()

clear_state()

get_state()

set_state(*state_dict*)

record_transition_data(*tdat*)

Update running statistics and write transition data to HDF5 (with buffering)

flush_transition_data()

Flush any unwritten output that may be present

start_accumulation(*assignments*, *weights*, *bin_pops*, *traj=0*, *n_iter=0*)

continue_accumulation(*assignments*, *weights*, *bin_pops*, *traj=0*, *n_iter=0*)

class westpa.oldtools.iframe.transitions.**TransitionAnalysisMixin**

Bases: [AnalysisMixin](#)


```

require_transitions_group()
delete_transitions_group()
get_transitions_ds()
add_args(parser, upcall=True)
process_args(args, upcall=True)
require_transitions()
find_transitions()

```

```
class westpa.oldtools.iframe.transitions.BFTransitionAnalysisMixin
```

```
    Bases: TransitionAnalysisMixin
```

```

    require_transitions()

    find_transitions(chunksize=65536)

```

6.5.4.3 westpa.oldtools.cmds package

6.5.4.3.1 westpa.oldtools.cmds module

6.5.4.3.2 westpa.oldtools.cmds.w_ttimes module

6.5.4.4 westpa.oldtools.stats package

6.5.4.4.1 westpa.oldtools.stats module

```

class westpa.oldtools.stats.RunningStatsAccumulator(shape, dtype=<class 'numpy.float64'>,
                                                    count_dtype=<class 'numpy.uint64'>,
                                                    weight_dtype=<class 'numpy.float64'>,
                                                    mask_value=nan)

```

```
    Bases: object
```

```

    incorporate(index, value, weight)

    average()

    mean()

    std()

```

6.5.4.4.2 westpa.oldtools.stats.accumulator module

```

class westpa.oldtools.stats.accumulator.RunningStatsAccumulator(shape, dtype=<class
                                                                    'numpy.float64'>,
                                                                    count_dtype=<class
                                                                    'numpy.uint64'>,
                                                                    weight_dtype=<class
                                                                    'numpy.float64'>,
                                                                    mask_value=nan)

```

Bases: object

incorporate(*index, value, weight*)

average()

mean()

std()

6.5.4.4.3 westpa.oldtools.stats.edfs module

class westpa.oldtools.stats.edfs.**EDF**(*values, weights=None*)

Bases: object

A class for creating and manipulating empirical distribution functions (cumulative distribution functions derived from sample data).

Construct a new EDF from the given values and (optionally) weights.

static from_array(*array*)

static from_arrays(*x, F*)

as_array()

Return this EDF as a (N,2) array, where N is the number of unique values passed to the constructor. Numpy type casting rules are applied (so, for instance, integral abscissae are converted to floating-point values).

quantiles(*p*)

Treating the EDF as a quantile function, return the values of the (statistical) variable whose probabilities are at least p. That is, $Q(p) = \inf \{x: p \leq F(x)\}$.

quantile(*p*)

median()

moment(*n*)

Calculate the nth moment of this probability distribution

$\langle x^n \rangle = \int_{-\infty}^{\infty} x^n dF(x)$

cmoment(*n*)

Calculate the nth central moment of this probability distribution

mean()

var()

Return the second central moment of this probability distribution.

std()

Return the standard deviation (root of the variance) of this probability distribution.

6.5.4.4 westpa.oldtools.stats.mcbs module

Tools for Monte Carlo bootstrap error analysis

`westpa.oldtools.stats.mcbs.add_mcbs_options(parser)`

Add arguments concerning Monte Carlo bootstrap (confidence and bssize) to the given parser

`westpa.oldtools.stats.mcbs.get_bssize(alpha)`

Return a bootstrap data set size appropriate for the given confidence level

`westpa.oldtools.stats.mcbs.bootstrap_ci(estimator, data, alpha, n_sets=None, args=(), kwargs={}, sort=<function msort>, extended_output=False)`

Perform a Monte Carlo bootstrap of a $(1-\alpha)$ confidence interval for the given estimator. Returns (fhat, ci_lower, ci_upper), where fhat is the result of `estimator(data, *args, **kwargs)`, and ci_lower and ci_upper are the lower and upper bounds of the surrounding confidence interval, calculated by calling `estimator(syndata, *args, **kwargs)` on each synthetic data set syndata. If n_sets is provided, that is the number of synthetic data sets generated, otherwise an appropriate size is selected automatically (see `get_bssize()`).

sort, if given, is applied to sort the results of calling estimator on each synthetic data set prior to obtaining the confidence interval.

Individual entries in synthetic data sets are selected by the first index of data, allowing this function to be used on arrays of multidimensional data.

If extended_output is True (by default not), instead of returning (fhat, lb, ub), this function returns (fhat, lb, ub, ub-lb, abs((ub-lb)/fhat), and max(ub-fhat, fhat-lb)) (that is, the estimated value, the lower and upper bounds of the confidence interval, the width of the confidence interval, the relative width of the confidence interval, and the symmetrized error bar of the confidence interval).

6.6 westpa.westtext package

6.6.1 Currently Supported

6.6.1.1 westpa.westtext.adaptvoronoi package

6.6.1.1.1 Submodules

6.6.1.1.2 westpa.westtext.adaptvoronoi.adaptVor_driver module

`westpa.westtext.adaptvoronoi.adaptVor_driver.check_bool(value, action='warn')`

Check that the given value is boolean in type. If not, either raise a warning (if action=='warn') or an exception (action=='raise').

exception `westpa.westtext.adaptvoronoi.adaptVor_driver.ConfigItemMissing(key, message=None)`

Bases: `KeyError`

class `westpa.westtext.adaptvoronoi.adaptVor_driver.VoronoiBinMapper(dfunc, centers, dfargs=None, dfkwargs=None)`

Bases: `BinMapper`

A one-dimensional mapper which assigns a multidimensional pcoord to the closest center based on a distance metric. Both the list of centers and the distance function must be supplied.

assign(*coords*, *mask=None*, *output=None*)

class westpa.westext.adaptvoronoi.adaptVor_driver.**AdaptiveVoronoiDriver**(*sim_manager*,
plugin_config)

Bases: object

This plugin implements an adaptive scheme using voronoi bins from Zhang 2010, J Chem Phys, 132. The options exposed to the configuration file are:

- **av_enabled** (bool, default False): Enables adaptive binning
- **max_centers** (int, default 10): The maximum number of voronoi centers to be placed
- **walk_count** (integer, default 5): Number of walkers per voronoi center
- **center_freq** (integer, default 1): Frequency of center placement
- **priority** (integer, default 1): Priority in the plugin order
- **dfunc_method** (**function, non-optional, no default**): **Non-optional user defined** function that will be used to calculate distances between voronoi centers and data points
- **mapper_func** (**function, optional**): **Optional user defined function for building bin** mappers for more complicated binning schemes e.g. embedding the voronoi binning in a portion of the state space. If not defined the plugin will build a VoronoiBinMapper with the information it has.

dfunc()

Distance function to be used by the plugin. This function will be used to calculate the distance between each point.

get_dfunc_method(*plugin_config*)

get_mapper_func(*plugin_config*)

get_initial_centers()

This function pulls from the centers from either the previous bin mapper or uses the definition from the system to calculate the number of centers

update_bin_mapper()

Update the bin_mapper using the current set of voronoi centers

update_centers(*iter_group*)

Update the set of Voronoi centers according to Zhang 2010, J Chem Phys, 132. A short description of the algorithm can be found in the text:

- 1) First reference structure is chosen randomly from the first set of given structure
- 2) Given a set of *n* reference structures, for each configuration in the iteration the distances to each reference structure is calculated and the minimum distance is found
- 3) The configuration with the minimum distance is selected as the next reference

prepare_new_iteration()

6.6.1.1.3 Module contents

class westpa.westext.adaptvoronoi.**AdaptiveVoronoiDriver**(*sim_manager*, *plugin_config*)

Bases: object

This plugin implements an adaptive scheme using voronoi bins from Zhang 2010, J Chem Phys, 132. The options exposed to the configuration file are:

- **av_enabled** (bool, default False): Enables adaptive binning
- **max_centers** (int, default 10): The maximum number of voronoi centers to be placed
- **walk_count** (integer, default 5): Number of walkers per voronoi center
- **center_freq** (integer, default 1): Frequency of center placement
- **priority** (integer, default 1): Priority in the plugin order

- **dfunc_method (function, non-optional, no default): Non-optional user defined**
function that will be used to calculate distances between voronoi centers and data points
- **mapper_func (function, optional): Optional user defined function for building bin**
mappers for more complicated binning schemes e.g. embedding the voronoi binning in a portion of the state space. If not defined the plugin will build a VoronoiBinMapper with the information it has.

dfunc()

Distance function to be used by the plugin. This function will be used to calculate the distance between each point.

get_dfunc_method(plugin_config)

get_mapper_func(plugin_config)

get_initial_centers()

This function pulls from the centers from either the previous bin mapper or uses the definition from the system to calculate the number of centers

update_bin_mapper()

Update the bin_mapper using the current set of voronoi centers

update_centers(iter_group)

Update the set of Voronoi centers according to Zhang 2010, J Chem Phys, 132. A short description of the algorithm can be found in the text:

1) First reference structure is chosen randomly from the first set of given structure 2) Given a set of n reference structures, for each configuration in the iteration the distances to each reference structure is calculated and the minimum distance is found 3) The configuration with the minimum distance is selected as the next reference

prepare_new_iteration()

6.6.1.2 westpa.westext.stringmethod package

6.6.1.2.1 Submodules

6.6.1.2.2 westpa.westext.stringmethod.fourier_fitting module

6.6.1.2.3 westpa.westext.stringmethod.string_driver module

6.6.1.2.4 westpa.westext.stringmethod.string_method module

6.6.1.2.5 Module contents

6.6.1.3 westpa.westext.hamsm_restarting package

6.6.1.3.1 Description

This plugin leverages haMSM analysis [1] to provide simulation post-analysis. This post-analysis can be used on its own, or can be used to initialize and run new WESTPA simulations using structures in the haMSM's best estimate of steady-state as described in [2], which may accelerate convergence to steady-state.

haMSM analysis is performed using the `msm_we` library.

Sample files necessary to run the restarting plugin (as described below) can be found in the [WESTPA GitHub Repo](#).

6.6.1.3.2 Usage

Configuration

`west.cfg`

This plugin requires the following section in `west.cfg` (or whatever your WE configuration file is named):

```
west:
  plugins:
  - plugin: westpa.westext.hamsm_restarting.restart_driver.RestartDriver
    n_restarts: 0          # Number of restarts to perform
    n_runs: 5             # Number of runs within each restart
    n_restarts_to_use: 0.5 # Amount of prior restarts' data to use. -1, a decimal in
    ↪ (0,1), or an integer. Details below.
    extension_iters: 5     # Number of iterations to continue runs for, if target is
    ↪ not reached by first restart period
    coord_len: 2           # Length of pcoords returned
    initialization_file: restart_initialization.json # JSON describing w_run parameters
    ↪ for new runs
    ref_pdb_file: common_files/bstate.pdb          # File containing reference
    ↪ structure/topology
    model_name: NaClFlux                          # Name for msm_we model
    n_clusters: 2                                 # Number of clusters in haMSM
    ↪ building
    we_folder: .                                  # Should point to the same
    ↪ directory as WEST_SIM_ROOT
    target_pcoord_bounds: [[-inf, 2.60]]          # Progress coordinate boundaries
    ↪ for the target state
    basis_pcoord_bounds: [[12.0, inf]]            # Progress coordinate boundaries
    ↪ for the basis state
    tau: 5e-13                                    # Resampling time, i.e. length of
    ↪ a WE iteration in physical units
    pcoord_ndim0: 1                              # Dimensionality of progress
    ↪ coordinate
    dim_reduce_method: pca                       # Dimensionality reduction scheme,
    ↪ either "pca", "vamp", or "none"
    parent_traj_filename: parent.xml             # Name of parent file in each
    ↪ segment
    child_traj_filename: seg.xml                 # Name of child file in each
    ↪ segment
    user_functions: westpa_scripts/restart_overrides.py # Python file defining
    ↪ coordinate processing
    struct_filetype: mdtraj.formats.PDBTrajectoryFile # Filetype for output
    ↪ start-structures
    debug: False                                # Optional, defaults to False. If true, enables debug-mode
    ↪ logging.
    streaming: True                             # Does clustering in a streaming fashion, versus trying to
    ↪ load all coords in memory
    n_cpus: 1                                   # Number of CPUs to use for parallel calculations
```

Some sample parameters are provided in the above, but of course should be modified to your specific system.

Note about `restarts_to_use` : `restarts_to_use` can be specified in a few different ways. A value of `-1`

means to use all available data. A decimal $0 < \text{restarts_to_use} < 1$ will use the last $\text{restarts_to_use} * \text{current_restart}$ iterations of data – so, for example, set to 0.5 to use the last half of the data, or 0.75 to use the last 3/4. Finally, an integer value will just use the last restarts_to_use iterations.

Note that `ref_pdb_file` can be any filetype supported by `msm_we.initialize()`'s structure loading. At the time of writing, this is limited to PDB, however that is planned to be extended. Also at the time of writing, that's only used to set `model.nAtoms`, so if you're using some weird topology that's unsupported, you should be able to scrap that and manually set `nAtoms` on the object.

Also in this file, `west.data.data_refs.basis_state` MUST point to `$WEST_SIM_ROOT/{basis_state.auxref}` and not a subdirectory if restarts are being used. This is because when the plugin initiates a restart, `start_state` references in `$WEST_SIM_ROOT/restartXX/start_states.txt` are set relative to `$WEST_SIM_ROOT`. All basis/start state references are defined relative to `west.data.data_refs.basis_state`, so if that points to a subdirectory of `$WEST_SIM_ROOT`, those paths will not be accurate.

Running

Once configured, just run your WESTPA simulation normally with `w_run`, and the plugin will automatically handle performing restarts, and extensions if necessary.

6.6.1.3.3 Extensions

To be clear: these are extensions in the sense of extending a simulation to be longer – not in the sense of “an extension to the WESTPA software package”!

Running with `extension_iters` greater than 0 will enable extensions before the first restart if the target state is not reached. This is useful to avoid restarting when you don't yet have structures spanning all the way from your basis to target. At the time of writing, it's not yet clear whether restarting from “incomplete” WE runs like this will help or hinder the total number of iterations it takes to reach the target.

Extensions are simple and work as follows: before doing the first restart, after all runs are complete, the output WESTPA h5 files are scanned to see if any recycling has occurred. If it hasn't, then each run is extended by `extension_iters` iterations.

`restart_initialization.json`

```
{
  "bstates": ["start,1,bstates/bstate.pdb"],
  "tstates": ["bound,2.6"],
  "bstate-file": "bstates/bstates.txt",
  "tstate-file": "tstate.file",
  "segs-per-state": 1
}
```

It is not necessary to specify both in-line states and a state-file for each, but that is shown in the sample for completeness.

It is important that `bstates` and `tstates` are lists of strings, and not just strings, even if only one `bstate`/`tstate` is being used!

With `n_runs > 1`, before doing any restart, multiple independent runs are performed. However, before the first restart (this applies if no restarts are performed as well), the plugin has no way of accessing the parameters that were initially passed to `w_init` and `w_run`.

Therefore, it is necessary to store those parameters in a file, so the plugin can read them and initiate subsequent runs.

After the first restart is performed, the plugin writes this file itself, so it is only necessary to manually configure for that first set of runs.

Featurization overrides

```
import numpy as np
import mdtraj as md

def processCoordinates(self, coords):
    log.debug("Processing coordinates")

    if self.dimReduceMethod == "none":
        nC = np.shape(coords)
        nC = nC[0]
        ndim = 3 * self.nAtoms
        data = coords.reshape(nC, 3 * self.nAtoms)
        return data

    if self.dimReduceMethod == "pca" or self.dimReduceMethod == "vamp":
        ### NaCl RMSD dimensionality reduction
        log.warning("Hardcoded selection: Doing dim reduction for Na, Cl. This is_
↳only for testing!")
        indNA = self.reference_structure.topology.select("element Na")
        indCL = self.reference_structure.topology.select("element Cl")

        diff = np.subtract(coords[:, indNA], coords[:, indCL])

        dist = np.array(np.sqrt(
            np.mean(
                np.power(
                    diff,
                    2)
                , axis=-1)
            ))

        return dist
```

This is the file whose path is provided in the configuration file in `plugin.user_functions`, and must be a Python file defining a function named `processCoordinates(self, coords)` which takes a numpy array of coordinates, featurizes it, and returns the numpy array of feature-coordinates.

This is left to be user-provided because whatever featurization you do will be system-specific. The provided function is monkey-patched into the `msm_we.modelWE` class.

An example is provided above, which does a simple RMSD coordinate reduction for the NaCl association tutorial system.

Doing only post-analysis

If you want to ONLY use this for haMSM post-analysis, and not restarting, just set `n_restarts: 0` in the configuration.

Work manager for restarting

If you're using some parallelism (which you should), and you're using the plugin to do restarts or multiple runs, then your choice of work manager can be important. This plugin handles starting new WESTPA runs using the Python API. The process work manager, by default, uses `fork` to start new workers which seems to eventually causes memory issues, since `fork` passes the entire contents of the parent to each child. Switching the spawn method to `forkserver` or `spawn` may introduce other issues.

Using the ZMQ work manager works well. The MPI work manager should also work well, though is untested. Both of these handle starting new workers in a more efficient way, without copying the full state of the parent.

Continuing a failed run

The restarting plugin has a few different things it expects to find when it runs. Crashes during the WE run should not affect this. However, if the plugin itself crashes while running, these may be left in a weird state.

If the plugin crashes while running, make sure:

- `restart.dat` contains the correct entries. `restarts_completed` is the number of restarts *successfully* completed, and same for `runs_completed` within that restart.
- `restart_initialization.json` is pointing to the correct restart

It may help to `w_truncate` the very last iteration and allow WESTPA to re-do it.

Potential Pitfalls/Troubleshooting

- Basis state calculation may take a LONG time with a large number of start-states. A simple RMSD calculation using `cpptraj` and 500,000 start-states took over 6 hours. Reducing the number of runs used through `n_restarts_to_use` will ameliorate this.
- If `restart_driver.prepare_coordinates()` has written a coordinate for an iteration, subsequent runs will NOT overwrite it, and will skip it.
- In general: verify that `msm_we` is installed
- Verify that `restart_initialization.json` has been correctly set
- This plugin does not yet attempt to resolve environment variables in the config, so things like say, `$WEST_SIM_ROOT`, will be interpreted literally in paths

References

- [1] Suárez, E., Adelman, J. L. & Zuckerman, D. M. Accurate Estimation of Protein Folding and Unfolding Times: Beyond Markov State Models. *J Chem Theory Comput* 12, 3473–3481 (2016).
- [2] Copperman, J. & Zuckerman, D. M. Accelerated Estimation of Long-Timescale Kinetics from Weighted Ensemble Simulation via Non-Markovian “Microbin” Analysis. *J Chem Theory Comput* 16, 6763–6775 (2020).

6.6.2 Deprecated

6.6.2.1 westpa.westext.weed package

6.6.2.1.1 Submodules

6.6.2.1.2 westpa.westext.weed.BinCluster module

class westpa.westext.weed.BinCluster.**ClusterList**(*ratios, nbins*)

Bases: object

join(*pairs*)

Join clusters given a tuple (i,j) of bin pairs

join_simple(*pairs*)

Join clusters using direct ratios given a tuple (i,j) of bin pairs

6.6.2.1.3 westpa.westext.weed.ProbAdjustEquil module

westpa.westext.weed.ProbAdjustEquil.**probAdjustEquil**(*binProb, rates, uncert, threshold=0.0, fullCalcClust=False, fullCalcBins=False*)

This function adjusts bin pops in binProb using rates and uncert matrices fullCalcBins → True for weighted avg, False for simple calc fullCalcClust → True for weighted avg, False for simple calc threshold → minimum weight (relative to max) for another value to be averaged
only matters if fullCalcBins == True (or later perhaps if fullCalcClust == True)

6.6.2.1.4 westpa.westext.weed.UncertMath module

class westpa.westext.weed.UncertMath.**UncertContainer**(*vals, vals_dmin, vals_dmax, mask=False*)

Bases: object

Container to hold uncertainty measurements. Data is convert to np masked arrays to avoid possible numerical problems

transpose()

recip()

update_mask()

concatenate(*value, axis=0*)

Concatentate UncertContainer value to self. Assumes that if dimensions of self and value do not match, to add a np.newaxis along axis of value

weighted_average(*axis=0, expaxis=None*)

Calculate weighted average of data along axis after optionally inserting a new dimension into the shape array at position expaxis

6.6.2.1.5 westpa.westext.weed.weed_driver module

westpa.westext.weed.weed_driver.check_bool(*value, action='warn'*)

Check that the given value is boolean in type. If not, either raise a warning (if *action*== 'warn') or an exception (if *action*== 'raise').

class westpa.westext.weed.weed_driver.RateAverager(*bin_mapper, system=None, data_manager=None, work_manager=None*)

Bases: object

Calculate bin-to-bin kinetic properties (fluxes, rates, populations) at 1-tau resolution

extract_data(*iter_indices*)

Extract data from the data_manger and place in dict mirroring the same underlying layout.

task_generator(*iter_start, iter_stop, block_size*)

calculate(*iter_start=None, iter_stop=None, n_blocks=1, queue_size=1*)

Read the HDF5 file and collect flux matrices and population vectors for each bin for each iteration in the range [iter_start, iter_stop). Break the calculation into n_blocks blocks. If the calculation is broken up into more than one block, queue_size specifies the maximum number of tasks in the work queue.

westpa.westext.weed.weed_driver.probAdjustEquil(*binProb, rates, uncert, threshold=0.0, fullCalcClust=False, fullCalcBins=False*)

This function adjusts bin pops in binProb using rates and uncert matrices fullCalcBins → True for weighted avg, False for simple calc fullCalcClust → True for weighted avg, False for simple calc threshold → minimum weight (relative to max) for another value to be averaged

only matters if fullCalcBins == True (or later perhaps if fullCalcClust == True)

westpa.westext.weed.weed_driver.bins_from_yaml_dict(*bin_dict*)

class westpa.westext.weed.weed_driver.WEEDDriver(*sim_manager, plugin_config*)

Bases: object

get_rates(*n_iter, mapper*)

Get rates and associated uncertainties as of n_iter, according to the window size the user has selected (self.windowsize)

prepare_new_iteration()

6.6.2.1.6 Module contents

westext.weed – Support for weighted ensemble equilibrium dynamics

Initial code by Dan Zuckerman (May 2011), integration by Matt Zwier, and testing by Carsen Stringer. Re-factoring and optimization of probability adjustment routines by Joshua L. Adelman (January 2012).

westpa.westext.weed.probAdjustEquil(*binProb, rates, uncert, threshold=0.0, fullCalcClust=False, fullCalcBins=False*)

This function adjusts bin pops in binProb using rates and uncert matrices fullCalcBins → True for weighted avg, False for simple calc fullCalcClust → True for weighted avg, False for simple calc threshold → minimum weight (relative to max) for another value to be averaged

only matters if fullCalcBins == True (or later perhaps if fullCalcClust == True)

class westpa.westext.weed.**WEEDDriver**(*sim_manager, plugin_config*)

Bases: object

get_rates(*n_iter, mapper*)

Get rates and associated uncertainties as of n_iter, according to the window size the user has selected (self.windowsize)

prepare_new_iteration()

6.6.2.2 westpa.westext.wess package

6.6.2.2.1 Submodules

6.6.2.2.2 westpa.westext.wess.ProbAdjust module

westpa.westext.wess.ProbAdjust.**solve_steady_state**(*T, U, target_bins_index*)

westpa.westext.wess.ProbAdjust.**prob_adjust**(*binprob, rates, uncert, oldindex, targets=[]*)

6.6.2.2.3 westpa.westext.wess.wess_driver module

westpa.westext.wess.wess_driver.**check_bool**(*value, action='warn'*)

Check that the given value is boolean in type. If not, either raise a warning (if action == 'warn') or an exception (action == 'raise').

class westpa.westext.wess.wess_driver.**RateAverager**(*bin_mapper, system=None, data_manager=None, work_manager=None*)

Bases: object

Calculate bin-to-bin kinetic properties (fluxes, rates, populations) at 1-tau resolution

extract_data(*iter_indices*)

Extract data from the data_manger and place in dict mirroring the same underlying layout.

task_generator(*iter_start, iter_stop, block_size*)

calculate(*iter_start=None, iter_stop=None, n_blocks=1, queue_size=1*)

Read the HDF5 file and collect flux matrices and population vectors for each bin for each iteration in the range [iter_start, iter_stop). Break the calculation into n_blocks blocks. If the calculation is broken up into more than one block, queue_size specifies the maximum number of tasks in the work queue.

westpa.westext.wess.wess_driver.**prob_adjust**(*binprob, rates, uncert, oldindex, targets=[]*)

westpa.westext.wess.wess_driver.**bins_from_yaml_dict**(*bin_dict*)

westpa.westext.wess.wess_driver.**reduce_array**(*Aij*)

Remove empty rows and columns from an array Aij and return the reduced array Bij and the list of non-empty states

```
class westpa.westext.wess.wess_driver.WESSDriver(sim_manager, plugin_config)
```

Bases: object

```
get_rates(n_iter, mapper)
```

Get rates and associated uncertainties as of n_iter, according to the window size the user has selected (self.windowsize)

```
prepare_new_iteration()
```

6.6.2.2.4 Module contents

```
westpa.westext.wess.prob_adjust(binprob, rates, uncert, oldindex, targets=[])
```

```
class westpa.westext.wess.WESSDriver(sim_manager, plugin_config)
```

Bases: object

```
get_rates(n_iter, mapper)
```

Get rates and associated uncertainties as of n_iter, according to the window size the user has selected (self.windowsize)

```
prepare_new_iteration()
```

6.6.3 Module contents

6.7 westpa.analysis package

This subpackage provides an API to facilitate the analysis of WESTPA simulation data. Its core abstraction is the Run class. A Run instance provides a read-only view of a WEST HDF5 (“west.h5”) file.

API reference: <https://westpa.readthedocs.io/en/latest/documentation/analysis/>

6.7.1 How To

Open a run:

```
>>> from westpa.analysis import Run
>>> run = Run.open('west.h5')
>>> run
<WESTPA Run with 500 iterations at 0x7fcaf8f0d5b0>
```

Iterate over iterations and walkers:

```
>>> for iteration in run:
...     for walker in iteration:
...         pass
... 
```

Access a particular iteration:

```
>>> iteration = run.iteration(10)
>>> iteration
Iteration(10, <WESTPA Run with 500 iterations at 0x7fcaf8f0d5b0>))
```

Access a particular walker:

```
>>> walker = iteration.walker(4)
>>> walker
Walker(4, Iteration(10, <WESTPA Run with 500 iterations at 0x7fc8f0d5b0>))
```

Get the weight and progress coordinate values of a walker:

```
>>> walker.weight
9.876543209876543e-06
>>> walker.pcoords
array([[3.1283207],
       [3.073721 ],
       [2.959221 ],
       [2.6756208],
       [2.7888207]], dtype=float32)
```

Get the parent and children of a walker:

```
>>> walker.parent
Walker(2, Iteration(9, <WESTPA Run with 500 iterations at 0x7fc8f0d5b0>))
>>> for child in walker.children:
...     print(child)
...
Walker(0, Iteration(11, <WESTPA Run with 500 iterations at 0x7fc8f0d5b0>))
Walker(1, Iteration(11, <WESTPA Run with 500 iterations at 0x7fc8f0d5b0>))
Walker(2, Iteration(11, <WESTPA Run with 500 iterations at 0x7fc8f0d5b0>))
Walker(3, Iteration(11, <WESTPA Run with 500 iterations at 0x7fc8f0d5b0>))
Walker(4, Iteration(11, <WESTPA Run with 500 iterations at 0x7fc8f0d5b0>))
```

Trace the ancestry of a walker:

```
>>> trace = walker.trace()
>>> trace
Trace(Walker(4, Iteration(10, <WESTPA Run with 500 iterations at 0x7fc8f0d5b0>)))
>>> for walker in trace:
...     print(walker)
...
Walker(1, Iteration(1, <WESTPA Run with 500 iterations at 0x7fc8f0d5b0>))
Walker(4, Iteration(2, <WESTPA Run with 500 iterations at 0x7fc8f0d5b0>))
Walker(5, Iteration(3, <WESTPA Run with 500 iterations at 0x7fc8f0d5b0>))
Walker(6, Iteration(4, <WESTPA Run with 500 iterations at 0x7fc8f0d5b0>))
Walker(9, Iteration(5, <WESTPA Run with 500 iterations at 0x7fc8f0d5b0>))
Walker(8, Iteration(6, <WESTPA Run with 500 iterations at 0x7fc8f0d5b0>))
Walker(8, Iteration(7, <WESTPA Run with 500 iterations at 0x7fc8f0d5b0>))
Walker(13, Iteration(8, <WESTPA Run with 500 iterations at 0x7fc8f0d5b0>))
Walker(2, Iteration(9, <WESTPA Run with 500 iterations at 0x7fc8f0d5b0>))
Walker(4, Iteration(10, <WESTPA Run with 500 iterations at 0x7fc8f0d5b0>))
```

Close a run (and its underlying HDF5 file):

```
>>> run.close()
>>> run
<Closed WESTPA Run at 0x7fc8f0d5b0>
```

(continues on next page)

(continued from previous page)

```
>>> run.h5file
<Closed HDF5 file>
```

6.7.2 Retrieving Trajectories

6.7.2.1 Built-in Reader

MD trajectory data stored in an identical manner as in the [Basic NaCl tutorial](#) may be retrieved using the built-in BasicMDTrajectory reader with its default settings:

```
>>> from westpa.analysis import BasicMDTrajectory
>>> trajectory = BasicMDTrajectory()
```

Here `trajectory` is a callable object that takes either a `Walker` or a `Trace` instance as input and returns an `MDTraj Trajectory`:

```
>>> traj = trajectory(walker)
>>> traj
<mdtraj.Trajectory with 5 frames, 33001 atoms, 6625 residues, and unitcells at 0x7fcae484ad00>
>>> traj = trajectory(trace)
>>> traj
<mdtraj.Trajectory with 41 frames, 33001 atoms, 6625 residues, and unitcells at 0x7fcae487c790>
```

Minor variations of the “basic” trajectory storage protocol (e.g., use of different file formats) can be handled by changing the parameters of the `BasicMDTrajectory` reader. For example, suppose that instead of storing the coordinate and topology data for trajectory segments in separate files (“seg.dcd” and “bstate.pdb”), we store them together in a `MDTraj HDF5` trajectory file (“seg.h5”). This change can be accommodated by explicitly setting the `traj_ext` and `top` parameters of the trajectory reader:

```
>>> trajectory = BasicMDTrajectory(traj_ext='.h5', top=None)
```

Trajectories that are saved with the HDF5 Framework can use `HDF5MDTrajectory` reader instead.

6.7.2.2 Custom Readers

For users requiring greater flexibility, custom trajectory readers can be implemented using the `westpa.analysis.Trajectory` class. Implementing a custom reader requires two ingredients:

1. A function for retrieving individual trajectory segments. The function must take a `Walker` instance as its first argument and return a sequence (e.g., a list, NumPy array, or `MDTraj Trajectory`) representing the trajectory of the walker. Moreover, it must accept a Boolean keyword argument `include_initpoint`, which specifies whether the returned trajectory includes its initial point.
2. A function for concatenating trajectory segments. A default implementation is provided by the `concatenate()` function in the `westpa.analysis.trajectories` module.

6.7.3 westpa.analysis.core module

class westpa.analysis.core.Run(*h5filename*='west.h5')

A read-only view of a WESTPA simulation run.

Parameters

h5filename (*str* or *file-like object*, default 'west.h5') – Pathname or stream of a main WESTPA HDF5 data file.

classmethod open(*h5filename*='west.h5')

Alternate constructor.

Parameters

h5filename (*str* or *file-like object*, default 'west.h5') – Pathname or stream of a main WESTPA HDF5 data file.

close()

Close the Run instance by closing the underlying WESTPA HDF5 file.

property closed

Whether the Run instance is closed.

Type

bool

property summary

Summary data by iteration.

Type

pd.DataFrame

property num_iterations

Number of completed iterations.

Type

int

property iterations

Sequence of iterations.

Type

Sequence[[Iteration](#)]

property num_walkers

Total number of walkers.

Type

int

property num_segments

Total number of trajectory segments (alias self.num_walkers).

Type

int

property walkers

All walkers in the run.

Type

Iterable[[Walker](#)]

property recycled_walkers

Walkers that stopped in the sink.

Type

Iterable[[Walker](#)]

property initial_walkers

Walkers whose parents are initial states.

Type

Iterable[[Walker](#)]

iteration(number)

Return a specific iteration.

Parameters

number (*int*) – Iteration number (1-based).

Returns

The iteration indexed by *number*.

Return type

[Iteration](#)

class westpa.analysis.core.Iteration(number, run)

An iteration of a WESTPA simulation.

Parameters

- **number** (*int*) – Iteration number (1-based).
- **run** ([Run](#)) – Simulation run to which the iteration belongs.

property h5group

HDF5 group containing the iteration data.

Type

h5py.Group

property prev

Previous iteration.

Type

[Iteration](#)

property next

Next iteration.

Type

[Iteration](#)

property summary

Iteration summary.

Type

pd.DataFrame

property segment_summaries

Segment summary data for the iteration.

Type

pd.DataFrame

property pcoords

Progress coordinate snapshots of each walker.

Type

3D ndarray

property weights

Statistical weight of each walker.

Type

1D ndarray

property bin_target_counts

Target count for each bin.

Type

1D ndarray, dtype=uint64

property bin_mapper

Bin mapper used in the iteration.

Type

BinMapper

property num_bins

Number of bins.

Type

int

property bins

Bins.

Type

Iterable[*Bin*]

property num_walkers

Number of walkers in the iteration.

Type

int

property num_segments

Number of trajectory segments (alias self.num_walkers).

Type

int

property walkers

Walkers in the iteration.

Type

Iterable[*Walker*]

property recycled_walkers

Walkers that stopped in the sink.

Type

Iterable[*Walker*]

property initial_walkers

Walkers whose parents are initial states.

Type

Iterable[*Walker*]

property auxiliary_data

Auxiliary data stored for the iteration.

Type

h5py.Group or None

property basis_state_summaries

Basis state summary data.

Type

pd.DataFrame

property basis_state_pcoords

Progress coordinates of each basis state.

Type

2D ndarray

property basis_states

Basis states in use for the iteration.

Type

list[*BasisState*]

property has_target_states

Whether target (sink) states are defined for this iteration.

Type

bool

property target_state_summaries

Target state summary data.

Type

pd.DataFrame or None

property target_state_pcoords

Progress coordinates of each target state.

Type

2D ndarray or None

property target_states

Target states in use for the iteration.

Type

list[*TargetState*]

property sink

Union of bins serving as the recycling sink.

Type

BinUnion or None

bin(index)

Return the bin with the given index.

Parameters

index (*int*) – Bin index (0-based).

Returns

The bin indexed by *index*.

Return type

Bin

walker(index)

Return the walker with the given index.

Parameters

index (*int*) – Walker index (0-based).

Returns

The walker indexed by *index*.

Return type

Walker

basis_state(*index*)

Return the basis state with the given index.

Parameters

index (*int*) – Basis state index (0-based).

Returns

The basis state indexed by *index*.

Return type

BasisState

target_state(*index*)

Return the target state with the given index.

Parameters

index (*int*) – Target state index (0-based).

Returns

The target state indexed by *index*.

Return type

TargetState

class westpa.analysis.core.**Walker**(*index*, *iteration*)

A walker in an iteration of a WESTPA simulation.

Parameters

- **index** (*int*) – Walker index (0-based).
- **iteration** (*Iteration*) – Iteration to which the walker belongs.

property **run**

Run to which the walker belongs.

Type

Run

property **weight**

Statistical weight of the walker.

Type

float64

property **pcoords**

Progress coordinate snapshots.

Type

2D ndarray

property **num_snapshots**

Number of snapshots.

Type

int

property **segment_summary**

Segment summary data.

Type

pd.Series

property **parent**

The parent of the walker.

Type

Walker or *InitialState*

property children

The children of the walker.

Type

Iterable[[Walker](#)]

property recycled

True if the walker stopped in the sink, False otherwise.

Type

bool

property initial

True if the parent of the walker is an initial state, False otherwise.

Type

bool

property auxiliary_data

Auxiliary data for the walker.

Type

dict

trace(kwargs)**

Return the trace (ancestral line) of the walker.

For full documentation see [Trace](#).

Returns

The trace of the walker.

Return type

[Trace](#)

class westpa.analysis.core.BinUnion(indices, mapper)

A (disjoint) union of bins defined by a common bin mapper.

Parameters

- **indices** (*iterable of int*) – The indices of the bins comprising the union.
- **mapper** ([BinMapper](#)) – The bin mapper defining the bins.

union(*others)

Return the union of the bin union and all others.

Parameters

***others** ([BinUnion](#)) – Other [BinUnion](#) instances, consisting of bins defined by the same underlying bin mapper.

Returns

The union of *self* and *others*.

Return type

[BinUnion](#)

intersection(*others)

Return the intersection of the bin union and all others.

Parameters

***others** ([BinUnion](#)) – Other [BinUnion](#) instances, consisting of bins defined by the same underlying bin mapper.

Returns

The intersection of *self* and *others*.

Return type*BinUnion***class** westpa.analysis.core.**Bin**(*index, mapper*)

A bin defined by a bin mapper.

Parameters

- **index** (*int*) – The index of the bin.
- **mapper** (*BinMapper*) – The bin mapper defining the bin.

class westpa.analysis.core.**Trace**(*walker, source=None, max_length=None*)

A trace of a walker's ancestry.

Parameters

- **walker** (*Walker*) – The terminal walker.
- **source** (*Bin, BinUnion, or collections.abc.Container, optional*) – A source (macro)state, specified as a container object whose `__contains__()` method is the indicator function for the corresponding subset of progress coordinate space. The trace is stopped upon encountering a walker that stopped in *source*.
- **max_length** (*int, optional*) – The maximum number of walkers in the trace.

6.7.4 westpa.analysis.trajectories module

class westpa.analysis.trajectories.**Trajectory**(*fget=None, *, fconcat=None*)

A callable that returns the trajectory of a walker or trace.

Parameters

- **fget** (*callable*) – Function for retrieving a single trajectory segment. Must take a *Walker* instance as its first argument and accept a boolean keyword argument *include_initpoint*. The function should return a sequence (e.g., a list or ndarray) representing the trajectory of the walker. If *include_initpoint* is *True*, the trajectory segment should include its initial point. Otherwise, the trajectory segment should exclude its initial point.
- **fconcat** (*callable, optional*) – Function for concatenating trajectory segments. Must take a sequence of trajectory segments as input and return their concatenation. The default concatenation function is *concatenate()*.

property segment_collector

Segment retrieval manager.

Type*SegmentCollector***property fget**

Function for getting trajectory segments.

Type

callable

property fconcat

Function for concatenating trajectory segments.

Type

callable

class westpa.analysis.trajectories.**SegmentCollector**(*trajectory, use_threads=False, max_workers=None, show_progress=False*)

An object that manages the retrieval of trajectory segments.

Parameters

- **trajectory** ([Trajectory](#)) – The trajectory to which the segment collector is attached.
- **use_threads** (*bool*, *default False*) – Whether to use a pool of threads to retrieve trajectory segments asynchronously. Setting this parameter to True may be useful when segment retrieval is an I/O bound task.
- **max_workers** (*int*, *optional*) – Maximum number of threads to use. The default value is specified in the [ThreadPoolExecutor](#) documentation.
- **show_progress** (*bool*, *default False*) – Whether to show a progress bar when retrieving multiple segments.

get_segments(*walkers*, *initpoint_mask=None*, ***kwargs*)

Retrieve the trajectories of multiple walkers.

Parameters

- **walkers** (*sequence of Walker*) – The walkers for which to retrieve trajectories.
- **initpoint_mask** (*sequence of bool*, *optional*) – A Boolean mask indicating whether each trajectory segment should include (True) or exclude (False) its initial point. Default is all True.

Returns

The trajectory of each walker.

Return type

list of sequences

```
class westpa.analysis.trajectories.BasicMDTrajectory(top='bstate.pdb', traj_ext='.dcd',
                                                    state_ext='.xml', sim_root='.')
```

Trajectory reader for MD trajectories stored as in the [Basic Tutorial](#).

Parameters

- **top** (*str or mdtraj.Topology*, *default 'bstate.pdb'*) –
- **traj_ext** (*str*, *default '.dcd'*) –
- **state_ext** (*str*, *default '.xml'*) –
- **sim_root** (*str*, *default '.'*) –

```
class westpa.analysis.trajectories.HDF5MDTrajectory
```

Trajectory reader for MD trajectories stored by the HDF5 framework.

```
westpa.analysis.trajectories.concatenate(segments)
```

Return the concatenation of a sequence of trajectory segments.

Parameters

- **segments** (*sequence of sequences*) – A sequence of trajectory segments.

Returns

The concatenation of *segments*.

Return type

sequence

6.7.5 westpa.analysis.statistics module

`westpa.analysis.statistics.time_average(observable, iterations)`

Compute the time average of an observable.

Parameters

- **observable** (*Callable*[[[Walker](#)], *ArrayLike*]) – Function that takes a walker as input and returns a number or a fixed-size array of numbers.
- **iterations** (*Sequence*[[Iteration](#)]) – Sequence of iterations over which to compute the average.

Returns

The time average of *observable* over *iterations*.

Return type

ArrayLike

6.8 HDF5 File Schema

WESTPA stores all of its simulation data in the cross-platform, self-describing [HDF5](#) file format. This file format can be read and written by a variety of languages and toolkits, including C/C++, Fortran, Python, Java, and [Matlab](#) so that analysis of weighted ensemble simulations is not tied to using the WESTPA framework. HDF5 files are organized like a filesystem, where arbitrarily-nested groups (i.e. directories) are used to organize datasets (i.e. files). The excellent [HDFView](#) program may be used to explore WEST data files.

The canonical file format reference for a given version of the WEST code is described in [src/west/data_manager.py](#).

6.8.1 Overall structure

```
/
#ibstates/
  index
  naming
    bstate_index
    bstate_pcoord
    istrate_index
    istrate_pcoord
#tstates/
  index
bin_topologies/
  index
  pickles
iterations/
  iter_XXXXXXXX/iter_XXXXXXXX/
    auxdata/
    bin_target_counts
    ibstates/
      bstate_index
      bstate_pcoord
      istrate_index
      istrate_pcoord
    pcoord
```

(continues on next page)

(continued from previous page)

```

    seg_index
    wtgraph
    ...
summary

```

6.8.2 The root group (/)

The root of the WEST HDF5 file contains the following entries (where a trailing “/” denotes a group):

Name	Type	Description
ibstates/	Group	Initial and basis states for this simulation
tstates/	Group	Target (recycling) states for this simulation; may be empty
bin_topologies/	Group	Data pertaining to the binning scheme used in each iteration
iterations/	Group	Iteration data
summary	Dataset (1-dimensional, compound)	Summary data by iteration

6.8.2.1 The iteration summary table (/summary)

Field	Description
n_particles	the total number of walkers in this iteration
norm	total probability, for stability monitoring
min_bin_prob	smallest probability contained in a bin
max_bin_prob	largest probability contained in a bin
min_seg_prob	smallest probability carried by a walker
max_seg_prob	largest probability carried by a walker
cputime	total CPU time (in seconds) spent on propagation for this iteration
walltime	total wallclock time (in seconds) spent on this iteration
binhash	a hex string identifying the binning used in this iteration

6.8.3 Per iteration data (/iterations/iter_XXXXXXX)

Data for each iteration is stored in its own group, named according to the iteration number and zero-padded out to 8 digits, as in `/iterations/iter_00000001` for iteration 1. This is done solely for convenience in dealing with the data in external utilities that sort output by group name lexicographically. The field width is in fact configurable via the `iter_prec` configuration entry under data section of the WESTPA configuration file.

The HDF5 group for each iteration contains the following elements:

Name	Type	Description
auxdata/	Group	All user-defined auxiliary data0 sets
bin_target_co	Dataset (1-dimensional)	The per-bin target count for the iteration
ibstates/	Group	Initial and basis state data for the iteration
pcoord	Dataset (3-dimensional)	Progress coordinate data for the iteration stored as a (num of segments, pcoord_len, pcoord_ndim) array
seg_index	Dataset (1-dimensional, compound)	Summary data for each segment
wtgraph	Dataset (1-dimensional)	

6.8.3.1 The segment summary table (/iterations/iter_XXXXXXX/seg_index)

Field	Description
weight	Segment weight
parent_id	Index of parent
wtg_n_parents	
wtg_offset	
cputime	Total cpu time required to run the segment
walltime	Total walltime required to run the segment
endpoint_type	
status	

6.8.4 Bin Topologies group (/bin_topologies)

Bin topologies used during a WE simulation are stored as a unique hash identifier and a serialized `BinMapper` object in `python pickle` format. This group contains two datasets:

- **index:** Compound array containing the bin hash and pickle length
- **pickle:** The pickled `BinMapper` objects for each unique mapper stored in a (num unique mappers, max pickled size) array

6.9 Overview

6.10 Style Guide

6.10.1 Preface

The WESTPA documentation should help the user to understand how WESTPA works and how to use it. To aid in effective communication, a number of guidelines appear below.

When writing in the WESTPA documentation, please be:

- Correct
- Clear
- Consistent

- Concise

Articles in this documentation should follow the guidelines on this page. However, there may be cases when following these guidelines will make an article confusing: when in doubt, use your best judgment and ask for the opinions of those around you.

6.10.2 Style and Usage

6.10.2.1 Acronyms and abbreviations

- Software documentation often involves extensive use of acronyms and abbreviations.

Acronym: A word formed from the initial letter or letters of each or most of the parts of a compound term

Abbreviation: A shortened form of a written word or name that is used in place of the full word or name

- Define non-standard acronyms and abbreviations on their first use by using the full-length term, followed by the acronym or abbreviation in parentheses.

A potential of mean force (PMF) diagram may aid the user in visualizing the energy landscape of the simulation.

- Only use acronyms and abbreviations when they make an idea more clear than spelling out the full term. Consider clarity from the point of view of a new user who is intelligent but may have little experience with computers.

Correct: The WESTPA wiki supports HyperText Markup Language (HTML). For example, the user may use HTML tags to give text special formatting. However, be sure to test that the HTML tag gives the desired effect by previewing edits before saving.

Avoid: The WESTPA wiki supports HyperText Markup Language. For example, the user may use HyperText Markup Language tags to give text special formatting. However, be sure to test that the HyperText Markup Language tag gives the desired effect by previewing edits before saving.

Avoid: For each iter, make sure to return the pcoord and any auxdata.

- Use all capital letters for abbreviating file types. File extensions should be lowercase.

HDF5, PNG, MP4, GRO, XTC

west.h5, bound.png, unfolding.mp4, protein.gro, segment.xtc

- Provide pronunciations for acronyms that may be difficult to sound out.
- Do not use periods in acronyms and abbreviations except where it is customary:

Correct: HTML, U.S.

Avoid: H.T.M.L., US

6.10.2.2 Capitalization

- Capitalize at the beginning of each sentence.
- Do not capitalize after a semicolon.
- Do not capitalize after a colon, unless multiple sentences follow the colon.
- In this case, capitalize each sentence.
- Preserve the capitalization of computer language elements (commands, utilities, variables, modules, classes, and arguments).
- Capitalize generic Python variables according to the

- [PEP 0008 Python Style Guide](#). For example, generic class names should follow the *CapWords* convention, such as `GenericClass`.

6.10.2.3 Contractions

- Do not use contractions. Contractions are a shortened version of word characterized by the omission of internal letters.
Avoid: can't, don't, shouldn't
- Possessive nouns are not contractions. Use possessive nouns freely.

6.10.2.4 Internationalization

- Use short sentences (less than 25 words). Although we do not maintain WESTPA documentation in languages other than English, some users may use automatic translation programs. These programs function best with short sentences.
- Do not use technical terms where a common term would be equally or more clear.
- Use multiple simple sentences in place of a single complicated sentence.

6.10.2.5 Italics

- Use italics (surround the word with * * on each side) to highlight words that are not part of a sentence's normal grammar.
Correct: The word *istates* refers to the initial states that WESTPA uses to begin trajectories.

6.10.2.6 Non-English words

- Avoid Latin words and abbreviations.
Avoid: etc., et cetera, e.g., i.e.

6.10.2.7 Specially formatted characters

- Never begin a sentence with a specially formatted character. This includes abbreviations, variable names, and anything else this guide instructs to use with special tags. Sentences may begin with *WESTPA*.
Correct: The program `ls` allows the user to see the contents of a directory.
Avoid: `ls` allows the user to see the contents of a directory.
- Use the word *and* rather than an & ampersand .
- When a special character has a unique meaning to a program, first use the character surrounded by `` tags and then spell it out.
Correct: Append an & ampersand to a command to let it run in the background.
Avoid: Append an "&" to a command... Append an & to a command... Append an ampersand to a command...
- There are many names for the # hash mark, including hash tag, number sign, pound sign, and octothorpe. Refer to this symbol as a "hash mark".

6.10.2.8 Subject

- Refer to the end WESTPA user as *the user* in software documentation.

Correct: The user should use the `processes` work manager to run segments in parallel on a single node.

- Refer to the end WESTPA user as *you* in tutorials (you is the implied subject of commands). It is also acceptable to use personal pronouns such as *we* and *our*. Be consistent within the tutorial.

Correct: You should have two files in this directory, named `system.py` and `west.cfg`.

6.10.2.9 Tense

- Use *should* to specify proper usage.

Correct: The user should run `w_truncate -n <var>iter</var>` to remove iterations after and including `iter` from the HDF5 file specified in the WESTPA configuration file.

- Use *will* to specify expected results and output.

Correct: WESTPA will create a HDF5 file when the user runs `w_init`.

6.10.2.10 Voice

- Use active voice. Passive voice can obscure a sentence and add unnecessary words.

Correct: WESTPA will return an error if the sum of the weights of segments does not equal one.

Avoid: An error will be returned if the sum of the weights of segments does not equal one.

6.10.2.11 Weighted ensemble

- Refer to weighted ensemble in all lowercase, unless at the beginning of a sentence. Do not hyphenate.

Correct: WESTPA is an implementation of the weighted ensemble algorithm.

Avoid: WESTPA is an implementation of the weighted-ensemble algorithm.

Avoid: WESTPA is an implementation of the Weighted Ensemble algorithm.

6.10.2.12 WESTPA

- Refer to WESTPA in all capitals. Do not use bold, italics, or other special formatting except when another guideline from this style guide applies.

Correct: Install the WESTPA software package.

- The word *WESTPA* may refer to the software package or a entity of running software.

Correct: WESTPA includes a number of analysis utilities.

Correct: WESTPA will return an error if the user does not supply a configuration file.

6.10.3 Computer Language Elements

6.10.3.1 Classes, modules, and libraries

- Display class names in fixed-width font using the `` `` tag.

Correct: `WESTPropagator`

Correct: The `numpy` library provides access to various low-level mathematical and scientific calculation routines.

- Generic class names should be relevant to the properties of the class; do not use *foo* or *bar*

```
class UserDefinedBinMapper(RectilinearBinMapper)
```

6.10.3.2 Methods and commands

- Refer to a method by its name without parentheses, and without prepending the name of its class. Display methods in fixed-width font using the `` `` tag.

Correct: the `arange` method of the `numpy` library

Avoid: the `arange()` method of the `numpy` library

Avoid: the `numpy.arange` method

- When referring to the arguments that a method expects, mention the method without arguments first, and then use the method's name followed by parenthesis and arguments.

Correct: WESTPA calls the `assign` method as `assign(coords, mask=None, output=None)`

- Never use a method or command as a verb.

Correct: Run `cd` to change the current working directory.

Avoid: `cd` into the main simulation directory.

6.10.3.3 Programming languages

- Some programming languages are both a language and a command. When referring to the language, capitalize the word and use standard font. When referring to the command, preserve capitalization as it would appear in a terminal and use the `` `` tag.

Using WESTPA requires some knowledge of Python.

Run `python` to launch an interactive session.

The Bash shell provides some handy capabilities, such as wildcard matching.

Use `bash` to run `example.sh`.

6.10.3.4 Scripts

- Use the `.. code-block::` directive for short scripts. Options are available for some languages, such as `.. code-block:: bash` and `.. code-block:: python`.

```
#!/bin/bash
# This is a generic Bash script.

BASHVAR="Hello, world!"
echo $BASHVAR
```

```
#!/usr/bin/env python
# This is a generic Python script.

def main():
    pythonstr = "Hello, world!"
    print(pythonstr)
    return
if __name__ == "__main__":
    main()
```

- Begin a code snippet with a `#!` *shebang* (yes, this is the real term), followed by the usual path to a program. The line after the shebang should be an ellipsis, followed by lines of code. Use `#!/bin/bash` for Bash scripts, `#!/bin/sh` for generic shell scripts, and `#!/usr/bin/env python` for Python scripts. For Python code snippets that are not a stand-alone script, place any import commands between the shebang line and ellipsis.

```
#!/usr/bin/env python
import numpy
...
def some_function(generic_vals):
    return 1 + numpy.mean(generic_vals)
```

- Follow the [PEP 0008 Python Style Guide](#) for Python scripts.
 - Indents are four spaces.
 - For comments, use the `#` hash mark followed by a single space, and then the comment's text.
 - Break lines after 80 characters.
- For Bash scripts, consider following [Google's Shell Style Guide](#)
- Indents are two spaces.
- Use blank lines to improve readability
- Use `;` `do` and `;` `then` on the same line as `while`, `for`, and `if`.
- Break lines after 80 characters.
- For other languages, consider following a logical style guide. At minimum, be consistent.

6.10.3.5 Variables

- Use the fixed-width `` `` tag when referring to a variable.
the `ndim` attribute
- When explicitly referring to an attribute as well as its class, refer to an attribute as: the `attr` attribute of `GenericClass`, rather than `GenericClass.attr`
- Use the `$` dollar sign before Bash variables.
WESTPA makes the variable `$WEST_BSTATE_DATA_REF` available to new trajectories.

6.11 Source Code Management

6.12 Documentation Practices

6.12.1 Introduction to Editing the Sphinx Documentation

Documentation for WESTPA is maintained using [Sphinx](#). Docstrings are formatted in the [Numpy style](#), which are converted to ReStructuredText using Sphinx' [Napoleon](#) plugin, a feature included with Sphinx.

Make sure `sphinx` and `sphinx_rtd_theme` are installed on the system. The settings for the documentation are specified in `/westpa/doc/conf.py`. In order to successfully build the documentation, your system has to satisfy the minimum environment to install WESTPA.

The documentation may be built locally in the `_build` folder by navigating to the `doc` folder, and running:

```
make html
```

to prepare an html version or:

```
make latexpdf
```

To prepare a pdf. The latter requires `latex` to be available.

6.12.2 Uploading to ReadTheDocs

The online copy of WESTPA Sphinx documentation is hosted on [ReadtheDocs](#). The Sphinx documentations on the main branch are updated whenever the main branch is updated, via a webhook setup on ReadtheDocs and `/westpa/.readthedocs.yml`. The environment used to build the documentation on the RTD servers are described in `/westpa/doc/doc_env.yml`.

6.12.3 In Cases of Major Revisions in Code Base

Currently, each `.rst` file contains pre-written descriptions and autogenerated sections generated from docstrings via `automodule`. In cases where the WESTPA code base has significantly changed, the structure of the code base can be regenerated into the `test` folder by running the following command in the `doc` folder:

```
sphinx-apidoc -f -o test ../src/westpa
```


6.13 WESTPA Modules API

6.13.1 Binning

Bin assignment for WEST simulations. This module defines “bin mappers” which take vectors of coordinates (or rather, coordinate tuples), and assign each a definite integer value identifying a bin. Critical portions are implemented in a Cython extension module.

A number of pre-defined bin mappers are available here:

- *RectilinearBinMapper*, for bins divided by N-dimensional grids
- *FuncBinMapper*, for functions which directly calculate bin assignments for a number of coordinate values. This is best used with C/Cython/Numba functions, or intelligently-tuned numpy-based Python functions.
- *VectorizingFuncBinMapper*, for functions which calculate a bin assignment for a single coordinate value. This is best used for arbitrary Python functions.
- *PiecewiseBinMapper*, for using a set of boolean-valued functions, one per bin, to determine assignments. This is likely to be much slower than a *FuncBinMapper* or *VectorizingFuncBinMapper* equipped with an appropriate function, and its use is discouraged.

One “super-mapper” is available, for assembling more complex bin spaces from simpler components:

- *RecursiveBinMapper*, for nesting one set of bins within another.

Users are also free to implement their own mappers. A bin mapper must implement, at least, an `assign(coords, mask=None, output=None)` method, which is responsible for mapping each of the vector of coordinate tuples `coords` to an integer (`np.uint16`) indicating a what bin that coordinate tuple falls into. The optional `mask` (a numpy bool array) specifies that some coordinates are to be skipped; this is used, for instance, by the recursive (nested) bin mapper to minimize the number of calculations required to definitively assign a coordinate tuple to a bin. Similarly, the optional `output` must be an integer (`uint16`) array of the same length as `coords`, into which assignments are written. The `assign()` function must return a reference to `output`. (This is used to avoid allocating many temporary output arrays in complex binning scenarios.)

A user-defined bin mapper must also make an `nbins` property available, containing the total number of bins within the mapper.

6.13.2 YAMLCFG

YAML-based configuration files for WESTPA

6.13.3 RC

class `westpa.core._rc.WESTRC`

A class, an instance of which is accessible as `westpa.rc`, to handle global issues for WEST-PA code, such as loading modules and plugins, writing output based on verbosity level, adding default command line options, and so on.

6.14 WESTPA Tools

6.15 WEST

6.15.1 Setup

6.15.1.1 Defining and Calculating Progress Coordinates

6.15.1.2 Binning

The Weighted Ensemble method enhances sampling by partitioning the space defined by the progress coordinates into non-overlapping bins. WESTPA provides a number of pre-defined types of bins that the user must parameterize within the `system.py` file, which are detailed below.

Users are also free to implement their own mappers. A bin mapper must implement, at least, an `assign(coords, mask=None, output=None)` method, which is responsible for mapping each of the vector of coordinate tuples `coords` to an integer (`numpy.uint16`) indicating what bin that coordinate tuple falls into. The optional `mask` (a `numpy bool` array) specifies that some coordinates are to be skipped; this is used, for instance, by the recursive (nested) bin mapper to minimize the number of calculations required to definitively assign a coordinate tuple to a bin. Similarly, the optional `output` must be an integer (`uint16`) array of the same length as `coords`, into which assignments are written. The `assign()` function must return a reference to `output`. (This is used to avoid allocating many temporary output arrays in complex binning scenarios.)

A user-defined bin mapper must also make an `nbins` property available, containing the total number of bins within the mapper.

6.15.1.2.1 RectilinearBinMapper

Creates an N-dimensional grid of bins. The Rectilinear bin mapper is initialized by defining a set of bin boundaries:

```
self.bin_mapper = RectilinearBinMapper(boundaries)
```

where `boundaries` is a list or other iterable containing the bin boundaries along each dimension. The bin boundaries must be monotonically increasing along each dimension. It is important to note that a one-dimensional bin space must still be represented as a list of lists as in the following example::

```
bounds = [-float('inf'), 0.0, 1.0, 2.0, 3.0, float('inf')]
self.bin_mapper = RectilinearBinMapper([bounds])
```

A two-dimensional system might look like::

```
boundaries = [(-1,-0.5,0,0.5,1), (-1,-0.5,0,0.5,1)]
self.bin_mapper = RectilinearBinMapper(boundaries)
```

where the first tuple in the list defines the boundaries along the first progress coordinate, and the second tuple defines the boundaries along the second. Of course a list of arbitrary dimensions can be defined to create an N-dimensional grid discretizing the progress coordinate space.

6.15.1.2.2 VoronoiBinMapper

A one-dimensional mapper which assigns a multidimensional progress coordinate to the closest center based on a distance metric. The Voronoi bin mapper is initialized with the following signature within the `WESTSystem`. `initialize::`

```
self.bin_mapper = VoronoiBinMapper(dfunc, centers, dfargs=None, dfkwargs=None)
```

- `centers` is a `(n_centers, pcoord_ndim)` shaped numpy array defining the generators of the Voronoi cells
- `dfunc` is a method written in Python that returns an `(n_centers,)` shaped array containing the distance between a single set of progress coordinates for a segment and all of the centers defining the Voronoi tessellation. It takes the general form::

```
def dfunc(p, centers, *dfargs, **dfkwargs):
    ...
    return d
```

where `p` is the progress coordinates of a single segment at one time slice of shape `(pcoord_ndim,)`, `centers` is the full set of centers, `dfargs` is a tuple or list of positional arguments and `dfkwargs` is a dictionary of keyword arguments. The bin mapper's `assign` method then assigns the progress coordinates to the closest bin (minimum distance). It is the responsibility of the user to ensure that the distance is calculated using the appropriate metric.

- `dfargs` is an optional list or tuple of positional arguments to pass into `dfunc`.
- `dfkwargs` is an optional dict of keyword arguments to pass into `dfunc`.

6.15.1.2.3 FuncBinMapper

A bin mapper that employs a set of user-defined function, which directly calculate bin assignments for a number of coordinate values. The function is responsible for iterating over the entire coordinate set. This is best used with C/Cython/Numba methods, or intellegently-tuned numpy-based Python functions.

The `FuncBinMapper` is initialized as::

```
self.bin_mapper = FuncBinMapper(func, nbins, args=None, kwargs=None)
```

where `func` is the user-defined method to assign coordinates to bins, `nbins` is the number of bins in the partitioning space, and `args` and `kwargs` are optional positional and keyword arguments, respectively, that are passed into `func` when it is called.

The user-defined function should have the following form::

```
def func(coords, mask, output, *args, **kwargs)
    ....
```

where the assignments returned in the `output` array, which is modified in-place.

As a contrived example, the following function would assign all segments to bin 0 if the sum of the first two progress coordinates was less than `s*0.5`, and to bin 1 otherwise, where `s=1.5`::

```
def func(coords, mask, output, s):
    output[coords[:,0] + coords[:,1] < s*0.5] = 0
    output[coords[:,0] + coords[:,1] >= s*0.5] = 1
    ....
```

(continues on next page)

(continued from previous page)

```
self.bin_mapper = FuncBinMapper(func, 2, args=(1.5,))
```

6.15.1.2.4 VectorizingFuncBinMapper

Like the `FuncBinMapper`, the `VectorizingFuncBinMapper` uses a user-defined method to calculate bin assignments. They differ, however, in that while the user-defined method passed to an instance of the `FuncBinMapper` is responsible for iterating over all coordinate sets passed to it, the function associated with the `VectorizingFuncBinMapper` is evaluated once for each unmasked coordinate tuple provided. It is not responsible explicitly for iterating over multiple progress coordinate sets.

The `VectorizingFuncBinMapper` is initialized as::

```
self.bin_mapper = VectorizingFuncBinMapper(func, nbins, args=None, kwargs=None)
```

where `func` is the user-defined method to assign coordinates to bins, `nbins` is the number of bins in the partitioning space, and `args` and `kwargs` are optional positional and keyword arguments, respectively, that are passed into `func` when it is called.

The user-defined function should have the following form::

```
def func(coords, *args, **kwargs)
    ....
```

Mirroring the simple example shown for the `FuncBinMapper`, the following should result in the same result for a given set of coordinates. Here segments would be assigned to bin 0 if the sum of the first two progress coordinates was less than $s \cdot 0.5$, and to bin 1 otherwise, where $s=1.5$::

```
def func(coords, s):
    if coords[0] + coords[1] < s*0.5:
        return 0
    else:
        return 1
    ....

self.bin_mapper = VectorizingFuncBinMapper(func, 2, args=(1.5,))
```

6.15.1.2.5 PiecewiseBinMapper

6.15.1.2.6 RecursiveBinMapper

The `RecursiveBinMapper` is used for assembling more complex bin spaces from simpler components and nesting one set of bins within another. It is initialized as::

```
self.bin_mapper = RecursiveBinMapper(base_mapper, start_index=0)
```

The `base_mapper` is an instance of one of the other bin mappers, and `start_index` is an (optional) offset for indexing the bins. Starting with the `base_mapper`, additional bins can be nested into it using the `add_mapper(mapper, replaces_bin_at)`. This method will replace the bin containing the coordinate tuple `replaces_bin_at` with the mapper specified by `mapper`.

As a simple example consider a bin space in which the `base_mapper` assigns a segment with progress coordinate with values <1 into one bin and ≥ 1 into another. Within the former bin, we will nest a second mapper which partitions progress coordinate space into one bin for progress coordinate values <0.5 and another for progress coordinates with values ≥ 0.5 . The bin space would look like the following with corresponding code::

```
"""
    0                                1                                2
    +-----+-----+-----+
    |           0.5           |           |           | | | |
    | +-----+-----+-----+ |           |           |
    | |           |           | |           |           |
    | |           |           | |           |           |
    | |           |           | |           |           |
    | |           |           | |           |           |
    | |           |           | |           |           |
    | +-----+-----+-----+ |           |           |
    +-----+-----+-----+
"""

def fn1(coords, mask, output):
    test = coords[:,0] < 1
    output[mask & test] = 0
    output[mask & ~test] = 1

def fn2(coords, mask, output):
    test = coords[:,0] < 0.5
    output[mask & test] = 0
    output[mask & ~test] = 1

outer_mapper = FuncBinMapper(fn1,2)
inner_mapper = FuncBinMapper(fn2,2)
rmapper = RecursiveBinMapper(outer_mapper)
rmapper.add_mapper(inner_mapper, [0.5])

/prettyprint
```

Examples of more complicated nesting schemes can be found in the [tests](#) for the WESTPA binning apparatus.

6.15.1.3 Initial/Basis States

A WESTPA simulation is initialized using `w_init` with an initial distribution of replicas generated from a set of basis states. These basis states are used to generate initial states for new trajectories, either at the beginning of the simulation or due to recycling. Basis states are specified when running `w_init` either in a file specified with `--bstates-from`, or by one or more `--bstate` arguments. If neither `--bstates-from` nor at least one `--bstate` argument is provided, then a default basis state of probability one identified by the state ID zero and label “basis” will be created (a warning will be printed in this case, to remind you of this behavior, in case it is not what you wanted).

When using a file passed to `w_init` using `--bstates-from`, each line in that file defines a state, and contains a label, the probability, and optionally a data reference, separated by whitespace, as in::

```
unbound      1.0
```

or:

```
unbound_0    0.6      state0.pdb
unbound_1    0.4      state1.pdb
```

Basis states can also be supplied at the command line using one or more `--bstate` flags, where the argument matches the format used in the state file above. The total probability summed over all basis states should equal unity, however WESTPA will renormalize the distribution if this condition is not met.

Initial states are the generated from the basis states by optionally applying some perturbation or modification to the basis state. For example if WESTPA was being used to simulate ligand binding, one might want to have a basis state where the ligand was some set distance from the binding partner, and initial states are generated by randomly orienting the ligand at that distance. When using the executable propagator, this is done using the script specified under the `gen_istate` section of the executable configuration. Otherwise, if defining a custom propagator, the user must override the `gen_istate` method of `WESTPropagator`.

When using the executable propagator, the the script specified by `gen_istate` should take the data supplied by the environmental variable `$WEST_BSTATE_DATA_REF` and return the generated initial state to `$WEST_ISTATE_DATA_REF`. If no transform need be performed, the user may simply copy the data directly without modification. This data will then be available via `$WEST_PARENT_DATA_REF` if `$WEST_CURRENT_SEG_INITPOINT_TYPE` is `SEG_INITPOINT_NEWTRAJ`.

6.15.1.4 Target States

WESTPA can be run in a recycling mode in which replicas reaching a target state are removed from the simulation and their weights are assigned to new replicas created from one of the initial states. This mode creates a non-equilibrium steady-state that isolates members of the trajectory ensemble originating in the set of initial states and transitioning to the target states. The flux of probability into the target state is then inversely proportional to the mean first passage time (MFPT) of the transition.

Target states are defined when initializing a WESTPA simulation when calling `w_init`. Target states are specified either in a file specified with `--tstates-from`, or by one or more `--tstate` arguments. If neither `--tstates-from` nor at least one `--tstate` argument is provided, then an equilibrium simulation (without any sinks) will be performed.

Target states can be defined using a text file, where each line defines a state, and contains a label followed by a representative progress coordinate value, separated by whitespace, as in::

```
bound      0.02
```

for a single target and one-dimensional progress coordinates or::

```
bound      2.7      0.0
drift      100      50.0
```

for two targets and a two-dimensional progress coordinate.

The argument associated with `--tstate` is a string of the form `'label, pcoord0 [,pcoord1[,...]]'`, similar to a line in the example target state definition file above. This argument may be specified more than once, in which case the given states are appended to the list of target states for the simulation in the order they appear on the command line, after those that are specified by `--tstates-from`, if any.

WESTPA uses the representative progress coordinate of a target-state and converts the **entire** bin containing that progress coordinate into a recycling sink.

6.15.1.5 Propagators

6.15.1.5.1 The Executable Propagator

6.15.1.5.2 Writing custom propagators

While most users will use the Executable propagator to run dynamics by calling out to an external piece of software, it is possible to write custom propagators that can be used to generate sampling directly through the python interface. This is particularly useful when simulating simple systems, where the overhead of starting up an external program is large compared to the actual cost of computing the trajectory segment. Other use cases might include running sampling with software that has a Python API (e.g. [OpenMM](#)).

In order to create a custom propagator, users must define a class that inherits from `WESTPropagator` and implement three methods:

- `get_pcoord(self, state)`: Get the progress coordinate of the given basis or initial state.
- `gen_istate(self, basis_state, initial_state)`: Generate a new initial state from the given basis state. This method is optional if `gen_istates` is set to `False` in the propagation section of the configuration file, which is the default setting.
- `propagate(self, segments)`: Propagate one or more segments, including any necessary per-iteration setup and teardown for this propagator.

There are also two stubs that, if overridden, provide a mechanism for modifying the simulation before or after the iteration:

- `prepare_iteration(self, n_iter, segments)`: Perform any necessary per-iteration preparation. This is run by the work manager.
- `finalize_iteration(self, n_iter, segments)`: Perform any necessary post-iteration cleanup. This is run by the work manager.

Several examples of custom propagators are available:

- [1D Over-damped Langevin dynamics](#)
- [2D Langevin dynamics](#)
- [Langevin dynamics - CA atom Elastic Network Model](#)

6.15.1.6 Configuration File

The configuration of a WESTPA simulation is specified using a plain text file written in [YAML](#). This file specifies, among many other things, the length of the simulation, which modules should be loaded for specifying the system, how external data should be organized on the file system, and which plugins should be used. [YAML](#) is a hierarchical format and WESTPA organizes the configuration settings into blocks for each component. While below, the configuration file will be referred to as `west.cfg`, the user is free to name the configuration file something else. Most of the scripts and tools that WESTPA provides, however, require that the name of the configuration file be specified if the default name is not used.

The top most heading in `west.cfg` should be specified as::

```
---
west:
  ...
```

with all sub-section specified below it. A complete example can be found for the NaCl example: https://github.com/westpa/westpa/blob/master/lib/examples/nacl_gmx/west.cfg

In the following section, the specifications for each section of the file can be found, along with default parameters and descriptions. Required parameters are indicated as REQUIRED.:

```
---
west:
    ...
    system:
        driver: REQUIRED
        module_path: []
```

The `driver` parameter must be set to a subclass of `WESTSystem`, and given in the form `module.class`. The `module_path` parameter is appended to the system path and indicates where the class is defined.:

```
---
west:
    ...
    we:
        adjust_counts: True
        weight_split_threshold: 2.0
        weight_merge_cutoff: 1.0
```

The `we` section specifies parameters related to the Huber and Kim resampling algorithm. WESTPA implements a variation of the method, in which setting `adjust_counts` to `True` strictly enforces that the number of replicas per bin is exactly `system.bin_target_counts`. Otherwise, the number of replicas per bin is allowed to fluctuate as in the original implementation of the algorithm. Adjusting the counts can improve load balancing for parallel simulations. Replicas with weights greater than `weight_split_threshold` times the ideal weight per bin are tagged as candidates for splitting. Replicas with weights less than `weight_merge_cutoff` times the ideal weight per bin are candidates for merging.:

```
---
west:
    ...
    propagation:
        gen_istates: False
        block_size: 1
        save_transition_matrices: False
        max_run_wallclock: None
        max_total_iterations: None
```

- `gen_istates`: Boolean specifying whether to generate initial states from the basis states. The executable propagator defines a specific configuration block (*add internal link to other section*), and custom propagators should override the `WESTPropagator.gen_istate()` method.
- `block_size`: An integer defining how many segments should be passed to a worker at a time. When using the serial work manager, this value should be set to the maximum number of segments per iteration to avoid significant overhead incurred by the locking mechanism in the `WMFutures` framework. Parallel work managers might benefit from setting this value greater than one in some instances to decrease network communication load.
- `save_transition_matrices`:
- `max_run_wallclock`: A time in `dd:hh:mm:ss` or `hh:mm:ss` specifying the maximum wallclock time of a particular WESTPA run. If running on a batch queuing system, this time should be set to less than the job allocation time to ensure that WESTPA shuts down cleanly.

- **max_total_iterations**: An integer value specifying the number of iterations to run. This parameter is checked against the last completed iteration stored in the HDF5 file, not the number of iterations completed for a specific run. The default value of `None` only stops upon external termination of the code.:

```

---
west:
    ...
    data:
        west_data_file: REQUIRED
        aux_compression_threshold: 1048576
        iter_prec: 8
        datasets:
            -name: REQUIRED
            h5path:
                store: True
                load: False
                dtype:
                    scaleoffset: None
                    compression: None
                    chunks: None
        data_refs:
            segment:
            basis_state:
            initial_state:

```

- **west_data_file**: The name of the main HDF5 data storage file for the WESTPA simulation.
- **aux_compression_threshold**: The threshold in bytes for compressing the auxiliary data in a dataset on an iteration-by-iteration basis.
- **iter_prec**: The length of the iteration index with zero-padding. For the default value, iteration 1 would be specified as `iter_00000001`.
- **datasets**:
- **data_refs**:
- **plugins**
- **executable**

6.15.1.7 Environmental Variables

There are a number of environmental variables that can be set by the user in order to configure a WESTPA simulation:

- **WEST_ROOT**: path to the base directory containing the WESTPA install
- **WEST_SIM_ROOT**: path to the base directory of the WESTPA simulation
- **WEST_PYTHON**: path to python executable to run the WESTPA simulation
- **WEST_PYTHONPATH**: path to any additional modules that WESTPA will require to run the simulation
- **WEST_KERNPROF**: path to `kernprof.py` script to perform line-by-line profiling of a WESTPA simulation (see [python line_profiler](#)). This is only required for users who need to profile specific methods in a running WESTPA simulation.

Work manager related environmental variables:

- **WM_WORK_MANAGER**

- WM_N_WORKERS

WESTPA makes available to any script executed by it (e.g. `runseg.sh`), a number of environmental variables that are set dynamically by the executable propagator from the running simulation.

6.15.1.7.1 Programs executed for an iteration

The following environment variables are passed to programs executed on a per-iteration basis, notably pre-iteration and post-iteration scripts.

Variable	Possible values	Function
WEST_CURRENT_ITER	Integer ≥ 1	Current iteration number

6.15.1.7.2 Programs executed for a segment

The following environment variables are passed to programs executed on a per-segment basis, notably dynamics propagation.

Variable	Possible values	Function
WEST_CURRENT_I	Integer ≥ 1	Current iteration number
WEST_CURRENT_S	Integer ≥ 0	Current segment ID
WEST_CURRENT_S	String	General-purpose reference, based on current segment information, configured in <code>west.cfg</code> . Usually used for storage paths
WEST_CURRENT_S	Enumeration: SEG_INITPOINT_CONTINUES, SEG_INITPOINT_NEWTRAJ	Whether this segment continues a previous trajectory or initiates a new one.
WEST_PARENT_ID	Integer	Segment ID of parent segment. Negative for initial points.
WEST_PARENT_DA	String	General purpose reference, based on parent segment information, configured in <code>west.cfg</code> . Usually used for storage paths
WEST_PCOORD_RE	Filename	Where progress coordinate data must be stored
WEST_RAND16	Integer	16-bit random integer
WEST_RAND32	Integer	32-bit random integer
WEST_RAND64	Integer	64-bit random integer
WEST_RAND128	Integer	128-bit random integer
WEST_RANDFLOAT	Floating-point	Random number in $[0,1)$.

Additionally for any additional datasets specified in the configuration file, WESTPA automatically provides `WEST_X_RETURN`, where X is the uppercase name of the dataset. For example if the configuration file contains the following:

```
data:
...
  datasets: # dataset storage options
    - name: energy
```

WESTPA would make `WEST_ENERGY_RETURN` available.

6.15.1.7.3 Programs executed for a single point

Programs used for creating initial states from basis states (`gen_istate.sh`) or extracting progress coordinates from structures (e.g. `get_pcoord.sh`) are provided the following environment variables:

Variable	Available for	Possible values	Function
WEST_STRUCT_D	All single-point calculations	String	General-purpose reference, usually a pathname, associated with the basis/initial state.
WEST_BSTATE_IC	<code>get_pcoord</code> for basis state, <code>gen_istate</code>	Integer ≥ 0	Basis state ID
WEST_BSTATE_D	<code>get_pcoord</code> for basis state, <code>gen_istate</code>	String	Basis state data reference
WEST_ISTATE_ID	<code>get_pcoord</code> for initial state, <code>gen_istate</code>	Integer ≥ 0	Initial state ID
WEST_ISTATE_DA	<code>get_pcoord</code> for initial state, <code>gen_istate</code>	String	Initial state data references, usually a pathname
WEST_PCOORD_R	<code>get_pcoord</code> for basis or initial state	Pathname	Where progress coordinate data is expected to be found after execution

6.15.1.8 Plugins

WESTPA has a extensible plugin architecture that allows the user to manipulate the simulation at specified points during an iteration.

- Activating plugins in the config file
- Plugin execution order/priority

6.15.1.9 Weighted Ensemble Algorithm (Resampling)

6.15.2 Running

6.15.2.1 Overview

The `w_run` command is used to run weighted ensemble simulations *configured* `<setup>` with `w_init`.

6.15.2.2 Setting simulation limits

6.15.2.3 Running a simulation

6.15.2.3.1 Running on a single node

6.15.2.3.2 Running on multiple nodes with MPI

6.15.2.3.3 Running on multiple nodes with ZeroMQ

6.15.2.4 Managing data

6.15.2.5 Recovering from errors

By default, information about simulation progress is stored in **west-JOBID.log** (where JOBID refers to the job ID given by the submission engine); any errors will be logged here.

- The error “could not read pcoord from ‘tempfile’: progress coordinate has incorrect shape” may come about from multiple causes; it is possible that the progress coordinate length is incorrectly specified in `system.py` (`self.pcoord_len`), or that GROMACS (or whatever simulation package you are using) had an error during the simulation.
- The first case will be obvious by what comes after the message: (XX, YY) (where XX is non-zero), expected (ZZ, GG) (whatever is in `system.py`). This can be corrected by adjusting `system.py`.
- In the second case, the progress coordinate length is 0; this indicates that no progress coordinate data exists (null string), which implies that the simulation software did not complete successfully. By default, the simulation package (GROMACS or otherwise) terminal output is stored in a log file inside of `seg_logs`. Any error that occurred during the actual simulation will be logged here, and can be corrected as needed.

6.15.3 Analysis

6.15.3.1 Gauging simulation progress and convergence

6.15.3.1.1 Progress coordinate distribution (`w_pcpdist`)

`w_pcpdist` and `plothist`

6.15.3.1.2 Kinetics for source/sink simulations

`w_fluxanl`

6.15.3.1.3 Kinetics for arbitrary state definitions

In order to calculate rate constants, it is necessary to run three different tools:

```
- :ref:`w_assign`
- :ref:`w_kinetics`
- :ref:`w_kinavg`
```

The `w_assign` tool assigns trajectories to states (states which correspond to a target bin) at a sub-tau resolution. This allows `w_kinetics` to properly trace the trajectories and prepare the data for further analysis.

Although the bin and state definitions can be pulled from the system, it is frequently more convenient to specify custom bin boundaries and states; this eliminates the need to know what constitutes a state prior to starting the simulation. Both files must be in the YAML format, of which there are numerous examples of online. A quick example for each file follows:

```
States:
---
states:
  - label: unbound
    coords:
      - [25,0]
  - label: boun
    coords:
      - [1.5,33.0]

Bins:
---
bins:
  type: RectilinearBinMapper
  boundaries: [[0.0,1.57,25.0,10000],[0.0,33.0,10000]]
```

This system has a two dimensional progress coordinate, and two definite states, as defined by the PMF. The binning used during the simulation was significantly more complex; defining a smaller progress coordinate (in which we have three regions: bound, unbound, and in between) is simply a matter of convenience. Note that these custom bins do not change the simulation in any fashion; you can adjust state definitions and bin boundaries at will without altering the way the simulation runs.

The help definition, included by running:

```
w_assign --help
```

usually contains the most up-to-date help information, and so more information about command line options can be obtained from there. To run with the above YAML files, assuming they are named `STATES` and `BINS`, you would run the following command:

```
w_assign --states-from-file STATES --bins-from-file BINS
```

By default, this produces a `.h5` file (named `assign.h5`); this can be changed via the command line.

The `w_kinetics` tool uses the information generated from `w_assign` to trace through trajectories and calculate flux with included color information. There are two main methods to run `w_kinetics`:

```
w_kinetics trace
w_kinetics matrix
```

The matrix method is still in development; at this time, trace is the recommended method.

Once the `w_kinetics` analysis is complete, you can check for convergence of the rate constants. WESTPA includes two tools to help you do this: `w_kinavg` and `ploterr`. First, begin by running the following command (keep in mind that `w_kinavg` has the same type of analysis as `w_kinetics` does; whatever method you chose (trace or matrix) in the `w_kinetics` step should be used here, as well):

```
w_kinavg trace -e cumulative
```

This instructs `w_kinavg` to produce a .h5 file with the cumulative rate information; by then using `ploterr`, you can determine whether the rates have stopped changing:

```
ploterr kinavg
```

By default, this produces a set of .pdf files, containing cumulative rate and flux information for each state-to-state transition as a function of the WESTPA iteration. Determine at which iteration the rate stops changing; then, rerun `w_kinavg` with the following systems:

```
w_kinavg trace --first-iter ITER
```

where ITER is the beginning of the unchanging region. This will then output information much like the following:

```
fluxes into macrostates:
unbound: mean=1.712580005863456e-02 CI=(1.596595628304422e-02, 1.808249529394858e-
↪02) * tau^-1
bound : mean=5.944989301935855e-04 CI=(4.153556214886056e-04, 7.789568983584020e-
↪04) * tau^-1

fluxes from state to state:
unbound -> bound : mean=5.944989301935855e-04 CI=(4.253003401668849e-04, 7.
↪720997503648696e-04) * tau^-1
bound -> unbound: mean=1.712580005863456e-02 CI=(1.590547796439216e-02, 1.
↪808154616175579e-02) * tau^-1

rates from state to state:
unbound -> bound : mean=9.972502012305491e-03 CI=(7.165030136921814e-03, 1.
↪313767180582492e-02) * tau^-1
bound -> unbound: mean=1.819520888349874e-02 CI=(1.704608273094848e-02, 1.
↪926165865735958e-02) * tau^-1
```

Divide by tau to calculate your rate constant.

6.16 WEST Tools

The command line tools included with the WESTPA software package are broadly separable into two categories: **Tools for initializing a simulation** and **tools for analyzing results**.

Command function can be user defined and modified. The particular parameters of different command line tools are specified, in order of precedence, by:

- User specified command line arguments
- User defined environmental variables
- Package defaults

This page focuses on outlining the general functionality of the command line tools and providing an overview of command line arguments that are shared by multiple tools. See the [index of command-line tools](#) for a more comprehensive overview of each tool.

6.16.1 Overview

All tools are located in the `$WEST_ROOT/bin` directory, where the shell variable `WEST_ROOT` points to the path where the WESTPA package is located on your machine.

You may wish to set this variable automatically by adding the following to your `~/.bashrc` or `~/.profile` file:

```
export WEST_ROOT="$HOME/westpa"
```

where the path to the westpa suite is modified accordingly.

6.16.1.1 Tools for setting up and running a simulation

Use the following commands to initialize, configure, and run a weighted ensemble simulation. Command line arguments or environmental variables can be set to specify the work managers for running the simulation, where configuration data is read from, and the *HDF5* file in which results are stored.

Com-mand	Function
<code>w_init</code>	Initializes simulation configuration files and environment. Always run this command before starting a new simulation.
<code>w_bins</code>	Set up binning, progress coordinate
<code>w_run</code>	Launches a simulation. Command arguments/environmental variables can be included to specify the work managers and simulation parameters
<code>w_truncate</code>	Truncates the weighted ensemble simulation from a given iteration.

6.16.1.2 Tools for analyzing simulation results

The following command line tools are provided for analysis after running a weighted ensemble simulation (and collecting the results in an *HDF5* file).

With the exception of the plotting tool `plothist`, all analysis tools read from and write to *HDF5* type files.

Com-mand	Function
<code>w_assign</code>	Assign walkers to bins and macrostates (using simulation output as input). Must be done before some other analysis tools (e.g. <code>w_kinetics</code> , <code>w_kinavg</code>)
<code>w_trace</code>	Trace the path of a given walker segment over a user-specified number of simulation iterations.
<code>w_fluxani</code>	Calculate average probability flux into user-defined ‘target’ state with relevant statistics.
<code>w_pdist</code>	Construct a probability distribution of results (e.g. progress coordinate membership) for subsequent plotting with <code>plothist</code> .
<code>plothist</code>	Tool to plot output from other analysis tools (e.g. <code>w_pdist</code>).

6.16.2 General Command Line Options

The following arguments are shared by all command line tools:

```
-r config file, --rcfile config file
  Use config file as the configuration file (Default: File named west.cfg)
--quiet, --verbose, --debug
  Specify command tool output verbosity (Default: 'quiet' mode)
--version
  Print WESTPA version number and exit
-h, --help
  Output the help information for this command line tool and exit
```

6.16.2.1 A note on specifying a configuration file

A *configuration file*, which should be stored in your simulation root directory, is read by all command line tools. The *configuration file* specifies parameters for general simulation setup, as well as the *hdf5* file name where simulation data is stored and read by analysis tools.

If not specified, the **default configuration file** is assumed to be named **west.cfg**.

You can override this to use configuration file *file* by either:

- Setting the environmental variable WESTRC equal to *file*:

```
export WESTRC=/path/to/westrcfile
```

- Including the command line argument `-r /path/to/westrcfile`

6.16.3 Work Manager Options

Note: See *wwwmgr overview* for a more detailed explanation of the work manager framework.

Work managers are used by a number of command-line tools to process more complex tasks, especially in setting up and running simulations (i.e. *w_init* and *w_run*) - in general, work managers are involved in tasks that require multiprocessing and/or tasks distributed over multiple nodes in a cluster.

6.16.3.1 Overview

The following command-line tools make use of work managers:

- *w_init*
- *w_run*

6.16.3.2 General work manager options

The following are general options used for specifying the type of work manager and number of cores:

```
--wm-work-manager work_manager
Specify which type of work manager to use, where the possible choices for
work_manager are: {processes, gcserial, threads, mpi, or zmq}. See the
wmmgr overview page <wmmgr>_ for more information on the different types of
work managers (Default: gcprocesses)

--wm-n-workers n_workers
Specify the number of cores to use as gcn_workers, if the work manager you
selected supports this option (work managers that do not will ignore this
option). If using an gcmpi or zmq work manager, specify gc--wm-n-workers=0
for a dedicated server (Default: Number of cores available on machine)
```

The mpi work manager is generally sufficient for most tasks that make use of multiple nodes on a cluster. The zmq work manager is preferable if the mpi work manager does not work properly on your cluster or if you prefer to have more explicit control over the distribution of communication tasks on your cluster.

6.16.3.3 ZeroMQ ('zmq') work manager

The ZeroMQ work manager offers a number of additional options (all of which are optional and have default values). All of these options focus on whether the zmq work manager is set up as a server (i.e. task distributor/ventilator) or client (task processor):

```
--wm-zmq-mode mode
Options: {server or client}. Specify whether the ZMQ work manager on this
node will operate as a server or a client (Default: server)

--wm-zmq-info-file info_file
Specify the name of a temporary file to write (as a server) or read (as a
client) socket connection endpoints (Default: server_x.json, where x is a
unique identifier string)

--wm-zmq-task-endpoint task_endpoint
Explicitly use task_endpoint to bind to (as server) or connect to (as
client) for task distribution (Default: A randomly determined endpoint that
is written or read from the specified info_file)

--wm-zmq-result-endpoint result_endpoint
Explicitly use result_endpoint to bind to (as server) or connect to (as
client) to distribute and collect task results (Default: A randomly
determined endpoint that is written to or read from the specified
info_file)

--wm-zmq-announce-endpoint announce_endpoint
Explicitly use announce_endpoint to bind to (as server) or connect to (as
client) to distribute central announcements (Default: A randomly determined
endpoint that is written to or read from the specified info_file)

--wm-zmq-heartbeat-interval interval
If a server, send an Im alive ping to connected clients every interval
seconds; If a client, expect to hear a server ping every approximately
```

(continues on next page)

(continued from previous page)

```

interval seconds, or else assume the server has crashed and shutdown
(Default: 600 seconds)

--wm-zmq-task-timeout timeout
Kill worker processes/jobs after that take longer than timeout seconds to
complete (Default: no time limit)

--wm-zmq-client-comm-mode mode
Use the communication mode, mode, (options: {ipc for Unix sockets, or tcp
for TCP/IP sockets}) to communicate with worker processes (Default: ipc)

```

6.16.4 Initializing/Running Simulations

For a more complete overview of all the files necessary for setting up a simulation, see the *user guide for setting up a simulation*

6.17 WEST Work Manager

6.17.1 Introduction

WWMGR is the parallel task distribution framework originally included as part of the WEMD source. It was extracted to permit independent development, and (more importantly) independent testing. A number of different schemes can be selected at run-time for distributing work across multiple cores/nodes, as follows:

Name	Implementation	Multi-Core	Multi-Node	Appropriate For
serial	None	No	No	Testing, minimizing overhead when dynamics is inexpensive
threac	Python “threading” module	Yes	No	Dynamics propagated by external executables, large amounts of data transferred per segment
processes	Python “multiprocessing” module	Yes	No	Dynamics propagated by Python routines, modest amounts of data transferred per segment
mpi	<code>mpi4py</code> compiled and linked against system MPI	Yes	Yes	Distributing calculations across multiple nodes. Start with this on your cluster of choice.
zmq	<code>ZeroMQ</code> and <code>PyZMQ</code>	Yes	Yes	Distributing calculations across multiple nodes. Use this if MPI does not work properly on your cluster (particularly for spawning child processes).

6.17.2 Environment variables

6.17.2.1 For controlling task distribution

While the original WEMD work managers were controlled by command-line options and entries in `wemd.cfg`, the new work manager is controlled using command-line options or environment variables (much like OpenMP). These variables are as follow:

Variable	Appli- cable to	Default	Meaning
WM_WORK	(none)	processes	Use the given task distribution system: “serial”, “threads”, “processes”, or “zmq”
WM_N_WORKERS	threads, pro- cesses, zmq	number of cores in machine	Use this number of workers. In the case of zmq, use this many workers on the current machine only (can be set independently on different nodes).
WM_ZMQ_I	zmq	server	Start as a server (“server”) or a client (“client”). Servers coordinate a given calculation, and clients execute tasks related to that calculation.
WM_ZMQ_T	zmq	60	Time (in seconds) after which a worker will be considered hung, terminated, and restarted. This must be updated for long-running dynamics segments. Set to zero to disable hang checks entirely.
WM_ZMQ_P	zmq	Random port	Master distributes tasks at this address
WM_ZMQ_R	zmq	Random port	Master receives task results at this address
WM_ZMQ_A	zmq	Random port	Master publishes announcements (such as “shut down now”) at this address
WM_ZMQ_S	zmq	zmq_server_info_PID_ID. json (where PID is a process ID and ID is a nearly random hex number)	A file describing the above endpoints can be found here (to ease cluster-wide startup)

6.17.2.2 For passing information to workers

One environment variable is made available by multi-process work managers (processes and ZMQ) to help clients configure themselves (e.g. select an appropriate GPU on a multi-GPU node):

Variable	Applica- ble to	Meaning
WM_PROCESS_ID	processes, zmq	Contains an integer, 0 based, identifying the process among the set of processes started on a given node.

6.17.3 The ZeroMQ work manager for clusters

The ZeroMQ (“zmq”) work manager can be used for both single-machine and cluster-wide communication. Communication occurs over sockets using the [ZeroMQ](#) messaging protocol. Within nodes, [Unix sockets](#) are used for efficient communication, while between nodes, TCP sockets are used. This also minimizes the number of open sockets on the master node.

The quick and dirty guide to using this on a cluster is as follows:

```
source env.sh
export WM_WORK_MANAGER=zmq
export WM_ZMQ_COMM_MODE=tcp
export WM_ZMQ_SERVER_INFO=$WEST_SIM_ROOT/wemd_server_info.json

w_run &

# manually run w_run on each client node, as appropriate for your batch system
# e.g. qssh -inherit for Grid Engine, or maybe just simple SSH

for host in $(cat $TMPDIR/machines | sort | uniq); do
    qssh -inherit -V $host $PWD/node-ltcl.sh &
done
```

6.18 WEST Extensions

6.18.1 Post-Analysis Reweighting

6.18.2 String Method

6.18.3 Weighted Ensemble Equilibrium Dynamics

6.18.4 Weighted Ensemble Steady State

6.19 Command Line Tool Index

6.19.1 w_init

usage:

```
w_init [-h] [-r RCFILE] [--quiet | --verbose | --debug] [--version] [--force]
      [--bstate-file BSTATE_FILE] [--bstate BSTATES] [--tstate-file TSTATE_FILE]
      [--tstate TSTATES] [--segs-per-state N] [--no-we]
      [--serial | --parallel | --work-manager WORK_MANAGER] [--n-workers N_
↪WORKERS]
      [--zmq-mode MODE] [--zmq-comm-mode COMM_MODE] [--zmq-write-host-info INFO_
↪FILE]
      [--zmq-read-host-info INFO_FILE] [--zmq-upstream-rr-endpoint ENDPOINT]
      [--zmq-upstream-ann-endpoint ENDPOINT] [--zmq-downstream-rr-endpoint_
↪ENDPOINT]
      [--zmq-downstream-ann-endpoint ENDPOINT] [--zmq-master-heartbeat MASTER_
```

(continues on next page)

(continued from previous page)

```

↪HEARTBEAT]
    [--zmq-worker-heartbeat WORKER_HEARTBEAT] [--zmq-timeout-factor FACTOR]
    [--zmq-startup-timeout STARTUP_TIMEOUT] [--zmq-shutdown-timeout SHUTDOWN_
↪TIMEOUT]

```

Initialize a new WEST simulation, creating the WEST HDF5 file and preparing the first iteration's segments. Initial states are generated from one or more "basis states" which are specified either in a file specified with `--bstates-from`, or by one or more `--bstate` arguments. If neither `--bstates-from` nor at least one `--bstate` argument is provided, then a default basis state of probability one identified by the state ID zero and label "basis" will be created (a warning will be printed in this case, to remind you of this behavior, in case it is not what you wanted). Target states for (non-equilibrium) steady-state simulations are specified either in a file specified with `--tstates-from`, or by one or more `--tstate` arguments. If neither `--tstates-from` nor at least one `--tstate` argument is provided, then an equilibrium simulation (without any sinks) will be performed.

optional arguments:

```

-h, --help            show this help message and exit
--force              Overwrite any existing simulation data
--bstate-file BSTATE_FILE, --bstates-from BSTATE_FILE
                    Read basis state names, probabilities, and (optionally) data
↪references from
                    BSTATE_FILE.
--bstate BSTATES      Add the given basis state (specified as a string 'label,
↪probability[,auxref]')
                    to the list of basis states (after those specified in --bstates-
↪from, if any).
                    This argument may be specified more than once, in which case the
↪given states
                    are appended in the order they are given on the command line.
--tstate-file TSTATE_FILE, --tstates-from TSTATE_FILE
                    Read target state names and representative progress coordinates
↪from
                    TSTATE_FILE
--tstate TSTATES      Add the given target state (specified as a string
                    'label,pcoord0[,pcoord1[,...]]') to the list of target states
↪(after those
                    specified in the file given by --tstates-from, if any). This
↪argument may be
                    specified more than once, in which case the given states are
↪appended in the
                    order they appear on the command line.
--segs-per-state N    Initialize N segments from each basis state (default: 1).
--no-we, --shotgun    Do not run the weighted ensemble bin/split/merge algorithm on
↪newly-created
                    segments.

```

general options:

```

-r RCFILE, --rcfile RCFILE
                    use RCFILE as the WEST run-time configuration file (default: west.
↪cfg)
--quiet              emit only essential information
--verbose            emit extra information

```

(continues on next page)

(continued from previous page)

```
--debug          enable extra checks and emit copious information
--version        show program's version number and exit
```

parallelization options:

```
--serial          run in serial mode
--parallel        run in parallel mode (using processes)
--work-manager WORK_MANAGER
                  use the given work manager for parallel task distribution.
↪ Available work  managers are ('serial', 'threads', 'processes', 'zmq'); default is
↪ 'serial'
--n-workers N_WORKERS
                  Use up to N_WORKERS on this host, for work managers which support
↪ this option.    Use 0 for a dedicated server. (Ignored by work managers which do
↪ not support     this option.)
```

options for ZeroMQ (“zmq”) work manager (master or node):

- zmq-mode MODE** Operate as a master (server) or a node (workers/client). “server” is a deprecated synonym for “master” and “client” is a deprecated synonym for “node”.
- zmq-comm-mode COMM_MODE** Use the given communication mode – TCP or IPC (Unix-domain) – sockets for communication within a node. IPC (the default) may be more efficient but is not available on (exceptionally rare) systems without node-local storage (e.g. /tmp); on such systems, TCP may be used instead.
- zmq-write-host-info INFO_FILE** Store hostname and port information needed to connect to this instance in INFO_FILE. This allows the master and nodes assisting in coordinating the communication of other nodes to choose ports randomly. Downstream nodes read this file with `--zmq-read-host-info` and know where how to connect.
- zmq-read-host-info INFO_FILE** Read hostname and port information needed to connect to the master (or other coordinating node) from INFO_FILE. This allows the master and nodes assisting in coordinating the communication of other nodes to choose ports randomly, writing that information with `--zmq-write-host-info` for this instance to read.
- zmq-upstream-rr-endpoint ENDPOINT** ZeroMQ endpoint to which to send request/response (task and result) traffic toward the master.
- zmq-upstream-ann-endpoint ENDPOINT** ZeroMQ endpoint on which to receive announcement (heartbeat and shutdown notification) traffic from the master.
- zmq-downstream-rr-endpoint ENDPOINT** ZeroMQ endpoint on which to listen for request/response (task and result) traffic from subsidiary workers.
- zmq-downstream-ann-endpoint ENDPOINT** ZeroMQ endpoint on which to send announcement (heartbeat and shutdown notification) traffic toward workers.
- zmq-master-heartbeat MASTER_HEARTBEAT** Every MASTER_HEARTBEAT seconds, the master announces its presence to workers.

- zmq-worker-heartbeat** **WORKER_HEARTBEAT** Every **WORKER_HEARTBEAT** seconds, workers announce their presence to the master.
- zmq-timeout-factor** **FACTOR** Scaling factor for heartbeat timeouts. If the master doesn't hear from a worker in **WORKER_HEARTBEAT*FACTOR**, the worker is assumed to have crashed. If a worker doesn't hear from the master in **MASTER_HEARTBEAT*FACTOR** seconds, the master is assumed to have crashed. Both cases result in shutdown.
- zmq-startup-timeout** **STARTUP_TIMEOUT** Amount of time (in seconds) to wait for communication between the master and at least one worker. This may need to be changed on very large, heavily-loaded computer systems that start all processes simultaneously.
- zmq-shutdown-timeout** **SHUTDOWN_TIMEOUT** Amount of time (in seconds) to wait for workers to shut down.

6.19.2 w_bins

w_bins deals with binning modification and statistics

6.19.2.1 Overview

Usage:

```
$WEST_ROOT/bin/w_bins [-h] [-r RCFILE] [--quiet | --verbose | --debug] [--version]
                        [-W WEST_H5FILE]
                        {info,rebin} ...
```

Display information and statistics about binning in a WEST simulation, or modify the binning for the current iteration of a WEST simulation.

6.19.2.2 Command-Line Options

See the [general command-line tool reference](#) for more information on the general options.

6.19.2.2.1 Options Under 'info'

Usage:

```
$WEST_ROOT/bin/w_bins info [-h] [-n N_ITER] [--detail]
                        [--bins-from-system | --bins-from-expr BINS_FROM_EXPR | --bins-from-
↪function BINS_FROM_FUNCTION | --bins-from-file]
```

Positional options:

```
info
    Display information about binning.
```

Options for 'info':

```
-n N_ITER, --n-iter N_ITER
  Consider initial points of segment N_ITER (default: current
  iteration).

--detail
  Display detailed per-bin information in addition to summary
  information.
```

Binning options for ‘info’:

```
--bins-from-system
  Bins are constructed by the system driver specified in the WEST
  configuration file (default where stored bin definitions not
  available).

--bins-from-expr BINS_FROM_EXPR, --binbounds BINS_FROM_EXPR
  Construct bins on a rectilinear grid according to the given BINEXPR.
  This must be a list of lists of bin boundaries (one list of bin
  boundaries for each dimension of the progress coordinate), formatted
  as a Python expression. E.g. "[[0,1,2,4,inf],[-inf,0,inf]]". The
  numpy module and the special symbol "inf" (for floating-point
  infinity) are available for use within BINEXPR.

--bins-from-function BINS_FROM_FUNCTION, --binfunc BINS_FROM_FUNCTION
  Supply an external function which, when called, returns a properly
  constructed bin mapper which will then be used for bin assignments.
  This should be formatted as "[PATH:]MODULE.FUNC", where the function
  FUNC in module MODULE will be used; the optional PATH will be
  prepended to the module search path when loading MODULE.

--bins-from-file
  Load bin specification from the data file being examined (default
  where stored bin definitions available).
```

6.19.2.2.2 Options Under ‘rebin’

Usage:

```
$WEST_ROOT/bin/w_bins rebin [-h] [--confirm] [--detail]
                        [--bins-from-system | --bins-from-expr BINS_FROM_EXPR | --bins-from-
↪function BINS_FROM_FUNCTION]
                        [--target-counts TARGET_COUNTS | --target-counts-from FILENAME]
```

Positional option:

```
rebin
  Rebuild current iteration with new binning.
```

Options for ‘rebin’:

```
--confirm
  Commit the revised iteration to HDF5; without this option, the
```

(continues on next page)

(continued from previous page)

effects of the new binning are only calculated **and** printed.

`--detail`

Display detailed per-**bin** information **in** addition to summary information.

Binning options for ‘rebin’;

Same as the binning options for ‘info’.

Bin target count options for ‘rebin’;

`--target-counts TARGET_COUNTS`

Use TARGET_COUNTS instead of stored **or** system driver target counts. TARGET_COUNTS **is** a comma-separated **list** of integers. As a special case, a single integer **is** acceptable, **in** which case the same target count **is** used **for** all bins.

`--target-counts-from FILENAME`

Read target counts **from the** text file FILENAME instead of using stored **or** system driver target counts. FILENAME must contain a **list** of integers, separated by arbitrary whitespace (including newlines).

6.19.2.3 Input Options

`-W WEST_H5FILE, --west_data WEST_H5FILE`

Take WEST data **from** WEST_H5FILE (default: read **from the** HDF5 file specified **in** west.cfg).

6.19.2.4 Examples

(TODO: Write up an example)

6.19.3 w_run

usage:

`w_run [-h]`

Start/continue a WEST simulation

optional arguments:

<code>-h, --help</code>	show this help message and exit
<code>--oneseg</code>	only propagate one segment (useful for debugging propagators)

general options:

`-r RCFILE, --rcfile RCFILE`
 use RCFILE **as** the WEST run-time configuration file (default: west.
`↪cfg`)

(continues on next page)

(continued from previous page)

```
--quiet          emit only essential information
--verbose        emit extra information
--debug         enable extra checks and emit copious information
--version        show program's version number and exit
```

parallelization options:

```
--serial          run in serial mode
--parallel        run in parallel mode (using processes)
--work-manager WORK_MANAGER
                  use the given work manager for parallel task distribution.
↳ Available work managers are ('serial', 'threads', 'processes', 'zmq'); default is
↳ 'serial'
--n-workers N_WORKERS
                  Use up to N_WORKERS on this host, for work managers which support
↳ this option.
                  Use 0 for a dedicated server. (Ignored by work managers which do
↳ not support
                  this option.)
```

options for ZeroMQ (“zmq”) work manager (master or node):

```
--zmq-mode MODE   Operate as a master (server) or a node (workers/client). "server"
↳ is a
                  deprecated synonym for "master" and "client" is a deprecated
↳ synonym for
                  "node".
--zmq-comm-mode COMM_MODE
                  Use the given communication mode -- TCP or IPC (Unix-domain) --
↳ sockets for
                  communication within a node. IPC (the default) may be more
↳ efficient but is not
                  available on (exceptionally rare) systems without node-local
↳ storage (e.g.
                  /tmp); on such systems, TCP may be used instead.
--zmq-write-host-info INFO_FILE
                  Store hostname and port information needed to connect to this
↳ instance in
                  INFO_FILE. This allows the master and nodes assisting in
↳ coordinating the
                  communication of other nodes to choose ports randomly. Downstream
↳ nodes read
                  this file with --zmq-read-host-info and know where how to connect.
--zmq-read-host-info INFO_FILE
                  Read hostname and port information needed to connect to the master
↳ (or other
                  coordinating node) from INFO_FILE. This allows the master and
↳ nodes assisting
                  in coordinating the communication of other nodes to choose ports
↳ randomly,
                  writing that information with --zmq-write-host-info for this
```

(continues on next page)

(continued from previous page)

```

↪instance to read.
--zmq-upstream-rr-endpoint ENDPOINT
    ZeroMQ endpoint to which to send request/response (task and
↪result) traffic
    toward the master.
--zmq-upstream-ann-endpoint ENDPOINT
    ZeroMQ endpoint on which to receive announcement (heartbeat and
↪shutdown
    notification) traffic from the master.
--zmq-downstream-rr-endpoint ENDPOINT
    ZeroMQ endpoint on which to listen for request/response (task and
↪result)
    traffic from subsidiary workers.
--zmq-downstream-ann-endpoint ENDPOINT
    ZeroMQ endpoint on which to send announcement (heartbeat and
↪shutdown
    notification) traffic toward workers.
--zmq-master-heartbeat MASTER_HEARTBEAT
    Every MASTER_HEARTBEAT seconds, the master announces its presence
↪to workers.
--zmq-worker-heartbeat WORKER_HEARTBEAT
    Every WORKER_HEARTBEAT seconds, workers announce their presence to
↪the master.
--zmq-timeout-factor FACTOR
    Scaling factor for heartbeat timeouts. If the master doesn't hear
↪from a worker
    in WORKER_HEARTBEAT*FACTOR, the worker is assumed to have crashed.
↪If a worker
    doesn't hear from the master in MASTER_HEARTBEAT*FACTOR seconds,
↪the master is
    assumed to have crashed. Both cases result in shutdown.
--zmq-startup-timeout STARTUP_TIMEOUT
    Amount of time (in seconds) to wait for communication between the
↪master and at
    least one worker. This may need to be changed on very large,
↪heavily-loaded
    computer systems that start all processes simultaneously.
--zmq-shutdown-timeout SHUTDOWN_TIMEOUT
    Amount of time (in seconds) to wait for workers to shut down.

```

6.19.4 w_truncate

NOTE: w_truncate only deletes iteration groups from the HDF5 data store. It is recommended that any iteration data saved to the file system (e.g. in the traj_segs directory) is deleted or moved for the corresponding iterations.

usage:

```
w_truncate [-h] [-r RCFILE] [--quiet | --verbose | --debug] [--version] [-n N_ITER]
```

Remove all iterations after a certain point in a WESTPA simulation.

optional arguments:

```
-h, --help          show this help message and exit
-n N_ITER, --iter N_ITER
                    Truncate this iteration and those following.
```

general options:

```
-r RCFILE, --rcfile RCFILE
                    use RCFILE as the WEST run-time configuration file (default: west.
↳cfg)
--quiet            emit only essential information
--verbose         emit extra information
--debug           enable extra checks and emit copious information
--version         show program's version number and exit
```

6.19.5 w_fork

usage:

```
w_fork [-h] [-r RCFILE] [--quiet | --verbose | --debug] [--version] [-i INPUT_H5FILE]
      [-I N_ITER] [-o OUTPUT_H5FILE] [--istate-map ISTATE_MAP] [--no-headers]
```

Prepare a new weighted ensemble simulation from an existing one at a particular point. A new HDF5 file is generated. In the case of executable propagation, it is the user's responsibility to prepare the new simulation directory appropriately, particularly making the old simulation's restart data from the appropriate iteration available as the new simulations initial state data; a mapping of old simulation segment to new simulation initial states is created, both in the new HDF5 file and as a flat text file, to aid in this. Target states and basis states for the new simulation are taken from those in the original simulation.

optional arguments:

```
-h, --help          show this help message and exit
-i INPUT_H5FILE, --input INPUT_H5FILE
                    Create simulation from the given INPUT_H5FILE (default: read from
↳configuration
                    file.
-I N_ITER, --iteration N_ITER
                    Take initial distribution for new simulation from iteration N_ITER
↳(default:
                    last complete iteration).
-o OUTPUT_H5FILE, --output OUTPUT_H5FILE
                    Save new simulation HDF5 file as OUTPUT (default: forked.h5).
--istate-map ISTATE_MAP
                    Write text file describing mapping of existing segments to new
↳initial states
                    in ISTATE_MAP (default: istate_map.txt).
--no-headers       Do not write header to ISTATE_MAP
```

general options:

```
-r RCFILE, --rcfile RCFILE
                    use RCFILE as the WEST run-time configuration file (default: west.
↳cfg)
--quiet            emit only essential information
```

(continues on next page)

(continued from previous page)

```
--verbose      emit extra information
--debug        enable extra checks and emit copious information
--version      show program's version number and exit
```

6.19.6 w_assign

usage:

```
w_assign [-h] [-r RCFILE] [--quiet | --verbose | --debug] [--version]
          [--max-queue-length MAX_QUEUE_LENGTH] [-W WEST_H5FILE]
          [--bins-from-system | --bins-from-expr BINS_FROM_EXPR | --bins-from-
↪function BINS_FROM_FUNCTION | --bins-from-file BINFILE | --bins-from-h5file]
          [--construct-dataset CONSTRUCT_DATASET | --dsspecs DSSPEC [DSSPEC ...]]
          [--states STATEDEF [STATEDEF ...] | --states-from-file STATEFILE |
          --states-from-function STATEFUNC] [-o OUTPUT] [--subsample] [--config-
↪from-file]
          [--scheme-name SCHEME] [--serial | --parallel | --work-manager WORK_
↪MANAGER]
          [--n-workers N_WORKERS] [--zmq-mode MODE] [--zmq-comm-mode COMM_MODE]
          [--zmq-write-host-info INFO_FILE] [--zmq-read-host-info INFO_FILE]
          [--zmq-upstream-rr-endpoint ENDPOINT] [--zmq-upstream-ann-endpoint_
↪ENDPOINT]
          [--zmq-downstream-rr-endpoint ENDPOINT] [--zmq-downstream-ann-endpoint_
↪ENDPOINT]
          [--zmq-master-heartbeat MASTER_HEARTBEAT] [--zmq-worker-heartbeat WORKER_
↪HEARTBEAT]
          [--zmq-timeout-factor FACTOR] [--zmq-startup-timeout STARTUP_TIMEOUT]
          [--zmq-shutdown-timeout SHUTDOWN_TIMEOUT]
```

Assign walkers to bins, producing a file (by default named “assign.h5”) which can be used in subsequent analysis.

For consistency in subsequent analysis operations, the entire dataset must be assigned, even if only a subset of the data will be used. This ensures that analyses that rely on tracing trajectories always know the originating bin of each trajectory.

6.19.6.1 Source data

Source data is provided either by a user-specified function (`--construct-dataset`) or a list of “data set specifications” (`--dsspecs`). If neither is provided, the progress coordinate dataset “pcoord” is used.

To use a custom function to extract or calculate data whose probability distribution will be calculated, specify the function in standard Python `MODULE.FUNCTION` syntax as the argument to `--construct-dataset`. This function will be called as `function(n_iter,iter_group)`, where `n_iter` is the iteration whose data are being considered and `iter_group` is the corresponding group in the main WEST HDF5 file (`west.h5`). The function must return data which can be indexed as `[segment][timepoint][dimension]`.

To use a list of data set specifications, specify `--dsspecs` and then list the desired datasets one-by-one (space-separated in most shells). These data set specifications are formatted as `NAME[,file=FILENAME,slice=SLICE]`, which will use the dataset called `NAME` in the HDF5 file `FILENAME` (defaulting to the main WEST HDF5 file `west.h5`), and slice it with the Python slice expression `SLICE` (as in `[0:2]` to select the first two elements of the first axis of the dataset). The `slice` option is most useful for selecting one column (or more) from a multi-column dataset, such as arises when using a progress coordinate of multiple dimensions.

6.19.6.2 Specifying macrostates

Optionally, kinetic macrostates may be defined in terms of sets of bins. Each trajectory will be labeled with the kinetic macrostate it was most recently in at each timepoint, for use in subsequent kinetic analysis. This is required for all kinetics analysis (w_kintrace and w_kinmat).

There are three ways to specify macrostates:

1. States corresponding to single bins may be identified on the command line using the `--states` option, which takes multiple arguments, one for each state (separated by spaces in most shells). Each state is specified as a coordinate tuple, with an optional label prepended, as in `bound: 1.0` or `unbound: (2.5, 2.5)`. Unlabeled states are named `stateN`, where `N` is the (zero-based) position in the list of states supplied to `--states`.
2. States corresponding to multiple bins may use a YAML input file specified with `--states-from-file`. This file defines a list of states, each with a name and a list of coordinate tuples; bins containing these coordinates will be mapped to the containing state. For instance, the following file:

```
---
states:
  - label: unbound
    coords:
      - [9.0, 1.0]
      - [9.0, 2.0]
  - label: bound
    coords:
      - [0.1, 0.0]
```

produces two macrostates: the first state is called “unbound” and consists of bins containing the (2-dimensional) progress coordinate values (9.0, 1.0) and (9.0, 2.0); the second state is called “bound” and consists of the single bin containing the point (0.1, 0.0).

3. Arbitrary state definitions may be supplied by a user-defined function, specified as `--states-from-function=MODULE.FUNCTION`. This function is called with the bin mapper as an argument (`function(mapper)`) and must return a list of dictionaries, one per state. Each dictionary must contain a vector of coordinate tuples with key “coords”; the bins into which each of these tuples falls define the state. An optional name for the state (with key “label”) may also be provided.

6.19.6.3 Output format

The output file (`-o/-output`, by default “`assign.h5`”) contains the following attributes/datasets:

```
``nbins`` attribute
*(Integer)* Number of valid bins. Bin assignments range from 0 to
*nbins*-1, inclusive.

``nstates`` attribute
*(Integer)* Number of valid macrostates (may be zero if no such states are
specified). Trajectory ensemble assignments range from 0 to *nstates*-1,
inclusive, when states are defined.

``/assignments`` [iteration][segment][timepoint]
*(Integer)* Per-segment and -timepoint assignments (bin indices).

``/npts`` [iteration]
*(Integer)* Number of timepoints in each iteration.
```

(continues on next page)

(continued from previous page)

```

``/nsegs`` [iteration]
  *(Integer)* Number of segments in each iteration.

``/labeled_populations`` [iterations][state][bin]
  *(Floating-point)* Per-iteration and -timepoint bin populations, labeled
  by most recently visited macrostate. The last state entry (*nstates-1*)
  corresponds to trajectories initiated outside of a defined macrostate.

``/bin_labels`` [bin]
  *(String)* Text labels of bins.

```

When macrostate assignments are given, the following additional datasets are present:

```

``/trajlabels`` [iteration][segment][timepoint]
  *(Integer)* Per-segment and -timepoint trajectory labels, indicating the
  macrostate which each trajectory last visited.

``/state_labels`` [state]
  *(String)* Labels of states.

``/state_map`` [bin]
  *(Integer)* Mapping of bin index to the macrostate containing that bin.
  An entry will contain *nbins+1* if that bin does not fall into a
  macrostate.

```

Datasets indexed by state and bin contain one more entry than the number of valid states or bins. For N bins, axes indexed by bin are of size $N+1$, and entry N (0-based indexing) corresponds to a walker outside of the defined bin space (which will cause most mappers to raise an error). More importantly, for M states (including the case $M=0$ where no states are specified), axes indexed by state are of size $M+1$ and entry M refers to trajectories initiated in a region not corresponding to a defined macrostate.

Thus, `labeled_populations[:, :, :].sum(axis=1)[:, :-1]` gives overall per-bin populations, for all defined bins and `labeled_populations[:, :, :].sum(axis=2)[:, :-1]` gives overall per-trajectory-ensemble populations for all defined states.

6.19.6.4 Parallelization

This tool supports parallelized binning, including reading/calculating input data.

6.19.6.5 Command-line options

optional arguments:

```

-h, --help          show this help message and exit

```

general options:

```

-r RCFILE, --rcfile RCFILE
                        use RCFILE as the WEST run-time configuration file (default: west.
↳ cfg)
--quiet                emit only essential information

```

(continues on next page)

(continued from previous page)

```
--verbose      emit extra information
--debug        enable extra checks and emit copious information
--version      show program's version number and exit
```

parallelization options:

```
--max-queue-length MAX_QUEUE_LENGTH
                        Maximum number of tasks that can be queued. Useful to limit RAM.
↳ use for tasks        that have very large requests/response. Default: no limit.
```

WEST input data options:

```
-W WEST_H5FILE, --west-data WEST_H5FILE
                        Take WEST data from WEST_H5FILE (default: read from the HDF5 file.
↳ specified in          west.cfg).
```

binning options:

- bins-from-system** Bins are constructed by the system driver specified in the WEST configuration file (default where stored bin definitions not available).
- bins-from-expr BINS_FROM_EXPR, --binbounds BINS_FROM_EXPR** Construct bins on a rectilinear grid according to the given BINEXPR. This must be a list of lists of bin boundaries (one list of bin boundaries for each dimension of the progress coordinate), formatted as a Python expression. E.g. "[[0,1,2,4,inf],[-inf,0,inf]]". The numpy module and the special symbol "inf" (for floating-point infinity) are available for use within BINEXPR.
- bins-from-function BINS_FROM_FUNCTION, --binfunc BINS_FROM_FUNCTION** Supply an external function which, when called, returns a properly constructed bin mapper which will then be used for bin assignments. This should be formatted as "[PATH:]MODULE.FUNC", where the function FUNC in module MODULE will be used; the optional PATH will be prepended to the module search path when loading MODULE.
- bins-from-file BINFILE, --binfile BINFILE** Load bin specification from the YAML file BINFILE. This currently takes the form {"bins": {"type": "RectilinearBinMapper", "boundaries": [[boundset1], [boundset2], ...]}}; only rectilinear bin bounds are supported.
- bins-from-h5file** Load bin specification from the data file being examined (default where stored bin definitions available).

input dataset options:

```
--construct-dataset CONSTRUCT_DATASET
                        Use the given function (as in module.function) to extract source.
↳ data. This           function will be called once per iteration as function(n_iter,
↳ iter_group) to        construct data for one iteration. Data returned must be indexable.
↳ as                    [seg_id][timepoint][dimension]
```

(continues on next page)

(continued from previous page)

```
--dsspecs DSSPEC [DSSPEC ...]
        Construct source data from one or more DSSPECs.
```

macrostate definitions:

```
--states STATEDEF [STATEDEF ...]
        Single-bin kinetic macrostate, specified by a coordinate tuple (e.
    ↪ g. '1.0' or
        '[1.0,1.0]'), optionally labeled (e.g. 'bound:[1.0,1.0]'). States
    ↪ corresponding
        to multiple bins must be specified with --states-from-file.
--states-from-file STATEFILE
        Load kinetic macrostates from the YAML file STATEFILE. See
    ↪ description above
        for the appropriate structure.
--states-from-function STATEFUNC
        Load kinetic macrostates from the function STATEFUNC, specified as
        module_name.func_name. This function is called with the bin mapper
    ↪ as an
        argument, and must return a list of dictionaries {'label': state_
    ↪ label,
        'coords': 2d_array_like} one for each macrostate; the 'coords'
    ↪ entry must
        contain enough rows to identify all bins in the macrostate.
```

other options:

```
-o OUTPUT, --output OUTPUT
        Store results in OUTPUT (default: assign.h5).
--subsample
        Determines whether or not the data should be subsampled. This is
    ↪ rather useful
        for analysing steady state simulations.
--config-from-file
        Load bins/macrostates from a scheme specified in west.cfg.
--scheme-name SCHEME
        Name of scheme specified in west.cfg.
```

parallelization options:

```
--serial
        run in serial mode
--parallel
        run in parallel mode (using processes)
--work-manager WORK_MANAGER
        use the given work manager for parallel task distribution.
    ↪ Available work
        managers are ('serial', 'threads', 'processes', 'zmq'); default is
    ↪ 'processes'
--n-workers N_WORKERS
        Use up to N_WORKERS on this host, for work managers which support
    ↪ this option.
        Use 0 for a dedicated server. (Ignored by work managers which do
    ↪ not support
        this option.)
```

options for ZeroMQ (“zmq”) work manager (master or node):

--zmq-mode MODE Operate as a master (server) or a node (workers/client). “server” is a dep-

recated synonym for “master” and “client” is a deprecated synonym for “node”.

- zmq-comm-mode COMM_MODE** Use the given communication mode – TCP or IPC (Unix-domain) – sockets for communication within a node. IPC (the default) may be more efficient but is not available on (exceptionally rare) systems without node-local storage (e.g. /tmp); on such systems, TCP may be used instead.
- zmq-write-host-info INFO_FILE** Store hostname and port information needed to connect to this instance in INFO_FILE. This allows the master and nodes assisting in coordinating the communication of other nodes to choose ports randomly. Downstream nodes read this file with `--zmq-read-host-info` and know where how to connect.
- zmq-read-host-info INFO_FILE** Read hostname and port information needed to connect to the master (or other coordinating node) from INFO_FILE. This allows the master and nodes assisting in coordinating the communication of other nodes to choose ports randomly, writing that information with `--zmq-write-host-info` for this instance to read.
- zmq-upstream-rr-endpoint ENDPOINT** ZeroMQ endpoint to which to send request/response (task and result) traffic toward the master.
- zmq-upstream-ann-endpoint ENDPOINT** ZeroMQ endpoint on which to receive announcement (heartbeat and shutdown notification) traffic from the master.
- zmq-downstream-rr-endpoint ENDPOINT** ZeroMQ endpoint on which to listen for request/response (task and result) traffic from subsidiary workers.
- zmq-downstream-ann-endpoint ENDPOINT** ZeroMQ endpoint on which to send announcement (heartbeat and shutdown notification) traffic toward workers.
- zmq-master-heartbeat MASTER_HEARTBEAT** Every MASTER_HEARTBEAT seconds, the master announces its presence to workers.
- zmq-worker-heartbeat WORKER_HEARTBEAT** Every WORKER_HEARTBEAT seconds, workers announce their presence to the master.
- zmq-timeout-factor FACTOR** Scaling factor for heartbeat timeouts. If the master doesn’t hear from a worker in `WORKER_HEARTBEAT*FACTOR`, the worker is assumed to have crashed. If a worker doesn’t hear from the master in `MASTER_HEARTBEAT*FACTOR` seconds, the master is assumed to have crashed. Both cases result in shutdown.
- zmq-startup-timeout STARTUP_TIMEOUT** Amount of time (in seconds) to wait for communication between the master and at least one worker. This may need to be changed on very large, heavily-loaded computer systems that start all processes simultaneously.
- zmq-shutdown-timeout SHUTDOWN_TIMEOUT** Amount of time (in seconds) to wait for workers to shut down.

6.19.7 w_trace

usage:

```
w_trace [-h] [-r RCFILE] [--quiet | --verbose | --debug] [--version] [-W WEST_H5FILE]
        [-d DSNAME] [--output-pattern OUTPUT_PATTERN] [-o OUTPUT]
        N_ITER:SEG_ID [N_ITER:SEG_ID ...]
```

Trace individual WEST trajectories and emit (or calculate) quantities along the trajectory.

Trajectories are specified as N_ITER:SEG_ID pairs. Each segment is traced back to its initial point, and then various quantities (notably n_iter and seg_id) are printed in order from initial point up until the given segment in the given iteration.

Output is stored in several files, all named according to the pattern given by the -o/--output-pattern parameter. The default output pattern is “traj_%d_%d”, where the printf-style format codes are replaced by the iteration number and segment ID of the terminal segment of the trajectory being traced.

Individual datasets can be selected for writing using the -d/--dataset option (which may be specified more than once). The simplest form is -d dsname, which causes data from dataset dsname along the trace to be stored to HDF5. The dataset is assumed to be stored on a per-iteration basis, with the first dimension corresponding to seg_id and the second dimension corresponding to time within the segment. Further options are specified as comma-separated key=value pairs after the data set name, as in:

```
-d dsname,alias=newname,index=idsname,file=otherfile.h5,slice=[100,...]
```

The following options for datasets are supported:

alias=newname

When writing this data to HDF5 or text files, use ``newname`` instead of ``dsname`` to identify the dataset. This is mostly of use in conjunction with the ``slice`` option in order, e.g., to retrieve two different slices of a dataset and store them with different names for future use.

index=idsname

The dataset is not stored on a per-iteration basis for all segments, but instead is stored as a single dataset whose first dimension indexes n_iter/seg_id pairs. The index to these n_iter/seg_id pairs is ``idsname``.

file=otherfile.h5

Instead of reading data from the main WEST HDF5 file (usually ``west.h5``), read data from ``otherfile.h5``.

slice=[100,...]

Retrieve only the given slice from the dataset. This can be used to pick a subset of interest to minimize I/O.

6.19.7.1 positional arguments

N_ITER:SEG_ID Trace trajectory ending (**or** at least alive at) N_ITER:SEG_ID.

6.19.7.2 optional arguments

```
-h, --help            show this help message and exit
-d DSNAME, --dataset DSNAME      Include the dataset named DSNAME in trace output. An extended form
↳like                              DSNAME[,alias=ALIAS][,index=INDEX][,file=FILE][,slice=SLICE] will
↳obtain the                        dataset from the given FILE instead of the main WEST HDF5 file,
↳slice it by                        SLICE, call it ALIAS in output, and/or access per-segment data by a
↳group for                         n_iter,seg_id INDEX instead of a seg_id indexed dataset in the
                                    n_iter.
```

6.19.7.3 general options

```
-r RCFILE, --rcfile RCFILE        use RCFILE as the WEST run-time configuration file (default: west.
↳cfg)
--quiet                emit only essential information
--verbose              emit extra information
--debug                enable extra checks and emit copious information
--version              show program's version number and exit
```

6.19.7.4 WEST input data options

```
-W WEST_H5FILE, --west-data WEST_H5FILE      Take WEST data from WEST_H5FILE (default: read from the HDF5 file,
↳specified in                        west.cfg).
```

6.19.7.5 output options

```
--output-pattern OUTPUT_PATTERN      Write per-trajectory data to output files/HDF5 groups whose names
↳begin with                        OUTPUT_PATTERN, which must contain two printf-style format flags
↳which will be                       replaced with the iteration number and segment ID of the terminal
↳segment of                         the trajectory being traced. (Default: traj_%d_%d.)
-o OUTPUT, --output OUTPUT
```

(continues on next page)

(continued from previous page)

```
Store intermediate data and analysis results to OUTPUT (default:
↳ trajns.h5).
```

6.19.8 w_fluxanl

w_fluxanl calculates the probability flux of a weighted ensemble simulation based on a pre-defined target state. Also calculates confidence interval of average flux. Monte Carlo bootstrapping techniques are used to account for autocorrelation between fluxes and/or errors that are not normally distributed.

6.19.8.1 Overview

usage:

```
$WEST_ROOT/bin/w_fluxanl [-h] [-r RCFILE] [--quiet | --verbose | --debug] [--version]
                        [-W WEST_H5FILE] [-o OUTPUT]
                        [--first-iter N_ITER] [--last-iter N_ITER]
                        [-a ALPHA] [--autocorrel-alpha ACALPHA] [-N NSETS] [--evol] [--
↳ evol-step ESTEP]
```

Note: All command line arguments are optional for w_fluxanl.

6.19.8.2 Command-Line Options

See the [general command-line tool reference](#) for more information on the general options.

6.19.8.2.1 Input/output options

These arguments allow the user to specify where to read input simulation result data and where to output calculated progress coordinate probability distribution data.

Both input and output files are *hdf5* format.:

```
-W, --west-data file
  Read simulation result data from file *file*. (**Default:** The
  *hdf5* file specified in the configuration file)

-o, --output file
  Store this tool's output in *file*. (**Default:** The *hdf5* file
  **pcpdist.h5**)
```

6.19.8.2.2 Iteration range options

Specify the range of iterations over which to construct the progress coordinate probability distribution.:

```
--first-iter n_iter
Construct probability distribution starting with iteration *n_iter*
(**Default:** 1)

--last-iter n_iter
Construct probability distribution's time evolution up to (and
including) iteration *n_iter* (**Default:** Last completed
iteration)
```

6.19.8.2.3 Confidence interval and bootstrapping options

Specify alpha values of constructed confidence intervals.:

```
-a alpha
Calculate a (1 - *alpha*) confidence interval for the mean flux
(**Default:** 0.05)

--autocorrel-alpha ACalpha
Identify autocorrelation of fluxes at *ACalpha* significance level.
Note: Specifying an *ACalpha* level that is too small may result in
failure to find autocorrelation in noisy flux signals (**Default:**
Same level as *alpha*)

-N n_sets, --nsets n_sets
Use *n_sets* samples for bootstrapping (**Default:** Chosen based
on *alpha*)

--evol
Calculate the time evolution of flux confidence intervals
(**Warning:** computationally expensive calculation)

--evol-step estep
(if ``--evol`` specified) Calculate the time evolution of flux
confidence intervals for every *estep* iterations (**Default:** 1)
```

6.19.8.3 Examples

Calculate the time evolution flux every 5 iterations:

```
$WEST_ROOT/bin/w_fluxan1 --evol --evol-step 5
```

Calculate mean flux confidence intervals at 0.01 significance level and calculate autocorrelations at 0.05 significance:

```
$WEST_ROOT/bin/w_fluxan1 --alpha 0.01 --autocorrel-alpha 0.05
```

Calculate the mean flux confidence intervals using a custom bootstrap sample size of 500:

```
$WEST_ROOT/bin/w_fluxanl --n-sets 500
```

6.19.9 w_ipa

usage:

```
w_ipa [-h] [-r RCFILE] [--quiet] [--verbose] [--version] [--max-queue-length MAX_QUEUE_
↳LENGTH]
        [-W WEST_H5FILE] [--analysis-only] [--reanalyze] [--ignore-hash] [--debug] [-
↳-terminal]
        [--serial | --parallel | --work-manager WORK_MANAGER] [--n-workers N_WORKERS]
        [--zmq-mode MODE] [--zmq-comm-mode COMM_MODE] [--zmq-write-host-info INFO_
↳FILE]
        [--zmq-read-host-info INFO_FILE] [--zmq-upstream-rr-endpoint ENDPOINT]
        [--zmq-upstream-ann-endpoint ENDPOINT] [--zmq-downstream-rr-endpoint_
↳ENDPOINT]
        [--zmq-downstream-ann-endpoint ENDPOINT] [--zmq-master-heartbeat MASTER_
↳HEARTBEAT]
        [--zmq-worker-heartbeat WORKER_HEARTBEAT] [--zmq-timeout-factor FACTOR]
        [--zmq-startup-timeout STARTUP_TIMEOUT] [--zmq-shutdown-timeout SHUTDOWN_
↳TIMEOUT]
```

optional arguments:

```
-h, --help          show this help message and exit
```

general options:

-r RCFILE, --rcfile RCFILE use RCFILE as the WEST run-time configuration file (default: west.cfg)

--quiet emit only essential information

--verbose emit extra information

--version show program's version number and exit

parallelization options:

```
--max-queue-length MAX_QUEUE_LENGTH
        Maximum number of tasks that can be queued. Useful to limit RAM_
↳use for tasks that
        have very large requests/response. Default: no limit.
```

WEST input data options:

-W WEST_H5FILE, --west-data WEST_H5FILE Take WEST data from WEST_H5FILE (default: read from the HDF5 file specified in west.cfg).

runtime options:

```
--analysis-only, -ao Use this flag to run the analysis and return to the terminal.
--reanalyze, -ra      Use this flag to delete the existing files and reanalyze.
--ignore-hash, -ih    Ignore hash and don't regenerate files.
--debug, -d           Debug output largely intended for development.
--terminal, -t        Plot output in terminal.
```

parallelization options:

```
--serial          run in serial mode
--parallel        run in parallel mode (using processes)
--work-manager WORK_MANAGER
                  use the given work manager for parallel task distribution.
↳ Available work managers
                  are ('serial', 'threads', 'processes', 'zmq'); default is
↳ 'processes'
--n-workers N_WORKERS
                  Use up to N_WORKERS on this host, for work managers which support
↳ this option. Use
                  0 for a dedicated server. (Ignored by work managers which do not
↳ support this
                  option.)
```

options for ZeroMQ (“zmq”) work manager (master or node):

```
--zmq-mode MODE    Operate as a master (server) or a node (workers/client). "server"
↳ is a deprecated
                  synonym for "master" and "client" is a deprecated synonym for "node
↳ ".
--zmq-comm-mode COMM_MODE
                  Use the given communication mode -- TCP or IPC (Unix-domain) --
↳ sockets for
                  communication within a node. IPC (the default) may be more
↳ efficient but is not
                  available on (exceptionally rare) systems without node-local
↳ storage (e.g. /tmp);
                  on such systems, TCP may be used instead.
--zmq-write-host-info INFO_FILE
                  Store hostname and port information needed to connect to this
↳ instance in
                  INFO_FILE. This allows the master and nodes assisting in
↳ coordinating the
                  communication of other nodes to choose ports randomly. Downstream
↳ nodes read this
                  file with --zmq-read-host-info and know where how to connect.
--zmq-read-host-info INFO_FILE
                  Read hostname and port information needed to connect to the master
↳ (or other
                  coordinating node) from INFO_FILE. This allows the master and
↳ nodes assisting in
                  coordinating the communication of other nodes to choose ports
↳ randomly, writing
                  that information with --zmq-write-host-info for this instance to
↳ read.
--zmq-upstream-rr-endpoint ENDPOINT
                  ZeroMQ endpoint to which to send request/response (task and
↳ result) traffic toward
                  the master.
--zmq-upstream-ann-endpoint ENDPOINT
                  ZeroMQ endpoint on which to receive announcement (heartbeat and
↳ shutdown
```

(continues on next page)

(continued from previous page)

```

notification) traffic from the master.
--zmq-downstream-rr-endpoint ENDPOINT
    ZeroMQ endpoint on which to listen for request/response (task and
↳ result) traffic
    from subsidiary workers.
--zmq-downstream-ann-endpoint ENDPOINT
    ZeroMQ endpoint on which to send announcement (heartbeat and
↳ shutdown
    notification) traffic toward workers.
--zmq-master-heartbeat MASTER_HEARTBEAT
    Every MASTER_HEARTBEAT seconds, the master announces its presence
↳ to workers.
--zmq-worker-heartbeat WORKER_HEARTBEAT
    Every WORKER_HEARTBEAT seconds, workers announce their presence to
↳ the master.
--zmq-timeout-factor FACTOR
    Scaling factor for heartbeat timeouts. If the master doesn't hear
↳ from a worker in
    WORKER_HEARTBEAT*FACTOR, the worker is assumed to have crashed. If
↳ a worker
    doesn't hear from the master in MASTER_HEARTBEAT*FACTOR seconds,
↳ the master is
    assumed to have crashed. Both cases result in shutdown.
--zmq-startup-timeout STARTUP_TIMEOUT
    Amount of time (in seconds) to wait for communication between the
↳ master and at
    least one worker. This may need to be changed on very large,
↳ heavily-loaded
    computer systems that start all processes simultaneously.
--zmq-shutdown-timeout SHUTDOWN_TIMEOUT
    Amount of time (in seconds) to wait for workers to shut down.

```

6.19.10 w_pdist

usage:

```

w_pdist [-h] [-r RCFILE] [--quiet | --verbose | --debug] [--version]
    [--max-queue-length MAX_QUEUE_LENGTH] [-W WEST_H5FILE] [--first-iter N_
↳ ITER]
    [--last-iter N_ITER] [-b BINEXPR] [-o OUTPUT] [-C] [--loose]
    [--construct-dataset CONSTRUCT_DATASET | --dsspecs DSSPEC [DSSPEC ...]]
    [--serial | --parallel | --work-manager WORK_MANAGER] [--n-workers N_
↳ WORKERS]
    [--zmq-mode MODE] [--zmq-comm-mode COMM_MODE] [--zmq-write-host-info INFO_
↳ FILE]
    [--zmq-read-host-info INFO_FILE] [--zmq-upstream-rr-endpoint ENDPOINT]
    [--zmq-upstream-ann-endpoint ENDPOINT] [--zmq-downstream-rr-endpoint
↳ ENDPOINT]
    [--zmq-downstream-ann-endpoint ENDPOINT] [--zmq-master-heartbeat MASTER_
↳ HEARTBEAT]
    [--zmq-worker-heartbeat WORKER_HEARTBEAT] [--zmq-timeout-factor FACTOR]

```

(continues on next page)

(continued from previous page)

```
[--zmq-startup-timeout STARTUP_TIMEOUT] [--zmq-shutdown-timeout SHUTDOWN_
→ TIMEOUT]
```

Calculate time-resolved, multi-dimensional probability distributions of WE datasets.

6.19.10.1 Source data

Source data is provided either by a user-specified function (`-construct-dataset`) or a list of “data set specifications” (`-dsspecs`). If neither is provided, the progress coordinate dataset “pcoord” is used.

To use a custom function to extract or calculate data whose probability distribution will be calculated, specify the function in standard Python `MODULE.FUNCTION` syntax as the argument to `-construct-dataset`. This function will be called as `function(n_iter,iter_group)`, where `n_iter` is the iteration whose data are being considered and `iter_group` is the corresponding group in the main WEST HDF5 file (`west.h5`). The function must return data which can be indexed as `[segment][timepoint][dimension]`.

To use a list of data set specifications, specify `-dsspecs` and then list the desired datasets one-by-one (space-separated in most shells). These data set specifications are formatted as `NAME[,file=FILENAME,slice=SLICE]`, which will use the dataset called `NAME` in the HDF5 file `FILENAME` (defaulting to the main WEST HDF5 file `west.h5`), and slice it with the Python slice expression `SLICE` (as in `[0:2]` to select the first two elements of the first axis of the dataset). The slice option is most useful for selecting one column (or more) from a multi-column dataset, such as arises when using a progress coordinate of multiple dimensions.

6.19.10.2 Histogram binning

By default, histograms are constructed with 100 bins in each dimension. This can be overridden by specifying `-b/-bins`, which accepts a number of different kinds of arguments:

```
a single integer N
  N uniformly spaced bins will be used in each dimension.

a sequence of integers N1,N2,... (comma-separated)
  N1 uniformly spaced bins will be used for the first dimension, N2 for the
  second, and so on.

a list of lists [[B11, B12, B13, ...], [B21, B22, B23, ...], ...]
  The bin boundaries B11, B12, B13, ... will be used for the first dimension,
  B21, B22, B23, ... for the second dimension, and so on. These bin
  boundaries need not be uniformly spaced. These expressions will be
  evaluated with Python's `eval` construct, with `np` available for
  use [e.g. to specify bins using np.arange()].
```

The first two forms (integer, list of integers) will trigger a scan of all data in each dimension in order to determine the minimum and maximum values, which may be very expensive for large datasets. This can be avoided by explicitly providing bin boundaries using the list-of-lists form.

Note that these bins are *NOT* at all related to the bins used to drive WE sampling.

6.19.10.3 Output format

The output file produced (specified by `-o/--output`, defaulting to “`pdist.h5`”) may be fed to `plothist` to generate plots (or appropriately processed text or HDF5 files) from this data. In short, the following datasets are created:

```
``histograms``
  Normalized histograms. The first axis corresponds to iteration, and
  remaining axes correspond to dimensions of the input dataset.

``/binbounds_0``
  Vector of bin boundaries for the first (index 0) dimension. Additional
  datasets similarly named (/binbounds_1, /binbounds_2, ...) are created
  for additional dimensions.

``/midpoints_0``
  Vector of bin midpoints for the first (index 0) dimension. Additional
  datasets similarly named are created for additional dimensions.

``n_iter``
  Vector of iteration numbers corresponding to the stored histograms (i.e.
  the first axis of the ``histograms`` dataset).
```

6.19.10.4 Subsequent processing

The output generated by this program (`-o/--output`, default “`pdist.h5`”) may be plotted by the `plothist` program. See `plothist --help` for more information.

6.19.10.5 Parallelization

This tool supports parallelized binning, including reading of input data. Parallel processing is the default. For simple cases (reading pre-computed input data, modest numbers of segments), serial processing (`--serial`) may be more efficient.

6.19.10.6 Command-line options

optional arguments:

```
-h, --help          show this help message and exit
-b BINEXPR, --bins BINEXPR
                    Use BINEXPR for bins. This may be an integer, which will be used
↳ for each         dimension of the progress coordinate; a list of integers
↳ (formatted as     [n1,n2,...]) which will use n1 bins for the first dimension, n2
↳ for the second    dimension, and so on; or a list of lists of boundaries (formatted
↳ as [[a1, a2,      ..., [b1, b2, ..., ... ]]), which will use [a1, a2, ...] as bin
↳ boundaries for    the first dimension, [b1, b2, ...] as bin boundaries for the
↳ second dimension, and so on. (Default: 100 bins in each dimension.)
```

(continues on next page)

(continued from previous page)

```

-o OUTPUT, --output OUTPUT      Store results in OUTPUT (default: pdist.h5).
-C, --compress                  Compress histograms. May make storage of higher-dimensional
↪ histograms more                tractable, at the (possible extreme) expense of increased analysis.
↪ time.                          (Default: no compression.)
--loose                          Ignore values that do not fall within bins. (Risky, as this can
↪ make buggy bin                boundaries appear as reasonable data. Only use if you are sure of
↪ your bin                      boundary specification.)

```

general options:

```

-r RCFILE, --rcfile RCFILE      use RCFILE as the WEST run-time configuration file (default: west.
↪ cfg)
--quiet                          emit only essential information
--verbose                        emit extra information
--debug                          enable extra checks and emit copious information
--version                        show program's version number and exit

```

parallelization options:

```

--max-queue-length MAX_QUEUE_LENGTH
↪ use for tasks                    Maximum number of tasks that can be queued. Useful to limit RAM
                                that have very large requests/response. Default: no limit.

```

WEST input data options:

-W WEST_H5FILE, --west-data WEST_H5FILE Take WEST data from WEST_H5FILE
(default: read from the HDF5 file specified in west.cfg).

iteration range:

```

--first-iter N_ITER             Begin analysis at iteration N_ITER (default: 1).
--last-iter N_ITER             Conclude analysis with N_ITER, inclusive (default: last completed
↪ iteration).

```

input dataset options:

```

--construct-dataset CONSTRUCT_DATASET
↪ data. This                    Use the given function (as in module.function) to extract source
↪ iter_group) to                function will be called once per iteration as function(n_iter,
↪ as                            construct data for one iteration. Data returned must be indexable.
                                [seg_id][timepoint][dimension]
--dsspecs DSSPEC [DSSPEC ...]   Construct probability distribution from one or more DSSPECs.

```

parallelization options:

```

--serial          run in serial mode
--parallel        run in parallel mode (using processes)
--work-manager WORK_MANAGER
                  use the given work manager for parallel task distribution.
↪ Available work  managers are ('serial', 'threads', 'processes', 'zmq'); default is
↪ 'processes'
--n-workers N_WORKERS
                  Use up to N_WORKERS on this host, for work managers which support
↪ this option.
                  Use 0 for a dedicated server. (Ignored by work managers which do
↪ not support    this option.)

```

options for ZeroMQ (“zmq”) work manager (master or node):

- zmq-mode MODE** Operate as a master (server) or a node (workers/client). “server” is a deprecated synonym for “master” and “client” is a deprecated synonym for “node”.
- zmq-comm-mode COMM_MODE** Use the given communication mode – TCP or IPC (Unix-domain) – sockets for communication within a node. IPC (the default) may be more efficient but is not available on (exceptionally rare) systems without node-local storage (e.g. /tmp); on such systems, TCP may be used instead.
- zmq-write-host-info INFO_FILE** Store hostname and port information needed to connect to this instance in INFO_FILE. This allows the master and nodes assisting in coordinating the communication of other nodes to choose ports randomly. Downstream nodes read this file with `--zmq-read-host-info` and know where how to connect.
- zmq-read-host-info INFO_FILE** Read hostname and port information needed to connect to the master (or other coordinating node) from INFO_FILE. This allows the master and nodes assisting in coordinating the communication of other nodes to choose ports randomly, writing that information with `--zmq-write-host-info` for this instance to read.
- zmq-upstream-rr-endpoint ENDPOINT** ZeroMQ endpoint to which to send request/response (task and result) traffic toward the master.
- zmq-upstream-ann-endpoint ENDPOINT** ZeroMQ endpoint on which to receive announcement (heartbeat and shutdown notification) traffic from the master.
- zmq-downstream-rr-endpoint ENDPOINT** ZeroMQ endpoint on which to listen for request/response (task and result) traffic from subsidiary workers.
- zmq-downstream-ann-endpoint ENDPOINT** ZeroMQ endpoint on which to send announcement (heartbeat and shutdown notification) traffic toward workers.
- zmq-master-heartbeat MASTER_HEARTBEAT** Every MASTER_HEARTBEAT seconds, the master announces its presence to workers.
- zmq-worker-heartbeat WORKER_HEARTBEAT** Every WORKER_HEARTBEAT seconds, workers announce their presence to the master.
- zmq-timeout-factor FACTOR** Scaling factor for heartbeat timeouts. If the master doesn’t hear from a worker in `WORKER_HEARTBEAT*FACTOR`, the worker is assumed to have crashed. If a worker doesn’t hear from the master

in `MASTER_HEARTBEAT*FACTOR` seconds, the master is assumed to have crashed. Both cases result in shutdown.

--zmq-startup-timeout STARTUP_TIMEOUT Amount of time (in seconds) to wait for communication between the master and at least one worker. This may need to be changed on very large, heavily-loaded computer systems that start all processes simultaneously.

--zmq-shutdown-timeout SHUTDOWN_TIMEOUT Amount of time (in seconds) to wait for workers to shut down.

6.19.11 w_succ

usage:

```
w_succ [-h] [-r RCFILE] [--quiet | --verbose | --debug] [--version] [-A H5FILE] [-W WEST_
↪H5FILE]
        [-o OUTPUT_FILE]
```

List segments which successfully reach a target state.

optional arguments:

```
-h, --help            show this help message and exit
-o OUTPUT_FILE, --output OUTPUT_FILE
                        Store output in OUTPUT_FILE (default: write to standard output).
```

general options:

```
-r RCFILE, --rcfile RCFILE
                        use RCFILE as the WEST run-time configuration file (default: west.
↪cfg)
--quiet               emit only essential information
--verbose             emit extra information
--debug              enable extra checks and emit copious information
--version             show program's version number and exit
```

general analysis options:

```
-A H5FILE, --analysis-file H5FILE
                        Store intermediate and final results in H5FILE (default: analysis.
↪h5).
```

WEST input data options:

```
-W WEST_H5FILE, --west-data WEST_H5FILE
                        Take WEST data from WEST_H5FILE (default: read from the HDF5 file,
↪specified in west.cfg).
```

6.19.12 w_crawl

usage:

```
w_crawl [-h] [-r RCFILE] [--quiet | --verbose | --debug] [--version]
        [--max-queue-length MAX_QUEUE_LENGTH] [-W WEST_H5FILE] [--first-iter N_
↪ITER]
        [--last-iter N_ITER] [-c CRAWLER_INSTANCE]
        [--serial | --parallel | --work-manager WORK_MANAGER] [--n-workers N_
↪WORKERS]
        [--zmq-mode MODE] [--zmq-comm-mode COMM_MODE] [--zmq-write-host-info INFO_
↪FILE]
        [--zmq-read-host-info INFO_FILE] [--zmq-upstream-rr-endpoint ENDPOINT]
        [--zmq-upstream-ann-endpoint ENDPOINT] [--zmq-downstream-rr-endpoint_
↪ENDPOINT]
        [--zmq-downstream-ann-endpoint ENDPOINT] [--zmq-master-heartbeat MASTER_
↪HEARTBEAT]
        [--zmq-worker-heartbeat WORKER_HEARTBEAT] [--zmq-timeout-factor FACTOR]
        [--zmq-startup-timeout STARTUP_TIMEOUT] [--zmq-shutdown-timeout SHUTDOWN_
↪TIMEOUT]
        task_callable
```

Crawl a weighted ensemble dataset, executing a function for each iteration. This can be used for postprocessing of trajectories, cleanup of datasets, or anything else that can be expressed as “do X for iteration N, then do something with the result”. Tasks are parallelized by iteration, and no guarantees are made about evaluation order.

6.19.12.1 Command-line options

optional arguments:

```
-h, --help          show this help message and exit
```

general options:

```
-r RCFILE, --rcfile RCFILE
        use RCFILE as the WEST run-time configuration file (default: west.
↪cfg)
--quiet          emit only essential information
--verbose        emit extra information
--debug          enable extra checks and emit copious information
--version        show program's version number and exit
```

parallelization options:

```
--max-queue-length MAX_QUEUE_LENGTH
        Maximum number of tasks that can be queued. Useful to limit RAM_
↪use for tasks
        that have very large requests/response. Default: no limit.
```

WEST input data options:

```
-W WEST_H5FILE, --west-data WEST_H5FILE
        Take WEST data from WEST_H5FILE (default: read from the HDF5 file_
```

(continues on next page)

(continued from previous page)

```
↪specified in
west.cfg).
```

iteration range:

```
--first-iter N_ITER   Begin analysis at iteration N_ITER (default: 1).
--last-iter N_ITER    Conclude analysis with N_ITER, inclusive (default: last completed_
↪iteration).
```

task options:

```
-c CRAWLER_INSTANCE, --crawler-instance CRAWLER_INSTANCE
↪of                               Use CRAWLER_INSTANCE (specified as module.instance) as an instance_
↪initialization,                 WESTPACrawler to coordinate the calculation. Required only if_
task_callable                 finalization, or task result processing is required.
↪ Required.                     Run TASK_CALLABLE (specified as module.function) on each iteration.
```

parallelization options:

```
--serial               run in serial mode
--parallel             run in parallel mode (using processes)
--work-manager WORK_MANAGER
↪Available work        use the given work manager for parallel task distribution_
↪'serial'              managers are ('serial', 'threads', 'processes', 'zmq'); default is
--n-workers N_WORKERS  Use up to N_WORKERS on this host, for work managers which support_
↪this option.          Use 0 for a dedicated server. (Ignored by work managers which do_
↪not support           this option.)
```

options for ZeroMQ (“zmq”) work manager (master or node):

```
--zmq-mode MODE        Operate as a master (server) or a node (workers/client). "server"_
↪is a                  deprecated synonym for "master" and "client" is a deprecated_
↪synonym for           "node".
--zmq-comm-mode COMM_MODE
↪sockets for           Use the given communication mode -- TCP or IPC (Unix-domain) --_
↪efficient but is not   communication within a node. IPC (the default) may be more_
↪storage (e.g.          available on (exceptionally rare) systems without node-local_
                        /tmp); on such systems, TCP may be used instead.
--zmq-write-host-info INFO_FILE
                        Store hostname and port information needed to connect to this_
```

(continues on next page)

(continued from previous page)

```

↪instance in
INFO_FILE. This allows the master and nodes assisting in
↪coordinating the
communication of other nodes to choose ports randomly. Downstream
↪nodes read
this file with --zmq-read-host-info and know where how to connect.
--zmq-read-host-info INFO_FILE
Read hostname and port information needed to connect to the master
↪(or other
coordinating node) from INFO_FILE. This allows the master and
↪nodes assisting
in coordinating the communication of other nodes to choose ports
↪randomly,
writing that information with --zmq-write-host-info for this
↪instance to read.
--zmq-upstream-rr-endpoint ENDPOINT
ZeroMQ endpoint to which to send request/response (task and
↪result) traffic
toward the master.
--zmq-upstream-ann-endpoint ENDPOINT
ZeroMQ endpoint on which to receive announcement (heartbeat and
↪shutdown
notification) traffic from the master.
--zmq-downstream-rr-endpoint ENDPOINT
ZeroMQ endpoint on which to listen for request/response (task and
↪result)
traffic from subsidiary workers.
--zmq-downstream-ann-endpoint ENDPOINT
ZeroMQ endpoint on which to send announcement (heartbeat and
↪shutdown
notification) traffic toward workers.
--zmq-master-heartbeat MASTER_HEARTBEAT
Every MASTER_HEARTBEAT seconds, the master announces its presence
↪to workers.
--zmq-worker-heartbeat WORKER_HEARTBEAT
Every WORKER_HEARTBEAT seconds, workers announce their presence to
↪the master.
--zmq-timeout-factor FACTOR
Scaling factor for heartbeat timeouts. If the master doesn't hear
↪from a worker
in WORKER_HEARTBEAT*FACTOR, the worker is assumed to have crashed.
↪If a worker
doesn't hear from the master in MASTER_HEARTBEAT*FACTOR seconds,
↪the master is
assumed to have crashed. Both cases result in shutdown.
--zmq-startup-timeout STARTUP_TIMEOUT
Amount of time (in seconds) to wait for communication between the
↪master and at
least one worker. This may need to be changed on very large,
↪heavily-loaded
computer systems that start all processes simultaneously.
--zmq-shutdown-timeout SHUTDOWN_TIMEOUT

```

(continues on next page)

(continued from previous page)

Amount of time (**in** seconds) to wait **for** workers to shut down.

6.19.13 w_direct

usage:

```
w_direct [-h] [-r RCFILE] [--quiet | --verbose | --debug] [--version]
          [--max-queue-length MAX_QUEUE_LENGTH]
          [--serial | --parallel | --work-manager WORK_MANAGER] [--n-workers N_
↪WORKERS]
          [--zmq-mode MODE] [--zmq-comm-mode COMM_MODE] [--zmq-write-host-info INFO_
↪FILE]
          [--zmq-read-host-info INFO_FILE] [--zmq-upstream-rr-endpoint ENDPOINT]
          [--zmq-upstream-ann-endpoint ENDPOINT] [--zmq-downstream-rr-endpoint_
↪ENDPOINT]
          [--zmq-downstream-ann-endpoint ENDPOINT] [--zmq-master-heartbeat MASTER_
↪HEARTBEAT]
          [--zmq-worker-heartbeat WORKER_HEARTBEAT] [--zmq-timeout-factor FACTOR]
          [--zmq-startup-timeout STARTUP_TIMEOUT] [--zmq-shutdown-timeout SHUTDOWN_
↪TIMEOUT]
          {help,init,average,kinetics,probs,all} ...
```

optional arguments:

-h, --help show this help message **and** exit

general options:

```
-r RCFILE, --rcfile RCFILE
                               use RCFILE as the WEST run-time configuration file (default: west.
↪cfg)
--quiet                       emit only essential information
--verbose                     emit extra information
--debug                       enable extra checks and emit copious information
--version                     show program's version number and exit
```

parallelization options:

```
--max-queue-length MAX_QUEUE_LENGTH
                               Maximum number of tasks that can be queued. Useful to limit RAM_
↪use for tasks that
                               have very large requests/response. Default: no limit.
```

direct kinetics analysis schemes:

```
{help,init,average,kinetics,probs,all}
  help      print help for this command or individual subcommands
  init      calculate state-to-state kinetics by tracing trajectories
  average   Averages and returns fluxes, rates, and color/state populations.
  kinetics  Generates rate and flux values from a WESTPA simulation via_
↪tracing.
```

(continues on next page)

(continued from previous page)

<code>probs</code>	Calculates color and state probabilities via tracing.
<code>all</code>	Runs the full suite, including the tracing of events.

parallelization options:

```
--serial          run in serial mode
--parallel        run in parallel mode (using processes)
--work-manager WORK_MANAGER
                  use the given work manager for parallel task distribution.
↳ Available work managers
                  are ('serial', 'threads', 'processes', 'zmq'); default is 'serial'
--n-workers N_WORKERS
                  Use up to N_WORKERS on this host, for work managers which support
↳ this option. Use
                  0 for a dedicated server. (Ignored by work managers which do not
↳ support this
                  option.)
```

options for ZeroMQ (“zmq”) work manager (master or node):

```
--zmq-mode MODE    Operate as a master (server) or a node (workers/client). "server"
↳ is a deprecated
                  synonym for "master" and "client" is a deprecated synonym for "node"
↳ ".
--zmq-comm-mode COMM_MODE
                  Use the given communication mode -- TCP or IPC (Unix-domain) --
↳ sockets for
                  communication within a node. IPC (the default) may be more
↳ efficient but is not
                  available on (exceptionally rare) systems without node-local
↳ storage (e.g. /tmp);
                  on such systems, TCP may be used instead.
--zmq-write-host-info INFO_FILE
                  Store hostname and port information needed to connect to this
↳ instance in
                  INFO_FILE. This allows the master and nodes assisting in
↳ coordinating the
                  communication of other nodes to choose ports randomly. Downstream
↳ nodes read this
                  file with --zmq-read-host-info and know where how to connect.
--zmq-read-host-info INFO_FILE
                  Read hostname and port information needed to connect to the master
↳ (or other
                  coordinating node) from INFO_FILE. This allows the master and
↳ nodes assisting in
                  coordinating the communication of other nodes to choose ports
↳ randomly, writing
                  that information with --zmq-write-host-info for this instance to
↳ read.
--zmq-upstream-rr-endpoint ENDPOINT
                  ZeroMQ endpoint to which to send request/response (task and
↳ result) traffic toward
```

(continues on next page)

(continued from previous page)

```

the master.
--zmq-upstream-ann-endpoint ENDPOINT
    ZeroMQ endpoint on which to receive announcement (heartbeat and
↳ shutdown
        notification) traffic from the master.
--zmq-downstream-rr-endpoint ENDPOINT
    ZeroMQ endpoint on which to listen for request/response (task and
↳ result) traffic
        from subsidiary workers.
--zmq-downstream-ann-endpoint ENDPOINT
    ZeroMQ endpoint on which to send announcement (heartbeat and
↳ shutdown
        notification) traffic toward workers.
--zmq-master-heartbeat MASTER_HEARTBEAT
    Every MASTER_HEARTBEAT seconds, the master announces its presence
↳ to workers.
--zmq-worker-heartbeat WORKER_HEARTBEAT
    Every WORKER_HEARTBEAT seconds, workers announce their presence to
↳ the master.
--zmq-timeout-factor FACTOR
    Scaling factor for heartbeat timeouts. If the master doesn't hear
↳ from a worker in
        WORKER_HEARTBEAT*FACTOR, the worker is assumed to have crashed. If
↳ a worker
        doesn't hear from the master in MASTER_HEARTBEAT*FACTOR seconds,
↳ the master is
        assumed to have crashed. Both cases result in shutdown.
--zmq-startup-timeout STARTUP_TIMEOUT
    Amount of time (in seconds) to wait for communication between the
↳ master and at
        least one worker. This may need to be changed on very large,
↳ heavily-loaded
        computer systems that start all processes simultaneously.
--zmq-shutdown-timeout SHUTDOWN_TIMEOUT
    Amount of time (in seconds) to wait for workers to shut down.

```

6.19.14 w_select

usage:

```

w_select [-h] [-r RCFILE] [--quiet | --verbose | --debug] [--version]
↳ ITER]
    [--max-queue-length MAX_QUEUE_LENGTH] [-W WEST_H5FILE] [--first-iter N_
    [--last-iter N_ITER] [-p MODULE.FUNCTION] [-v] [-a] [-o OUTPUT]
    [--serial | --parallel | --work-manager WORK_MANAGER] [--n-workers N_
↳ WORKERS]
    [--zmq-mode MODE] [--zmq-comm-mode COMM_MODE] [--zmq-write-host-info INFO_
↳ FILE]
    [--zmq-read-host-info INFO_FILE] [--zmq-upstream-rr-endpoint ENDPOINT]
    [--zmq-upstream-ann-endpoint ENDPOINT] [--zmq-downstream-rr-endpoint
↳ ENDPOINT]

```

(continues on next page)

(continued from previous page)

```

[--zmq-downstream-ann-endpoint ENDPOINT] [--zmq-master-heartbeat MASTER_
↪HEARTBEAT]
[--zmq-worker-heartbeat WORKER_HEARTBEAT] [--zmq-timeout-factor FACTOR]
[--zmq-startup-timeout STARTUP_TIMEOUT] [--zmq-shutdown-timeout SHUTDOWN_
↪TIMEOUT]

```

Select dynamics segments matching various criteria. This requires a user-provided predicate function. By default, only matching segments are stored. If the `-a/--include-ancestors` option is given, then matching segments and their ancestors will be stored.

6.19.14.1 Predicate function

Segments are selected based on a predicate function, which must be callable as `predicate(n_iter, iter_group)` and return a collection of segment IDs matching the predicate in that iteration.

The predicate may be inverted by specifying the `-v/--invert` command-line argument.

6.19.14.2 Output format

The output file (`-o/--output`, by default “select.h5”) contains the following datasets:

```

``/n_iter`` [iteration]
  *(Integer)* Iteration numbers for each entry in other datasets.

``/n_segs`` [iteration]
  *(Integer)* Number of segment IDs matching the predicate (or inverted
  predicate, if -v/--invert is specified) in the given iteration.

``/seg_ids`` [iteration][segment]
  *(Integer)* Matching segments in each iteration. For an iteration
  ``n_iter``, only the first ``n_iter`` entries are valid. For example,
  the full list of matching seg_ids in the first stored iteration is
  ``seg_ids[0][:n_segs[0]]``.

``/weights`` [iteration][segment]
  *(Floating-point)* Weights for each matching segment in ``/seg_ids``.

```

6.19.14.3 Command-line arguments

optional arguments:

```
-h, --help          show this help message and exit
```

general options:

```

-r RCFILE, --rcfile RCFILE
                        use RCFILE as the WEST run-time configuration file (default: west.
↪cfg)
--quiet                emit only essential information
--verbose              emit extra information

```

(continues on next page)

(continued from previous page)

```
--debug          enable extra checks and emit copious information
--version        show program's version number and exit
```

parallelization options:

```
--max-queue-length MAX_QUEUE_LENGTH
                        Maximum number of tasks that can be queued. Useful to limit RAM.
↳ use for tasks that  have very large requests/response. Default: no limit.
```

WEST input data options:

```
-W WEST_H5FILE, --west-data WEST_H5FILE
                        Take WEST data from WEST_H5FILE (default: read from the HDF5 file.
↳ specified in        west.cfg).
```

iteration range:

```
--first-iter N_ITER   Begin analysis at iteration N_ITER (default: 1).
--last-iter N_ITER    Conclude analysis with N_ITER, inclusive (default: last completed.
↳ iteration).
```

selection options:

```
-p MODULE.FUNCTION, --predicate-function MODULE.FUNCTION
                        Use the given predicate function to match segments. This function.
↳ should take an      iteration number and the HDF5 group corresponding to that.
↳ iteration and return a sequence of seg_ids matching the predicate, as in ``match_
↳ predicate(n_iter,   iter_group)``.
-v, --invert          Invert the match predicate.
-a, --include-ancestors
                        Include ancestors of matched segments in output.
```

output options:

-o OUTPUT, --output OUTPUT Write output to OUTPUT (default: select.h5).

parallelization options:

```
--serial          run in serial mode
--parallel         run in parallel mode (using processes)
--work-manager WORK_MANAGER
                        use the given work manager for parallel task distribution.
↳ Available work managers
                        are ('serial', 'threads', 'processes', 'zmq'); default is 'serial'
--n-workers N_WORKERS
                        Use up to N_WORKERS on this host, for work managers which support.
↳ this option. Use    0 for a dedicated server. (Ignored by work managers which do not.
```

(continues on next page)

(continued from previous page)

↪support this
option.)

options for ZeroMQ (“zmq”) work manager (master or node):

```
--zmq-mode MODE      Operate as a master (server) or a node (workers/client). "server"
↪is a deprecated    synonym for "master" and "client" is a deprecated synonym for "node"
↪".
--zmq-comm-mode COMM_MODE
↪sockets for        Use the given communication mode -- TCP or IPC (Unix-domain) --
↪efficient but is not communication within a node. IPC (the default) may be more
↪storage (e.g. /tmp); available on (exceptionally rare) systems without node-local
on such systems, TCP may be used instead.
--zmq-write-host-info INFO_FILE
↪instance in        Store hostname and port information needed to connect to this
↪coordinating the   INFO_FILE. This allows the master and nodes assisting in
↪nodes read this    communication of other nodes to choose ports randomly. Downstream
file with --zmq-read-host-info and know where how to connect.
--zmq-read-host-info INFO_FILE
↪(or other          Read hostname and port information needed to connect to the master
↪nodes assisting in coordinating node) from INFO_FILE. This allows the master and
↪randomly, writing   coordinating the communication of other nodes to choose ports
that information with --zmq-write-host-info for this instance to
↪read.
--zmq-upstream-rr-endpoint ENDPOINT
↪result) traffic toward ZeroMQ endpoint to which to send request/response (task and
the master.
--zmq-upstream-ann-endpoint ENDPOINT
↪shutdown          ZeroMQ endpoint on which to receive announcement (heartbeat and
notification) traffic from the master.
--zmq-downstream-rr-endpoint ENDPOINT
↪result) traffic    ZeroMQ endpoint on which to listen for request/response (task and
from subsidiary workers.
--zmq-downstream-ann-endpoint ENDPOINT
↪shutdown          ZeroMQ endpoint on which to send announcement (heartbeat and
notification) traffic toward workers.
--zmq-master-heartbeat MASTER_HEARTBEAT
Every MASTER_HEARTBEAT seconds, the master announces its presence.
```

(continues on next page)

(continued from previous page)

```

↪to workers.
--zmq-worker-heartbeat WORKER_HEARTBEAT
    Every WORKER_HEARTBEAT seconds, workers announce their presence to
↪the master.
--zmq-timeout-factor FACTOR
    Scaling factor for heartbeat timeouts. If the master doesn't hear
↪from a worker in
    WORKER_HEARTBEAT*FACTOR, the worker is assumed to have crashed. If
↪a worker
    doesn't hear from the master in MASTER_HEARTBEAT*FACTOR seconds,
↪the master is
    assumed to have crashed. Both cases result in shutdown.
--zmq-startup-timeout STARTUP_TIMEOUT
    Amount of time (in seconds) to wait for communication between the
↪master and at
    least one worker. This may need to be changed on very large,
↪heavily-loaded
    computer systems that start all processes simultaneously.
--zmq-shutdown-timeout SHUTDOWN_TIMEOUT
    Amount of time (in seconds) to wait for workers to shut down.

```

6.19.15 w_states

usage:

```

w_states [-h] [-r RCFILE] [--quiet | --verbose | --debug] [--version]
    [--show | --append | --replace] [--bstate-file BSTATE_FILE] [--bstate
↪BSTATES]
    [--tstate-file TSTATE_FILE] [--tstate TSTATES]
    [--serial | --parallel | --work-manager WORK_MANAGER] [--n-workers N_
↪WORKERS]
    [--zmq-mode MODE] [--zmq-comm-mode COMM_MODE] [--zmq-write-host-info INFO_
↪FILE]
    [--zmq-read-host-info INFO_FILE] [--zmq-upstream-rr-endpoint ENDPOINT]
    [--zmq-upstream-ann-endpoint ENDPOINT] [--zmq-downstream-rr-endpoint
↪ENDPOINT]
    [--zmq-downstream-ann-endpoint ENDPOINT] [--zmq-master-heartbeat MASTER_
↪HEARTBEAT]
    [--zmq-worker-heartbeat WORKER_HEARTBEAT] [--zmq-timeout-factor FACTOR]
    [--zmq-startup-timeout STARTUP_TIMEOUT] [--zmq-shutdown-timeout SHUTDOWN_
↪TIMEOUT]

```

Display or manipulate basis (initial) or target (recycling) states for a WEST simulation. By default, states are displayed (or dumped to files). If `--replace` is specified, all basis/target states are replaced for the next iteration. If `--append` is specified, the given target state(s) are appended to the list for the next iteration. Appending basis states is not permitted, as this would require renormalizing basis state probabilities in ways that may be error-prone. Instead, use `w_states --show --bstate-file=bstates.txt` and then edit the resulting `bstates.txt` file to include the new desired basis states, then use `w_states --replace --bstate-file=bstates.txt` to update the WEST HDF5 file appropriately.

optional arguments:


```

-h, --help          show this help message and exit
--bstate-file BSTATE_FILE
                    Read (--append/--replace) or write (--show) basis state names,
↳ probabilities, and
                    data references from/to BSTATE_FILE.
--bstate BSTATES    Add the given basis state (specified as a string 'label,
↳ probability[,auxref]') to
                    the list of basis states (after those specified in --bstate-file,
↳ if any). This
                    argument may be specified more than once, in which case the given
↳ states are
                    appended in the order they are given on the command line.
--tstate-file TSTATE_FILE
                    Read (--append/--replace) or write (--show) target state names and
↳ representative
                    progress coordinates from/to TSTATE_FILE
--tstate TSTATES    Add the given target state (specified as a string 'label,pcoord0[,
↳ pcoord1[,...]]')
                    to the list of target states (after those specified in the file
↳ given by
                    --tstates-from, if any). This argument may be specified more than
↳ once, in which
                    case the given states are appended in the order they appear on the
↳ command line.

```

general options:

```

-r RCFILE, --rcfile RCFILE
                    use RCFILE as the WEST run-time configuration file (default: west.
↳ cfg)
--quiet            emit only essential information
--verbose         emit extra information
--debug          enable extra checks and emit copious information
--version         show program's version number and exit

```

modes of operation:

```

--show            Display current basis/target states (or dump to files).
--append         Append the given basis/target states to those currently in use.
--replace        Replace current basis/target states with those specified.

```

parallelization options:

```

--serial         run in serial mode
--parallel       run in parallel mode (using processes)
--work-manager WORK_MANAGER
                    use the given work manager for parallel task distribution.
↳ Available work managers
                    are ('serial', 'threads', 'processes', 'zmq'); default is 'serial'
--n-workers N_WORKERS
                    Use up to N_WORKERS on this host, for work managers which support
↳ this option. Use
                    0 for a dedicated server. (Ignored by work managers which do not

```

(continues on next page)

(continued from previous page)

↪support this
option.)

options for ZeroMQ (“zmq”) work manager (master or node):

```
--zmq-mode MODE      Operate as a master (server) or a node (workers/client). "server"
↪is a deprecated    synonym for "master" and "client" is a deprecated synonym for "node"
↪".
--zmq-comm-mode COMM_MODE
↪sockets for        Use the given communication mode -- TCP or IPC (Unix-domain) --
↪efficient but is not communication within a node. IPC (the default) may be more
↪storage (e.g. /tmp); available on (exceptionally rare) systems without node-local
on such systems, TCP may be used instead.
--zmq-write-host-info INFO_FILE
↪instance in        Store hostname and port information needed to connect to this
↪coordinating the   INFO_FILE. This allows the master and nodes assisting in
↪nodes read this    communication of other nodes to choose ports randomly. Downstream
file with --zmq-read-host-info and know where how to connect.
--zmq-read-host-info INFO_FILE
↪(or other          Read hostname and port information needed to connect to the master
↪nodes assisting in coordinating node) from INFO_FILE. This allows the master and
↪randomly, writing   coordinating the communication of other nodes to choose ports
that information with --zmq-write-host-info for this instance to
↪read.
--zmq-upstream-rr-endpoint ENDPOINT
↪result) traffic toward ZeroMQ endpoint to which to send request/response (task and
the master.
--zmq-upstream-ann-endpoint ENDPOINT
↪shutdown          ZeroMQ endpoint on which to receive announcement (heartbeat and
notification) traffic from the master.
--zmq-downstream-rr-endpoint ENDPOINT
↪result) traffic    ZeroMQ endpoint on which to listen for request/response (task and
from subsidiary workers.
--zmq-downstream-ann-endpoint ENDPOINT
↪shutdown          ZeroMQ endpoint on which to send announcement (heartbeat and
notification) traffic toward workers.
--zmq-master-heartbeat MASTER_HEARTBEAT
Every MASTER_HEARTBEAT seconds, the master announces its presence.
```

(continues on next page)

(continued from previous page)

```

↪to workers.
--zmq-worker-heartbeat WORKER_HEARTBEAT
    Every WORKER_HEARTBEAT seconds, workers announce their presence to
↪the master.
--zmq-timeout-factor FACTOR
    Scaling factor for heartbeat timeouts. If the master doesn't hear
↪from a worker in
    WORKER_HEARTBEAT*FACTOR, the worker is assumed to have crashed. If
↪a worker
    doesn't hear from the master in MASTER_HEARTBEAT*FACTOR seconds,
↪the master is
    assumed to have crashed. Both cases result in shutdown.
--zmq-startup-timeout STARTUP_TIMEOUT
    Amount of time (in seconds) to wait for communication between the
↪master and at
    least one worker. This may need to be changed on very large,
↪heavily-loaded
    computer systems that start all processes simultaneously.
--zmq-shutdown-timeout SHUTDOWN_TIMEOUT
    Amount of time (in seconds) to wait for workers to shut down.

```

6.19.16 w_eddist

usage:

```

w_eddist [-h] [-r RCFILE] [--quiet | --verbose | --debug] [--version]
    [--max-queue-length MAX_QUEUE_LENGTH] [-b BINEXPR] [-C] [--loose] --
↪istate ISTATE
    --fstate FSTATE [--first-iter ITER_START] [--last-iter ITER_STOP] [-k
↪KINETICS]
    [-o OUTPUT] [--serial | --parallel | --work-manager WORK_MANAGER]
    [--n-workers N_WORKERS] [--zmq-mode MODE] [--zmq-comm-mode COMM_MODE]
    [--zmq-write-host-info INFO_FILE] [--zmq-read-host-info INFO_FILE]
    [--zmq-upstream-rr-endpoint ENDPOINT] [--zmq-upstream-ann-endpoint
↪ENDPOINT]
    [--zmq-downstream-rr-endpoint ENDPOINT] [--zmq-downstream-ann-endpoint
↪ENDPOINT]
    [--zmq-master-heartbeat MASTER_HEARTBEAT] [--zmq-worker-heartbeat WORKER_
↪HEARTBEAT]
    [--zmq-timeout-factor FACTOR] [--zmq-startup-timeout STARTUP_TIMEOUT]
    [--zmq-shutdown-timeout SHUTDOWN_TIMEOUT]

```

Calculate time-resolved transition-event duration distribution from kinetics results

6.19.16.1 Source data

Source data is collected from the results of ‘w_kinetics trace’ (see w_kinetics trace –help for more information on generating this dataset).

6.19.16.2 Histogram binning

By default, histograms are constructed with 100 bins in each dimension. This can be overridden by specifying -b/-bins, which accepts a number of different kinds of arguments:

```
a single integer N
    N uniformly spaced bins will be used in each dimension.

a sequence of integers N1,N2,... (comma-separated)
    N1 uniformly spaced bins will be used for the first dimension, N2 for the
    second, and so on.

a list of lists [[B11, B12, B13, ...], [B21, B22, B23, ...], ...]
    The bin boundaries B11, B12, B13, ... will be used for the first dimension,
    B21, B22, B23, ... for the second dimension, and so on. These bin
    boundaries need not be uniformly spaced. These expressions will be
    evaluated with Python's `eval` construct, with `np` available for
    use [e.g. to specify bins using np.arange()].
```

The first two forms (integer, list of integers) will trigger a scan of all data in each dimension in order to determine the minimum and maximum values, which may be very expensive for large datasets. This can be avoided by explicitly providing bin boundaries using the list-of-lists form.

Note that these bins are *NOT* at all related to the bins used to drive WE sampling.

6.19.16.3 Output format

The output file produced (specified by -o/-output, defaulting to “pdist.h5”) may be fed to plohist to generate plots (or appropriately processed text or HDF5 files) from this data. In short, the following datasets are created:

```
`histograms`
    Normalized histograms. The first axis corresponds to iteration, and
    remaining axes correspond to dimensions of the input dataset.

`/binbounds_0`
    Vector of bin boundaries for the first (index 0) dimension. Additional
    datasets similarly named (/binbounds_1, /binbounds_2, ...) are created
    for additional dimensions.

`/midpoints_0`
    Vector of bin midpoints for the first (index 0) dimension. Additional
    datasets similarly named are created for additional dimensions.

`n_iter`
    Vector of iteration numbers corresponding to the stored histograms (i.e.
    the first axis of the `histograms` dataset).
```

6.19.16.4 Subsequent processing

The output generated by this program (-o/-output, default “pdist.h5”) may be plotted by the plothist program. See plothist --help for more information.

6.19.16.5 Parallelization

This tool supports parallelized binning, including reading of input data. Parallel processing is the default. For simple cases (reading pre-computed input data, modest numbers of segments), serial processing (-serial) may be more efficient.

6.19.16.6 Command-line options

optional arguments:

```
-h, --help          show this help message and exit
-b BINEXPR, --bins BINEXPR
                    Use BINEXPR for bins. This may be an integer, which will be used
                    for each
                    dimension of the progress coordinate; a list of integers
                    (formatted as
                    [n1,n2,...]) which will use n1 bins for the first dimension, n2
                    for the second
                    dimension, and so on; or a list of lists of boundaries (formatted
                    as [[a1, a2,
                    ...], [b1, b2, ...], ... ]), which will use [a1, a2, ...] as bin
                    boundaries for
                    the first dimension, [b1, b2, ...] as bin boundaries for the
                    second dimension,
                    and so on. (Default: 100 bins in each dimension.)
-C, --compress      Compress histograms. May make storage of higher-dimensional
                    histograms more
                    tractable, at the (possible extreme) expense of increased analysis
                    time.
                    (Default: no compression.)
--loose             Ignore values that do not fall within bins. (Risky, as this can
                    make buggy bin
                    boundaries appear as reasonable data. Only use if you are sure of
                    your bin
                    boundary specification.)
--istate ISTATE     Initial state defining transition event
--fstate FSTATE     Final state defining transition event
```

general options:

```
-r RCFILE, --rcfile RCFILE
                    use RCFILE as the WEST run-time configuration file (default: west.
                    cfg)
--quiet            emit only essential information
--verbose         emit extra information
--debug          enable extra checks and emit copious information
--version         show program's version number and exit
```

parallelization options:

--max-queue-length MAX_QUEUE_LENGTH Maximum number of tasks that can be queued. Useful to limit RAM use for tasks that have very large requests/response. Default: no limit.

iteration range options:

```
--first-iter ITER_START      Iteration to begin analysis (default: 1)
--last-iter ITER_STOP       Iteration to end analysis
```

input/output options:

```
-k KINETICS, --kinetics KINETICS      Populations and transition rates (including evolution) are stored
↳ in KINETICS                        (default: kintrace.h5).
-o OUTPUT, --output OUTPUT           Store results in OUTPUT (default: eddist.h5).
```

parallelization options:

--serial run in serial mode

--parallel run in parallel mode (using processes)

--work-manager WORK_MANAGER use the given work manager for parallel task distribution. Available work managers are ('serial', 'threads', 'processes', 'zmq'); default is 'processes'

--n-workers N_WORKERS Use up to N_WORKERS on this host, for work managers which support this option. Use 0 for a dedicated server. (Ignored by work managers which do not support this option.)

options for ZeroMQ ("zmq") work manager (master or node):

```
--zmq-mode MODE      Operate as a master (server) or a node (workers/client). "server"
↳ is a              deprecated synonym for "master" and "client" is a deprecated
↳ synonym for      "node".
--zmq-comm-mode COMM_MODE      Use the given communication mode -- TCP or IPC (Unix-domain) --
↳ sockets for      communication within a node. IPC (the default) may be more
↳ efficient but is not available on (exceptionally rare) systems without node-local
↳ storage (e.g.    /tmp); on such systems, TCP may be used instead.
--zmq-write-host-info INFO_FILE      Store hostname and port information needed to connect to this
↳ instance in      INFO_FILE. This allows the master and nodes assisting in
↳ coordinating the communication of other nodes to choose ports randomly. Downstream
↳ nodes read      this file with --zmq-read-host-info and know where how to connect.
```

(continues on next page)

(continued from previous page)

```

--zmq-read-host-info INFO_FILE
    Read hostname and port information needed to connect to the master.
↳(or other
    coordinating node) from INFO_FILE. This allows the master and
↳nodes assisting
    in coordinating the communication of other nodes to choose ports.
↳randomly,
    writing that information with --zmq-write-host-info for this
↳instance to read.
--zmq-upstream-rr-endpoint ENDPOINT
    ZeroMQ endpoint to which to send request/response (task and
↳result) traffic
    toward the master.
--zmq-upstream-ann-endpoint ENDPOINT
    ZeroMQ endpoint on which to receive announcement (heartbeat and
↳shutdown
    notification) traffic from the master.
--zmq-downstream-rr-endpoint ENDPOINT
    ZeroMQ endpoint on which to listen for request/response (task and
↳result)
    traffic from subsidiary workers.
--zmq-downstream-ann-endpoint ENDPOINT
    ZeroMQ endpoint on which to send announcement (heartbeat and
↳shutdown
    notification) traffic toward workers.
--zmq-master-heartbeat MASTER_HEARTBEAT
    Every MASTER_HEARTBEAT seconds, the master announces its presence
↳to workers.
--zmq-worker-heartbeat WORKER_HEARTBEAT
    Every WORKER_HEARTBEAT seconds, workers announce their presence to
↳the master.
--zmq-timeout-factor FACTOR
    Scaling factor for heartbeat timeouts. If the master doesn't hear
↳from a worker
    in WORKER_HEARTBEAT*FACTOR, the worker is assumed to have crashed.
↳If a worker
    doesn't hear from the master in MASTER_HEARTBEAT*FACTOR seconds,
↳the master is
    assumed to have crashed. Both cases result in shutdown.
--zmq-startup-timeout STARTUP_TIMEOUT
    Amount of time (in seconds) to wait for communication between the
↳master and at
    least one worker. This may need to be changed on very large,
↳heavily-loaded
    computer systems that start all processes simultaneously.
--zmq-shutdown-timeout SHUTDOWN_TIMEOUT
    Amount of time (in seconds) to wait for workers to shut down.

```

6.19.17 w_ntop

usage:

```
w_ntop [-h] [-r RCFILE] [--quiet | --verbose | --debug] [--version] [-W WEST_H5FILE]
      [--first-iter N_ITER] [--last-iter N_ITER] [-a ASSIGNMENTS] [-n COUNT] [-t TIMEPOINT]
      [--highweight | --lowweight | --random] [-o OUTPUT]
```

Select walkers from bins . An assignment file mapping walkers to bins at each timepoint is required (see `w_assign -help` for further information on generating this file). By default, high-weight walkers are selected (hence the name `w_ntop`: select the N top-weighted walkers from each bin); however, minimum weight walkers and randomly-selected walkers may be selected instead.

6.19.17.1 Output format

The output file (-o/-output, by default “ntop.h5”) contains the following datasets:

```
``/n_iter`` [iteration]
  *(Integer)* Iteration numbers for each entry in other datasets.

``/n_segs`` [iteration][bin]
  *(Integer)* Number of segments in each bin/state in the given iteration.
  This will generally be the same as the number requested with
  ``--n/--count`` but may be smaller if the requested number of walkers
  does not exist.

``/seg_ids`` [iteration][bin][segment]
  *(Integer)* Matching segments in each iteration for each bin.
  For an iteration ``n_iter``, only the first ``n_iter`` entries are
  valid. For example, the full list of matching seg_ids in bin 0 in the
  first stored iteration is ``seg_ids[0][0][:n_segs[0]]``.

``/weights`` [iteration][bin][segment]
  *(Floating-point)* Weights for each matching segment in ``/seg_ids``.
```

6.19.17.2 Command-line arguments

optional arguments:

<code>-h, --help</code>	show this help message and exit
<code>--highweight</code>	Select COUNT highest-weight walkers from each bin .
<code>--lowweight</code>	Select COUNT lowest-weight walkers from each bin .
<code>--random</code>	Select COUNT walkers randomly from each bin .

general options:

<code>-r RCFILE, --rcfile RCFILE</code>	use RCFILE as the WEST run-time configuration file (default: west. <code>cfg</code>)
<code>--quiet</code>	emit only essential information
<code>--verbose</code>	emit extra information

(continues on next page)

(continued from previous page)

```
--debug          enable extra checks and emit copious information
--version        show program's version number and exit
```

WEST input data options:

```
-W WEST_H5FILE, --west-data WEST_H5FILE
    Take WEST data from WEST_H5FILE (default: read from the HDF5 file
    specified in west.cfg).
```

iteration range:

```
--first-iter N_ITER  Begin analysis at iteration N_ITER (default: 1).
--last-iter N_ITER   Conclude analysis with N_ITER, inclusive (default: last completed
    iteration).
```

input options:

```
-a ASSIGNMENTS, --assignments ASSIGNMENTS
    Use assignments from the given ASSIGNMENTS file (default: assign.
    h5).
```

selection options:

```
-n COUNT, --count COUNT
    Select COUNT walkers from each iteration for each bin (default: 1).
-t TIMEPOINT, --timepoint TIMEPOINT
    Base selection on the given TIMEPOINT within each iteration.
    Default (-1) corresponds to the last timepoint.
```

output options:

```
-o OUTPUT, --output OUTPUT
    Write output to OUTPUT (default: ntop.h5).
```

6.19.18 plothist

6.19.18.1 plothist_instant

usage:

```
plothist instant [-h] [-o PLOT_OUTPUT] [--hdf5-output HDF5_OUTPUT] [--plot-contour]
    [--title TITLE] [--linear | --energy | --zero-energy E | --log10]
    [--range RANGE] [--postprocess-function POSTPROCESS_FUNCTION]
    [--text-output TEXT_OUTPUT] [--iter N_ITER]
    input [DIMENSION] [ADDTLDIM]
```

Plot a probability distribution for a single WE iteration. The probability distribution must have been previously extracted with `w_pdist` (or, at least, must be compatible with the output format of `w_pdist`; see `w_pdist --help` for more information).

optional arguments:

-h, --help	show this help message and exit
------------	---------------------------------

input options:

input	HDF5 file containing histogram data
DIMENSION	Plot for the given DIMENSION, specified as INT[:[LB,UB]:LABEL], ↳ where INT is a zero-based integer identifying the dimension in the histogram, LB, ↳ and UB are lower and upper bounds for plotting, and LABEL is the label for , ↳ the plot axis. (Default: dimension 0, full range.)
ADDTLDIM	For instantaneous/average plots, plot along the given additional, ↳ dimension, producing a color map .
--iter N_ITER	Plot distribution for iteration N_ITER (default: last completed, ↳ iteration).

output options:

-o PLOT_OUTPUT, --output PLOT_OUTPUT, --plot-output PLOT_OUTPUT	Store plot as PLOT_OUTPUT. This may be set to an empty string (e.g. ↳ --plot- output='') to suppress plotting entirely. The output format is , ↳ determined by filename extension (and thus defaults to PDF). Default: "hist.pdf".
--hdf5-output HDF5_OUTPUT	Store plot data in the HDF5 file HDF5_OUTPUT.
--plot-contour ↳ heatmap for 2D	Determines whether or not to superimpose a contour plot over the, ↳ objects.
--text-output TEXT_OUTPUT	Store plot data in a text format at TEXT_OUTPUT. This option is , ↳ only valid for 1-D histograms. (Default: no text output.)

plot options:

--title TITLE	Include TITLE as the top-of-graph title
--linear	Plot the histogram on a linear scale.
--energy ↳ to (free)	Plot the histogram on an inverted natural log scale, corresponding, ↳ energy (default).
--zero-energy E	Set the zero of energy to E, which may be a scalar, "min" or "max"
--log10	Plot the histogram on a base-10 log scale.
--range RANGE ↳ ", where LB	Plot histogram ordinates over the given RANGE, specified as "LB,UB ↳ and UB are the lower and upper bounds, respectively. For 1-D plots, ↳ this is the Y axis. For 2-D plots, this is the colorbar axis. (Default: full, ↳ range.)
--postprocess-function POSTPROCESS_FUNCTION	Names a function (as in module.function) that will be called just, ↳

(continues on next page)

(continued from previous page)

```

→prior to          saving the plot. The function will be called as ``postprocess(hist,
→ midpoints,        binbounds)`` where ``hist`` is the histogram that was plotted,
→ ``midpoints`` is   the bin midpoints for each dimension, and ``binbounds`` is the bin
→ boundaries         for each dimension for 2-D plots, or None otherwise. The plot must
→ be modified        in place using the pyplot stateful interface.

```

6.19.18.2 plothist_average

usage:

```

plothist average [-h] [-o PLOT_OUTPUT] [--hdf5-output HDF5_OUTPUT] [--plot-contour]
                  [--title TITLE] [--linear | --energy | --zero-energy E | --log10]
                  [--range RANGE] [--postprocess-function POSTPROCESS_FUNCTION]
                  [--text-output TEXT_OUTPUT] [--first-iter N_ITER] [--last-iter N_
→ITER]
                  input [DIMENSION] [ADDTLDIM]

```

Plot a probability distribution averaged over multiple iterations. The probability distribution must have been previously extracted with `w_pdist` (or, at least, must be compatible with the output format of `w_pdist`; see `w_pdist --help` for more information).

optional arguments:

```

-h, --help          show this help message and exit

```

input options:

```

input              HDF5 file containing histogram data
DIMENSION          Plot for the given DIMENSION, specified as INT[:[LB,UB]:LABEL],
→where INT is a    zero-based integer identifying the dimension in the histogram, LB
→and UB are        lower and upper bounds for plotting, and LABEL is the label for
→the plot axis.    (Default: dimension 0, full range.)
ADDTLDIM           For instantaneous/average plots, plot along the given additional
→dimension,        producing a color map.
--first-iter N_ITER Begin averaging at iteration N_ITER (default: 1).
--last-iter N_ITER  Conclude averaging with N_ITER, inclusive (default: last completed
→iteration).

```

output options:

```

-o PLOT_OUTPUT, --output PLOT_OUTPUT, --plot-output PLOT_OUTPUT
                  Store plot as PLOT_OUTPUT. This may be set to an empty string (e.g.
→ --plot-

```

(continues on next page)

(continued from previous page)

```

    output='') to suppress plotting entirely. The output format is
↳ determined by filename extension (and thus defaults to PDF). Default: "hist.pdf".
--hdf5-output HDF5_OUTPUT
    Store plot data in the HDF5 file HDF5_OUTPUT.
--plot-contour
    Determines whether or not to superimpose a contour plot over the
↳ heatmap for 2D objects.
--text-output TEXT_OUTPUT
    Store plot data in a text format at TEXT_OUTPUT. This option is
↳ only valid for 1-D histograms. (Default: no text output.)

```

plot options:

```

--title TITLE
    Include TITLE as the top-of-graph title
--linear
    Plot the histogram on a linear scale.
--energy
    Plot the histogram on an inverted natural log scale, corresponding
↳ to (free) energy (default).
--zero-energy E
    Set the zero of energy to E, which may be a scalar, "min" or "max"
--log10
    Plot the histogram on a base-10 log scale.
--range RANGE
    Plot histogram ordinates over the given RANGE, specified as "LB,UB
↳ ", where LB and UB are the lower and upper bounds, respectively. For 1-D plots,
↳ this is the Y axis. For 2-D plots, this is the colorbar axis. (Default: full
↳ range.)
--postprocess-function POSTPROCESS_FUNCTION
    Names a function (as in module.function) that will be called just
↳ prior to saving the plot. The function will be called as ``postprocess(hist,
↳ midpoints, binbounds)`` where ``hist`` is the histogram that was plotted,
↳ ``midpoints`` is the bin midpoints for each dimension, and ``binbounds`` is the bin
↳ boundaries for each dimension for 2-D plots, or None otherwise. The plot must
↳ be modified in place using the pyplot stateful interface.

```

6.19.18.3 plohist_evolution

usage:

```

plohist evolution [-h] [-o PLOT_OUTPUT] [--hdf5-output HDF5_OUTPUT] [--plot-contour]
    [--title TITLE] [--linear | --energy | --zero-energy E | --
↳ log10]
    [--range RANGE] [--postprocess-function POSTPROCESS_FUNCTION]
    [--first-iter N_ITER] [--last-iter N_ITER] [--step-iter STEP]
    input [DIMENSION]

```

Plot a probability distribution as it evolves over iterations. The probability distribution must have been previously extracted with `w_pdist` (or, at least, must be compatible with the output format of `w_pdist`; see `w_pdist --help` for more information).

optional arguments:

<code>-h, --help</code>	show this help message and exit
-------------------------	---------------------------------

input options:

<code>input</code>	HDF5 file containing histogram data
<code>DIMENSION</code>	Plot for the given DIMENSION, specified as <code>INT[:[LB,UB]:LABEL]</code> , ↪ where <code>INT</code> is a zero-based integer identifying the dimension in the histogram, <code>LB</code> , ↪ and <code>UB</code> are lower and upper bounds for plotting, and <code>LABEL</code> is the label for , ↪ the plot axis. (Default: dimension <code>0</code> , full range .)
<code>--first-iter N_ITER</code>	Begin analysis at iteration <code>N_ITER</code> (default: <code>1</code>).
<code>--last-iter N_ITER</code>	Conclude analysis with <code>N_ITER</code> , inclusive (default: last completed ↪ iteration).
<code>--step-iter STEP</code>	Average in blocks of <code>STEP</code> iterations.

output options:

<code>-o PLOT_OUTPUT, --output PLOT_OUTPUT, --plot-output PLOT_OUTPUT</code>	Store plot as <code>PLOT_OUTPUT</code> . This may be set to an empty string (e.g. ↪ <code>--plot-</code> <code>output=''</code>) to suppress plotting entirely. The output format is , ↪ determined by filename extension (and thus defaults to PDF). Default: <code>"hist.pdf"</code> .
<code>--hdf5-output HDF5_OUTPUT</code>	Store plot data in the HDF5 file <code>HDF5_OUTPUT</code> .
<code>--plot-contour</code> ↪ <code>heatmap for 2D</code>	Determines whether or not to superimpose a contour plot over the objects.

plot options:

<code>--title TITLE</code>	Include <code>TITLE</code> as the top-of-graph title
<code>--linear</code>	Plot the histogram on a linear scale.
<code>--energy</code> ↪ <code>to (free)</code>	Plot the histogram on an inverted natural log scale, corresponding, energy (default).
<code>--zero-energy E</code>	Set the zero of energy to <code>E</code> , which may be a scalar, "min" or "max"
<code>--log10</code>	Plot the histogram on a base-10 log scale.
<code>--range RANGE</code> ↪ <code>", where LB</code> ↪ <code>this is the</code> ↪ <code>range.)</code>	Plot histogram ordinates over the given <code>RANGE</code> , specified as <code>"LB,UB</code> and <code>UB</code> are the lower and upper bounds, respectively. For 1-D plots, Y axis. For 2-D plots, this is the colorbar axis. (Default: full, range.)
<code>--postprocess-function POSTPROCESS_FUNCTION</code> ↪ <code>prior to</code>	Names a function (as in <code>module.function</code>) that will be called just prior to

(continues on next page)

(continued from previous page)

```

→ midpoints,          saving the plot. The function will be called as ``postprocess(hist,
→ ``midpoints`` is    binbounds)`` where ``hist`` is the histogram that was plotted,
→ boundaries          the bin midpoints for each dimension, and ``binbounds`` is the bin
→ be modified         for each dimension for 2-D plots, or None otherwise. The plot must
                     in place using the pyplot stateful interface.

```

usage:

```

plothist [-h] [-r RCFILE] [--quiet | --verbose | --debug] [--version]
         {help,instant,average,evolution} ...

```

Plot probability density functions (histograms) generated by `w_pdist` or other programs conforming to the same output format. This program operates in one of three modes:

instant

Plot 1-D and 2-D histograms for an individual iteration. See ```plothist instant --help``` for more information.

average

Plot 1-D and 2-D histograms, averaged over several iterations. See ```plothist average --help``` for more information.

evolution

Plot the time evolution 1-D histograms as waterfall (heat map) plots. See ```plothist evolution --help``` for more information.

This program takes the output of `w_pdist` as input (see `w_pdist --help` for more information), and can generate any kind of graphical output that matplotlib supports.

6.19.18.4 Command-line options

optional arguments:

```

-h, --help          show this help message and exit

```

general options:

```

-r RCFILE, --rcfile RCFILE
                        use RCFILE as the WEST run-time configuration file (default: west.
→ cfg)
--quiet                emit only essential information
--verbose              emit extra information
--debug                enable extra checks and emit copious information
--version              show program's version number and exit

```

plotting modes:

```
{help,instant,average,evolution}
  help          print help for this command or individual subcommands
  instant       plot probability distribution for a single WE iteration
  average       plot average of a probability distribution over a WE simulation
  evolution     plot evolution of a probability distribution over the course of a
↳ WE simulation
```

6.19.19 ploterr

usage:

```
ploterrs [-h] [-r RCFILE] [--quiet | --verbose | --debug] [--version]
          {help,d.kinetics,d.probs,rw.probs,rw.kinetics,generic} ...
```

Plots error ranges for weighted ensemble datasets.

6.19.19.1 Command-line options

optional arguments:

```
-h, --help          show this help message and exit
```

general options:

```
-r RCFILE, --rcfile RCFILE
                        use RCFILE as the WEST run-time configuration file (default: west.
↳ cfg)
--quiet               emit only essential information
--verbose             emit extra information
--debug              enable extra checks and emit copious information
--version            show program's version number and exit
```

supported input formats:

```
{help,d.kinetics,d.probs,rw.probs,rw.kinetics,generic}
  help          print help for this command or individual subcommands
  d.kinetics     output of w_direct kinetics
  d.probs        output of w_direct probs
  rw.probs       output of w_reweight probs
  rw.kinetics    output of w_reweight kinetics
  generic        arbitrary HDF5 file and dataset
```

6.19.20 w_kinavg

WARNING: w_kinavg is being deprecated. Please use w_direct instead.

usage:

```
w_kinavg trace [-h] [-W WEST_H5FILE] [--first-iter N_ITER] [--last-iter N_ITER] [--step-iter STEP]
               [-a ASSIGNMENTS] [-o OUTPUT] [-k KINETICS] [--disable-bootstrap] [--disable-correl]
               [--alpha ALPHA] [--autocorrel-alpha ACALPHA] [--nsets NSETS]
               [-e {cumulative,blocked,none}] [--window-frac WINDOW_FRAC] [--disable-averages]
```

Calculate average rates/fluxes and associated errors from weighted ensemble data. Bin assignments (usually “assign.h5”) and kinetics data (usually “direct.h5”) data files must have been previously generated (see “w_assign -help” and “w_direct init -help” for information on generating these files).

The evolution of all datasets may be calculated, with or without confidence intervals.

6.19.20.1 Output format

The output file (-o/-output, usually “direct.h5”) contains the following dataset:

```
/avg_rates [state,state]
(Structured -- see below) State-to-state rates based on entire window of
iterations selected.

/avg_total_fluxes [state]
(Structured -- see below) Total fluxes into each state based on entire
window of iterations selected.

/avg_conditional_fluxes [state,state]
(Structured -- see below) State-to-state fluxes based on entire window of
iterations selected.
```

If -evolution-mode is specified, then the following additional datasets are available:

```
/rate_evolution [window][state][state]
(Structured -- see below). State-to-state rates based on windows of
iterations of varying width. If --evolution-mode=cumulative, then
these windows all begin at the iteration specified with
--start-iter and grow in length by --step-iter for each successive
element. If --evolution-mode=blocked, then these windows are all of
width --step-iter (excluding the last, which may be shorter), the first
of which begins at iteration --start-iter.

/target_flux_evolution [window,state]
(Structured -- see below). Total flux into a given macro state based on
windows of iterations of varying width, as in /rate_evolution.

/conditional_flux_evolution [window,state,state]
(Structured -- see below). State-to-state fluxes based on windows of
varying width, as in /rate_evolution.
```


The structure of these datasets is as follows:

```

iter_start
  (Integer) Iteration at which the averaging window begins (inclusive).

iter_stop
  (Integer) Iteration at which the averaging window ends (exclusive).

expected
  (Floating-point) Expected (mean) value of the observable as evaluated within
  this window, in units of inverse tau.

ci_lbound
  (Floating-point) Lower bound of the confidence interval of the observable
  within this window, in units of inverse tau.

ci_ubound
  (Floating-point) Upper bound of the confidence interval of the observable
  within this window, in units of inverse tau.

stderr
  (Floating-point) The standard error of the mean of the observable
  within this window, in units of inverse tau.

corr_len
  (Integer) Correlation length of the observable within this window, in units
  of tau.

```

Each of these datasets is also stamped with a number of attributes:

```

mcbs_alpha
  (Floating-point) Alpha value of confidence intervals. (For example,
  *alpha=0.05* corresponds to a 95% confidence interval.)

mcbs_nsets
  (Integer) Number of bootstrap data sets used in generating confidence
  intervals.

mcbs_acalpha
  (Floating-point) Alpha value for determining correlation lengths.

```

6.19.20.2 Command-line options

optional arguments:

```
-h, --help          show this help message and exit
```

WEST input data options:

```

-W WEST_H5FILE, --west-data WEST_H5FILE
                        Take WEST data from WEST_H5FILE (default: read from the HDF5 file,
↳ specified in west.cfg).

```

iteration range:

```
--first-iter N_ITER   Begin analysis at iteration N_ITER (default: 1).
--last-iter N_ITER    Conclude analysis with N_ITER, inclusive (default: last completed
↳ iteration).
--step-iter STEP      Analyze/report in blocks of STEP iterations.
```

input/output options:

```
-a ASSIGNMENTS, --assignments ASSIGNMENTS
    Bin assignments and macrostate definitions are in ASSIGNMENTS
↳ (default:
    assign.h5).
-o OUTPUT, --output OUTPUT
    Store results in OUTPUT (default: kinavg.h5).
```

input/output options:

```
-k KINETICS, --kinetics KINETICS
    Populations and transition rates are stored in KINETICS (default:
↳ kintrace.h5).
```

confidence interval calculation options:

```
--disable-bootstrap, -db
    Enable the use of Monte Carlo Block Bootstrapping.
--disable-correl, -dc
    Disable the correlation analysis.
--alpha ALPHA
    Calculate a (1-ALPHA) confidence interval' (default: 0.05)
--autocorrel-alpha ACALPHA
    Evaluate autocorrelation to (1-ACALPHA) significance. Note that
↳ too small an
    ACALPHA will result in failure to detect autocorrelation in a
↳ noisy flux signal.
    (Default: same as ALPHA.)
--nsets NSETS
    Use NSETS samples for bootstrapping (default: chosen based on
↳ ALPHA)
```

calculation options:

```
-e {cumulative,blocked,none}, --evolution-mode {cumulative,blocked,none}
    How to calculate time evolution of rate estimates. ``cumulative``
↳ evaluates rates
    over windows starting with --start-iter and getting progressively
↳ wider to --stop-
    iter by steps of --step-iter. ``blocked`` evaluates rates over
↳ windows of width
    --step-iter, the first of which begins at --start-iter. ``none``
↳ (the default)
    disables calculation of the time evolution of rate estimates.
--window-frac WINDOW_FRAC
    Fraction of iterations to use in each window when running in
↳ ``cumulative`` mode.
    The (1 - frac) fraction of iterations will be discarded from the
```

(continues on next page)

(continued from previous page)

```
↪start of each
        window.
```

misc options:

```
--disable-averages, -da
        Whether or not the averages should be printed to the console (set_
↪to FALSE if flag
        is used).
```

6.19.21 w_kinetics

WARNING: w_kinetics is being deprecated. Please use w_direct instead.

usage:

```
w_kinetics trace [-h] [-W WEST_H5FILE] [--first-iter N_ITER] [--last-iter N_ITER]
               [--step-iter STEP] [-a ASSIGNMENTS] [-o OUTPUT]
```

Calculate state-to-state rates and transition event durations by tracing trajectories.

A bin assignment file (usually “assign.h5”) including trajectory labeling is required (see “w_assign -help” for information on generating this file).

This subcommand for w_direct is used as input for all other w_direct subcommands, which will convert the flux data in the output file into average rates/fluxes/populations with confidence intervals.

6.19.21.1 Output format

The output file (-o/-output, by default “direct.h5”) contains the following datasets:

```
``/conditional_fluxes`` [iteration][state][state]
*(Floating-point)* Macrostate-to-macrostate fluxes. These are **not**
normalized by the population of the initial macrostate.

``/conditional_arrivals`` [iteration][stateA][stateB]
*(Integer)* Number of trajectories arriving at state *stateB* in a given
iteration, given that they departed from *stateA*.

``/total_fluxes`` [iteration][state]
*(Floating-point)* Total flux into a given macrostate.

``/arrivals`` [iteration][state]
*(Integer)* Number of trajectories arriving at a given state in a given
iteration, regardless of where they originated.

``/duration_count`` [iteration]
*(Integer)* The number of event durations recorded in each iteration.

``/durations`` [iteration][event duration]
*(Structured -- see below)* Event durations for transition events ending
during a given iteration. These are stored as follows:
```

(continues on next page)

(continued from previous page)

```

istate
    *(Integer)* Initial state of transition event.
fstate
    *(Integer)* Final state of transition event.
duration
    *(Floating-point)* Duration of transition, in units of tau.
weight
    *(Floating-point)* Weight of trajectory at end of transition, **not**
    normalized by initial state population.

```

Because state-to-state fluxes stored in this file are not normalized by initial macrostate population, they cannot be used as rates without further processing. The `w_direct kinetics` command is used to perform this normalization while taking statistical fluctuation and correlation into account. See `w_direct kinetics --help` for more information. Target fluxes (total flux into a given state) require no such normalization.

6.19.21.2 Command-line options

optional arguments:

```
-h, --help          show this help message and exit
```

WEST input data options:

```

-W WEST_H5FILE, --west-data WEST_H5FILE
    Take WEST data from WEST_H5FILE (default: read from the HDF5 file,
↳ specified in west.cfg).

```

iteration range:

```

--first-iter N_ITER  Begin analysis at iteration N_ITER (default: 1).
--last-iter N_ITER   Conclude analysis with N_ITER, inclusive (default: last completed,
↳ iteration).
--step-iter STEP     Analyze/report in blocks of STEP iterations.

```

input/output options:

```

-a ASSIGNMENTS, --assignments ASSIGNMENTS
    Bin assignments and macrostate definitions are in ASSIGNMENTS,
↳ (default: assign.h5).
-o OUTPUT, --output OUTPUT
    Store results in OUTPUT (default: kintrace.h5).

```

6.19.22 w_stateprobs

WARNING: w_stateprobs is being deprecated. Please use w_direct instead.

usage:

```
w_stateprobs trace [-h] [-W WEST_H5FILE] [--first-iter N_ITER] [--last-iter N_ITER]
                  [--step-iter STEP] [-a ASSIGNMENTS] [-o OUTPUT] [-k KINETICS]
                  [--disable-bootstrap] [--disable-correl] [--alpha ALPHA]
                  [--autocorrel-alpha ACALPHA] [--nsets NSETS] [-e {cumulative,
↪ blocked,none}]
                  [--window-frac WINDOW_FRAC] [--disable-averages]
```

Calculate average populations and associated errors in state populations from weighted ensemble data. Bin assignments, including macrostate definitions, are required. (See “w_assign -help” for more information).

6.19.22.1 Output format

The output file (-o/-output, usually “direct.h5”) contains the following dataset:

```
/avg_state_probs [state]
  (Structured -- see below) Population of each state across entire
  range specified.

/avg_color_probs [state]
  (Structured -- see below) Population of each ensemble across entire
  range specified.
```

If -evolution-mode is specified, then the following additional datasets are available:

```
/state_pop_evolution [window][state]
  (Structured -- see below). State populations based on windows of
  iterations of varying width. If --evolution-mode=cumulative, then
  these windows all begin at the iteration specified with
  --start-iter and grow in length by --step-iter for each successive
  element. If --evolution-mode=blocked, then these windows are all of
  width --step-iter (excluding the last, which may be shorter), the first
  of which begins at iteration --start-iter.

/color_prob_evolution [window][state]
  (Structured -- see below). Ensemble populations based on windows of
  iterations of varying width. If --evolution-mode=cumulative, then
  these windows all begin at the iteration specified with
  --start-iter and grow in length by --step-iter for each successive
  element. If --evolution-mode=blocked, then these windows are all of
  width --step-iter (excluding the last, which may be shorter), the first
  of which begins at iteration --start-iter.
```

The structure of these datasets is as follows:

```
iter_start
  (Integer) Iteration at which the averaging window begins (inclusive).

iter_stop
```

(continues on next page)

(continued from previous page)

(Integer) Iteration at which the averaging window ends (exclusive).

expected

(Floating-point) Expected (mean) value of the observable **as** evaluated within this window, **in** units of inverse tau.

ci_lbound

(Floating-point) Lower bound of the confidence interval of the observable within this window, **in** units of inverse tau.

ci_ubound

(Floating-point) Upper bound of the confidence interval of the observable within this window, **in** units of inverse tau.

stderr

(Floating-point) The standard error of the mean of the observable within this window, **in** units of inverse tau.

corr_len

(Integer) Correlation length of the observable within this window, **in** units of tau.

Each of these datasets is also stamped with a number of attributes:

mcbs_alpha

(Floating-point) Alpha value of confidence intervals. (For example, ***alpha=0.05*** corresponds to a **95%** confidence interval.)

mcbs_nsets

(Integer) Number of bootstrap data sets used **in** generating confidence intervals.

mcbs_acalpha

(Floating-point) Alpha value **for** determining correlation lengths.

6.19.22.2 Command-line options

optional arguments:

-h, --help show this help message **and** exit

WEST input data options:

-W WEST_H5FILE, --west-data WEST_H5FILE
 Take WEST data **from** **WEST_H5FILE** (default: read **from** **the** HDF5 file **↪** specified **in** **west.cfg**).

iteration range:

--first-iter N_ITER Begin analysis at iteration N_ITER (default: **1**).
--last-iter N_ITER Conclude analysis **with** N_ITER, inclusive (default: last completed **↪**

(continues on next page)

(continued from previous page)

```
↪iteration).
--step-iter STEP      Analyze/report in blocks of STEP iterations.
```

input/output options:

```
-a ASSIGNMENTS, --assignments ASSIGNMENTS
                        Bin assignments and macrostate definitions are in ASSIGNMENTS.
↪(default:
                        assign.h5).
-o OUTPUT, --output OUTPUT
                        Store results in OUTPUT (default: stateprobs.h5).
```

input/output options:

```
-k KINETICS, --kinetics KINETICS
                        Populations and transition rates are stored in KINETICS (default:
↪assign.h5).
```

confidence interval calculation options:

```
--disable-bootstrap, -db
                        Enable the use of Monte Carlo Block Bootstrapping.
--disable-correl, -dc
                        Disable the correlation analysis.
--alpha ALPHA          Calculate a (1-ALPHA) confidence interval' (default: 0.05)
--autocorrel-alpha ACALPHA
                        Evaluate autocorrelation to (1-ACALPHA) significance. Note that
↪too small an
                        ACALPHA will result in failure to detect autocorrelation in a
↪noisy flux signal.
                        (Default: same as ALPHA.)
--nsets NSETS          Use NSETS samples for bootstrapping (default: chosen based on
↪ALPHA)
```

calculation options:

```
-e {cumulative,blocked,none}, --evolution-mode {cumulative,blocked,none}
                        How to calculate time evolution of rate estimates. ``cumulative``
↪evaluates rates
                        over windows starting with --start-iter and getting progressively
↪wider to --stop-
                        iter by steps of --step-iter. ``blocked`` evaluates rates over
↪windows of width
                        --step-iter, the first of which begins at --start-iter. ``none``
↪(the default)
                        disables calculation of the time evolution of rate estimates.
--window-frac WINDOW_FRAC
                        Fraction of iterations to use in each window when running in
↪``cumulative`` mode.
                        The (1 - frac) fraction of iterations will be discarded from the
↪start of each
                        window.
```

misc options:

```
--disable-averages, -da
                                Whether or not the averages should be printed to the console (set
→to FALSE if flag
                                is used).
```

6.20 HDF5 File Schema

WESTPA stores all of its simulation data in the cross-platform, self-describing [HDF5](#) file format. This file format can be read and written by a variety of languages and toolkits, including C/C++, Fortran, Python, Java, and [Matlab](#) so that analysis of weighted ensemble simulations is not tied to using the WESTPA framework. HDF5 files are organized like a filesystem, where arbitrarily-nested groups (i.e. directories) are used to organize datasets (i.e. files). The excellent [HDFView](#) program may be used to explore WEST data files.

The canonical file format reference for a given version of the WEST code is described in [src/west/data_manager.py](#).

6.20.1 Overall structure

```
/
#ibstates/
  index
  naming
    bstate_index
    bstate_pcoord
    istate_index
    istate_pcoord
#tstates/
  index
bin_topologies/
  index
  pickles
iterations/
  iter_XXXXXXXX/\|iter_XXXXXXXX/
    auxdata/
    bin_target_counts
    ibstates/
      bstate_index
      bstate_pcoord
      istate_index
      istate_pcoord
    pcoord
    seg_index
    wtgraph
  ...
summary
```


6.20.2 The root group (/)

The root of the WEST HDF5 file contains the following entries (where a trailing “/” denotes a group):

Name	Type	Description
ibstates/	Group	Initial and basis states for this simulation
tstates/	Group	Target (recycling) states for this simulation; may be empty
bin_topologies/	Group	Data pertaining to the binning scheme used in each iteration
iterations/	Group	Iteration data
summary	Dataset (1-dimensional, compound)	Summary data by iteration

6.20.2.1 The iteration summary table (/summary)

Field	Description
n_particles	the total number of walkers in this iteration
norm	total probability, for stability monitoring
min_bin_prob	smallest probability contained in a bin
max_bin_prob	largest probability contained in a bin
min_seg_prob	smallest probability carried by a walker
max_seg_prob	largest probability carried by a walker
cputime	total CPU time (in seconds) spent on propagation for this iteration
walltime	total wallclock time (in seconds) spent on this iteration
binhash	a hex string identifying the binning used in this iteration

6.20.3 Per iteration data (/iterations/iter_XXXXXXX)

Data for each iteration is stored in its own group, named according to the iteration number and zero-padded out to 8 digits, as in `/iterations/iter_00000001` for iteration 1. This is done solely for convenience in dealing with the data in external utilities that sort output by group name lexicographically. The field width is in fact configurable via the `iter_prec` configuration entry under `data` section of the WESTPA configuration file.

The HDF5 group for each iteration contains the following elements:

Name	Type	Description
auxdata/	Group	All user-defined auxiliary data0 sets
bin_target_co	Dataset (1-dimensional)	The per-bin target count for the iteration
ibstates/	Group	Initial and basis state data for the iteration
pcoord	Dataset (3-dimensional)	Progress coordinate data for the iteration stored as a (num of segments, pcoord_len, pcoord_ndim) array
seg_index	Dataset (1-dimensional, compound)	Summary data for each segment
wtgraph	Dataset (1-dimensional)	

6.20.3.1 The segment summary table (/iterations/iter_XXXXXXX/seg_index)

Field	Description
weight	Segment weight
parent_id	Index of parent
wtg_n_parents	
wtg_offset	
cputime	Total cpu time required to run the segment
walltime	Total walltime required to run the segment
endpoint_type	
status	

6.20.4 Bin Topologies group (/bin_topologies)

Bin topologies used during a WE simulation are stored as a unique hash identifier and a serialized `BinMapper` object in `python pickle` format. This group contains two datasets:

- **index**: Compound array containing the bin hash and pickle length
- **pickle**: The pickled `BinMapper` objects for each unique mapper stored in a (num unique mappers, max pickled size) array

6.21 Checklist

6.21.1 Configuring a WESTPA Simulation

- Files for dynamics propagation
 - Have you set up all of the files for propagating the dynamics (e.g. for GROMACS, the `.top`, `.gro`, `.mdp`, and `.ndx` files)?
- System implementation (`system.py`)
 - Is `self.pcoord_len` set to the number of data points that corresponds to the frequency with which the dynamics engine outputs the progress coordinate? Note: Many MD engines (e.g. GROMACS) output the initial point (i.e. zero).
 - Are the bins in the expected positions? You can easily view the positions of the bins using a Python interpreter.
- Initializing the simulation (`init.sh`)
 - Is the directory structure for the trajectory output files consistent with specifications in the master configuration file (`west.cfg`)?
 - Are the basis (`bstate`) states, and if applicable, target states (`tstate`), specified correctly?
- Calculating the progress coordinate for initial states (`get_pcoord.sh`)
 - Ensure that the procedure to extract the progress coordinate works by manually checking the procedure on one (or more) basis state files.
 - If your initialization (`init.sh`) gives an error message indicating the “incorrect shape” of the progress coordinate, check that `get_pcoord.sh` is not writing to a single file. If this is the case, `w_init` will crash

- One of the most error prone part during an iteration is the progress coordinate extraction. Programs that are not designed for quick execution have a lot of trouble during this step (VMD is a very commonly encountered one for example). Probably the best way to deal with this issue is to hard code a script to do the progress coordinate extraction. If you are doing molecular dynamics simulations multiple libraries for Python and C/C++ that deal with most output formats for MD packages exist and they usually come with a lot of convenience functions that can help you extract the progress coordinate. AMBER tools and GROMACS tools seems to work adequately for this purpose as well.
- Is your progress coordinate what you think it is?
 - Once your simulation it is running, it is well worth your time to ensure that the progress coordinate being reported is what you think it is. This can be done in a number of ways:
 - Check the `seg_log` output. This captures the standard error/output from the terminal session that your segment ran in, assuming you are running the executable propagator, and can be useful to ensure that everything is being done as you believe it should be (GROMACS tools, such as `g_dist`, for instance, report what groups have their distance being calculated here).
 - Look at a structure! Do so in a program such as VMD or pyMOL, and calculate your progress coordinate manually and check it visually, if feasible. Does it look correct, and seem to match what's being reported in the `.h5` file? This is well worth your time before the simulation has proceeded very far, and can save a significant amount of wallclock and computational time.

6.21.3 Analyzing a WESTPA simulation

- If you are running the analysis on shared computing resources
 - Be sure to use the `--serial` flag (see the individual tool documentation). Otherwise, many of the included tools default to parallel mode (`w_assign`, for instance), which will create as many Python threads as there are CPU cores available.

6.22 Frequently Asked Questions (FAQ)

This page may be outdated, the most recent list of FAQs are available [here](#):

6.22.1 Simulation

- How can I cleanly shutdown a simulation (without corrupting the `h5` file)?

It is generally safe to shutdown a WESTPA simulation by simply canceling the job through your queue management. However, to ensure data integrity in the `h5` file, you should wait until the WESTPA log indicates that an iteration has begun or is occurring; canceling a job too quickly after submission can result in the absolute corruption of the `h5` file and should be avoided.

- Storage of Large Files

During a normal WESTPA run, many small files are created and it is convenient to tar these into a larger file (one tarball per iteration, for instance). It is generally best to do this ‘offline’. An important aspect to consider is that some disk systems, such as LUSTRE, will suffer impaired performance if very large files are created. On Stampede, for instance, any file larger than 200 GB must be ‘striped’ properly (such that its individual bits are spread across numerous disks).

Within the user guide for such systems, there is generally a section on how to handle large files. Some computers have special versions of tar which stripe appropriately; others do not (such as Stampede). For those that do not, it may be necessary to contact the sysadmin, and/or create a directory where you can place your tarball with a different stripe level than the default.

- H5py Inflate() Failed error

While running or analyzing a simulation, you may run into an error such as `IOError: Can't write data (Inflate() failed)`. These errors may be related to an open bug in H5py. However, the following tips may help you to find a workaround.

WESTPA may present you with such an error when unable to read or write a data set. In the case that a simulation gives this error when you attempt to run it, it may be helpful to check if a data set may be read or written to using an interactive Python session. Restarting the simulation may require deleting and remaking the data set. Also, this error may be related to compression and other storage options. Thus, it may be helpful to disable compression and chunked storage. Note that existing datasets will retain compression and other options given to them at the time of their creation, so it may be necessary to truncate an iteration (for example, using `w_truncate`) in order for changes to take effect.

This error may also occur during repeated opening (e.g., 1000s of times) of an HDF5 data set. Thus, this error may occur while running analysis scripts. In this case, it may be helpful to cache data sets in physical memory (RAM) as numpy arrays when they are read, so that the script loads the dataset a minimal number of times.

- Dynamics Packages

WESTPA was designed to work cleanly with any dynamics package available (using the executable propagator); however, many of the tips and tricks available on the web or the user manual for these packages make the (reasonable) assumption that you will be running a set of brute force trajectories. As such, some of their guidelines for handling periodic boundary conditions may not be applicable.

- How can I restart a WESTPA simulation?

In general restarting a westpa simulation will restart an incomplete iteration, retaining data from segments that have completed and re-running segments that were incomplete (or never started).

In case that the iteration data got corrupted or you want to go back to an specific iteration and change something, you need to delete all the trajectory segments and other files related to that iteration and run `w_truncate` on that iteration. This will delete westpa's information about the nth iteration, which includes which segments have run and which have not. Then restarting your westpa simulation will restart that iteration afresh.

6.22.2 GROMACS

- Periodic Boundary Conditions

While many of the built in tools now handle periodic boundary conditions cleanly (such as `g_dist`) with relatively little user interaction, others, such as `g_rms`, do not. If your simulation analysis protocol requires you to run such a tool, you must correct for the periodic boundary conditions before running it. While there are guidelines available to help you correct for whatever conditions your system may have [here](#), there is an implicit assumption that you have one long running trajectory.

It will be necessary, within your executable propagator (usually `runseg.sh`) to run `trjconv` (typically, two or three times, depending on your needs: once to remove the periodic boundary conditions, then to make molecules whole, then to remove any jumps). If no extra input is supplied (the `-s` flag in GROMACS 4.X), GROMACS uses the first frame of your segment trajectory as a reference state to remove jumps. If your segment's parent ended the previous iteration having jumped across the box barrier, `trjconv` will erroneously assume this is the correct state and 'correct' any jump back across the barrier. **This can result in unusually high RMSD values for one segment for one or more iterations**, and can show as discontinuities on the probability distribution. It is important to note that a lack of discontinuities does not imply a lack of imaging problems.

To fix this, simply pass in the last frame of the imaged parent trajectory and use that as the reference structure for `trjconv`. This will ensure that `trjconv` is aware if your segment has crossed the barrier at time 0 and will make the appropriate corrections.

6.22.3 Development

- I'm trying to profile a parallel script using the `–profile` option of `bin/west`. I get a `PicklingError`. What gives?

When executing a script using `–profile`, the following error may crop up:

```
PicklingError: Can't pickle <type 'function'>: attribute lookup __builtin__.function_
↪failed
```

The `cProfile` module used by the `–profile` option modifies function definitions such that they are no longer pickleable, meaning that they cannot be passed through the work manager to other processes. If you absolutely must profile a parallel script, use the `threads` work manager.

PYTHON MODULE INDEX

W

westpa.analysis.core, 332
westpa.analysis.statistics, 340
westpa.analysis.trajectories, 338
westpa.cli.core.w_fork, 24
westpa.cli.core.w_init, 15
westpa.cli.core.w_run, 22
westpa.cli.core.w_states, 84
westpa.cli.core.w_succ, 57
westpa.cli.core.w_truncate, 23
westpa.cli.tools.ploterr, 116
westpa.cli.tools.plothist, 111
westpa.cli.tools.w_assign, 28
westpa.cli.tools.w_bins, 19
westpa.cli.tools.w_crawl, 63
westpa.cli.tools.w_direct, 68
westpa.cli.tools.w_dumpsegs, 140
westpa.cli.tools.w_eddist, 91
westpa.cli.tools.w_fluxanl, 157
westpa.cli.tools.w_ipa, 45
westpa.cli.tools.w_kinavg, 123
westpa.cli.tools.w_kinetics, 129
westpa.cli.tools.w_multi_west, 102
westpa.cli.tools.w_ntop, 96
westpa.cli.tools.w_pdist, 52
westpa.cli.tools.w_postanalysis_matrix, 142
westpa.cli.tools.w_postanalysis_reweight, 145
westpa.cli.tools.w_red, 105
westpa.cli.tools.w_reweight, 147
westpa.cli.tools.w_select, 78
westpa.cli.tools.w_stateprobs, 136
westpa.cli.tools.w_trace, 37
westpa.core, 188
westpa.core.binning, 162
westpa.core.binning.assign, 349
westpa.core.binning.bins, 349
westpa.core.binning.mab, 168
westpa.core.binning.mab_driver, 168
westpa.core.binning.mab_manager, 170
westpa.core.data_manager, 188
westpa.core.extloader, 198
westpa.core.h5io, 198
westpa.core.kinetics, 174
westpa.core.kinetics.events, 175
westpa.core.kinetics.matrates, 175
westpa.core.kinetics.rate_averaging, 177
westpa.core.progress, 222
westpa.core.propagators, 179
westpa.core.propagators.executable, 180
westpa.core.reweight, 187
westpa.core.reweight.matrix, 187
westpa.core.segment, 224
westpa.core.sim_manager, 225
westpa.core.states, 230
westpa.core.systems, 233
westpa.core.textio, 234
westpa.core.we_driver, 234
westpa.core.wm_ops, 238
westpa.core.yamlcfg, 349
westpa.fasthist, 298
westpa.mclib, 298
westpa.oldtools, 300
westpa.oldtools.iframe, 301
westpa.oldtools.iframe.atool, 307
westpa.oldtools.iframe.base_mixin, 307
westpa.oldtools.iframe.binning, 308
westpa.oldtools.iframe.data_reader, 308
westpa.oldtools.iframe.iter_range, 312
westpa.oldtools.iframe.kinetics, 313
westpa.oldtools.iframe.mcbs, 313
westpa.oldtools.iframe.output, 314
westpa.oldtools.iframe.plotting, 315
westpa.oldtools.iframe.trajwalker, 315
westpa.oldtools.iframe.transitions, 315
westpa.oldtools.cmds, 317
westpa.oldtools.files, 300
westpa.oldtools.miscfn, 301
westpa.oldtools.stats, 317
westpa.oldtools.stats.accumulator, 317
westpa.oldtools.stats.edfs, 318
westpa.oldtools.stats.mcbs, 319
westpa.tools, 276
westpa.tools.binning, 283
westpa.tools.core, 285

- `westpa.tools.data_reader`, 288
- `westpa.tools.dtypes`, 290
- `westpa.tools.iter_range`, 290
- `westpa.tools.kinetics_tool`, 292
- `westpa.tools.plot`, 294
- `westpa.tools.progress`, 294
- `westpa.tools.selected_segs`, 295
- `westpa.tools.wipi`, 297
- `westpa.trajtree`, 299
- `westpa.trajtree.trajtree`, 300
- `westpa.westext`, 329
 - `westpa.westext.adaptvoronoi`, 320
 - `westpa.westext.adaptvoronoi.adaptVor_driver`, 319
 - `westpa.westext.weed`, 327
 - `westpa.westext.weed.BinCluster`, 326
 - `westpa.westext.weed.ProbAdjustEquil`, 326
 - `westpa.westext.weed.UncertMath`, 326
 - `westpa.westext.weed.weed_driver`, 327
 - `westpa.westext.wess`, 329
 - `westpa.westext.wess.ProbAdjust`, 328
 - `westpa.westext.wess.wess_driver`, 328
- `westpa.work_managers`, 241
 - `westpa.work_managers.core`, 242
 - `westpa.work_managers.environment`, 245
 - `westpa.work_managers.mpi`, 246
 - `westpa.work_managers.processes`, 249
 - `westpa.work_managers.serial`, 252
 - `westpa.work_managers.threads`, 254
 - `westpa.work_managers.zeromq`, 256
 - `westpa.work_managers.zeromq.core`, 259
 - `westpa.work_managers.zeromq.node`, 263
 - `westpa.work_managers.zeromq.work_manager`, 265
 - `westpa.work_managers.zeromq.worker`, 272

A

- `accumulate_labeled_populations()` (in module `westpa.cli.tools.w_assign`), 29
- `accumulate_labeled_populations()` (in module `westpa.core.binning`), 164
- `accumulate_state_populations_from_labeled()` (in module `westpa.cli.tools.w_direct`), 70
- `accumulate_state_populations_from_labeled()` (in module `westpa.core.binning`), 164
- `accumulate_statistics()` (`westpa.cli.tools.w_reweight.RWReweight` method), 150
- `AccuracyError`, 236
- `ACK` (`westpa.work_managers.zeromq.core.Message` attribute), 260
- `ACK` (`westpa.work_managers.zeromq.node.Message` attribute), 264
- `ACK` (`westpa.work_managers.zeromq.work_manager.Message` attribute), 267
- `ACK` (`westpa.work_managers.zeromq.worker.Message` attribute), 274
- `AdaptiveVoronoiDriver` (class in `westpa.westext.adaptvoronoi`), 320
- `AdaptiveVoronoiDriver` (class in `westpa.westext.adaptvoronoi.adaptVor_driver`), 320
- `add()` (`westpa.tools.selected_segs.AllSegmentSelection` method), 296
- `add()` (`westpa.tools.selected_segs.SegmentSelection` method), 296
- `add()` (`westpa.trajtree.trajtree.AllSegmentSelection` method), 300
- `add()` (`westpa.work_managers.core.FutureWatcher` method), 244
- `add_all_args()` (`westpa.tools.binning.WESTToolComponent` method), 283
- `add_all_args()` (`westpa.tools.core.WESTToolComponent` method), 285
- `add_all_args()` (`westpa.tools.data_reader.WESTToolComponent` method), 288
- `add_all_args()` (`westpa.tools.iter_range.WESTToolComponent` method), 290
- `add_all_args()` (`westpa.tools.progress.WESTToolComponent` method), 295
- `add_all_args()` (`westpa.tools.selected_segs.WESTToolComponent` method), 296
- `add_all_args()` (`westpa.tools.WESTToolComponent` method), 277
- `add_args()` (`westpa.cli.core.w_succ.WESTAnalysisTool` method), 58
- `add_args()` (`westpa.cli.core.w_succ.WESTDataReaderMixin` method), 58
- `add_args()` (`westpa.cli.tools.ploterr.CommonPloterrs` method), 117
- `add_args()` (`westpa.cli.tools.ploterr.DirectKinetics` method), 118
- `add_args()` (`westpa.cli.tools.ploterr.DirectStateprobs` method), 119
- `add_args()` (`westpa.cli.tools.ploterr.GenericIntervalSubcommand` method), 118
- `add_args()` (`westpa.cli.tools.ploterr.ProgressIndicatorComponent` method), 117
- `add_args()` (`westpa.cli.tools.ploterr.WESTMasterCommand` method), 116
- `add_args()` (`westpa.cli.tools.plothist.AveragePlotHist` method), 114
- `add_args()` (`westpa.cli.tools.plothist.EvolutionPlotHist` method), 115
- `add_args()` (`westpa.cli.tools.plothist.InstantPlotHist` method), 114
- `add_args()` (`westpa.cli.tools.plothist.PlotHistBase` method), 113
- `add_args()` (`westpa.cli.tools.plothist.PlotSupports2D` method), 113
- `add_args()` (`westpa.cli.tools.plothist.WESTMasterCommand` method), 112
- `add_args()` (`westpa.cli.tools.w_assign.BinMappingComponent` method), 30
- `add_args()` (`westpa.cli.tools.w_assign.ProgressIndicatorComponent` method), 31
- `add_args()` (`westpa.cli.tools.w_assign.WAssign` method), 35
- `add_args()` (`westpa.cli.tools.w_assign.WESTDataReader` method), 29

`add_args()` (`westpa.cli.tools.w_assign.WESTDSSynthesizer` method), 30

`add_args()` (`westpa.cli.tools.w_assign.WESTParallelTool` method), 29

`add_args()` (`westpa.cli.tools.w_bins.BinMappingComponent` method), 20

`add_args()` (`westpa.cli.tools.w_bins.WBinTool` method), 21

`add_args()` (`westpa.cli.tools.w_bins.WESTDataReader` method), 19

`add_args()` (`westpa.cli.tools.w_bins.WESTTool` method), 19

`add_args()` (`westpa.cli.tools.w_crawl.IterRangeSelection` method), 64

`add_args()` (`westpa.cli.tools.w_crawl.ProgressIndicatorComponent` method), 64

`add_args()` (`westpa.cli.tools.w_crawl.WCrawl` method), 65

`add_args()` (`westpa.cli.tools.w_crawl.WESTDataReader` method), 63

`add_args()` (`westpa.cli.tools.w_crawl.WESTParallelTool` method), 63

`add_args()` (`westpa.cli.tools.w_direct.AverageCommands` method), 70

`add_args()` (`westpa.cli.tools.w_direct.WESTKineticsBase` method), 69

`add_args()` (`westpa.cli.tools.w_direct.WESTMasterCommand` method), 68

`add_args()` (`westpa.cli.tools.w_direct.WESTParallelTool` method), 69

`add_args()` (`westpa.cli.tools.w_dumpsegs.WDumpSegs` method), 142

`add_args()` (`westpa.cli.tools.w_dumpsegs.WESTDataReader` method), 140

`add_args()` (`westpa.cli.tools.w_dumpsegs.WESTTool` method), 140

`add_args()` (`westpa.cli.tools.w_eddist.ProgressIndicatorComponent` method), 92

`add_args()` (`westpa.cli.tools.w_eddist.WEDDist` method), 94

`add_args()` (`westpa.cli.tools.w_eddist.WESTParallelTool` method), 91

`add_args()` (`westpa.cli.tools.w_fluxanl.IterRangeSelection` method), 161

`add_args()` (`westpa.cli.tools.w_fluxanl.WESTDataReader` method), 160

`add_args()` (`westpa.cli.tools.w_fluxanl.WESTTool` method), 160

`add_args()` (`westpa.cli.tools.w_fluxanl.WFluxanlTool` method), 162

`add_args()` (`westpa.cli.tools.w_ipa.ProgressIndicatorComponent` method), 46

`add_args()` (`westpa.cli.tools.w_ipa.WESTDataReader` method), 46

`add_args()` (`westpa.cli.tools.w_ipa.WESTParallelTool` method), 45

`add_args()` (`westpa.cli.tools.w_ipa.WIPI` method), 48

`add_args()` (`westpa.cli.tools.w_kinavg.WESTMasterCommand` method), 123

`add_args()` (`westpa.cli.tools.w_kinavg.WESTParallelTool` method), 124

`add_args()` (`westpa.cli.tools.w_kinetics.WESTMasterCommand` method), 129

`add_args()` (`westpa.cli.tools.w_kinetics.WESTParallelTool` method), 130

`add_args()` (`westpa.cli.tools.w_multi_west.ProgressIndicatorComponent` method), 102

`add_args()` (`westpa.cli.tools.w_multi_west.WESTMultiTool` method), 103

`add_args()` (`westpa.cli.tools.w_multi_west.WESTTool` method), 102

`add_args()` (`westpa.cli.tools.w_multi_west.WMultiWest` method), 103

`add_args()` (`westpa.cli.tools.w_ntop.IterRangeSelection` method), 97

`add_args()` (`westpa.cli.tools.w_ntop.ProgressIndicatorComponent` method), 98

`add_args()` (`westpa.cli.tools.w_ntop.WESTDataReader` method), 97

`add_args()` (`westpa.cli.tools.w_ntop.WESTTool` method), 96

`add_args()` (`westpa.cli.tools.w_ntop.WNTopTool` method), 99

`add_args()` (`westpa.cli.tools.w_pdist.IterRangeSelection` method), 53

`add_args()` (`westpa.cli.tools.w_pdist.ProgressIndicatorComponent` method), 54

`add_args()` (`westpa.cli.tools.w_pdist.WESTDataReader` method), 52

`add_args()` (`westpa.cli.tools.w_pdist.WESTDSSynthesizer` method), 52

`add_args()` (`westpa.cli.tools.w_pdist.WESTParallelTool` method), 52

`add_args()` (`westpa.cli.tools.w_pdist.WESTWDSSynthesizer` method), 53

`add_args()` (`westpa.cli.tools.w_pdist.WPDist` method), 56

`add_args()` (`westpa.cli.tools.w_postanalysis_matrix.RWMatrix` method), 144

`add_args()` (`westpa.cli.tools.w_postanalysis_matrix.WESTMasterCommand` method), 142

`add_args()` (`westpa.cli.tools.w_postanalysis_matrix.WESTParallelTool` method), 143

`add_args()` (`westpa.cli.tools.w_postanalysis_reweight.WESTMasterCommand` method), 145

`add_args()` (`westpa.cli.tools.w_postanalysis_reweight.WESTParallelTool` method), 146

`add_args()` (`westpa.cli.tools.w_red.WESTParallelTool` method), 146

method), 105

`add_args()` (`westpa.cli.tools.w_reweight.AverageCommand` method), 148

`add_args()` (`westpa.cli.tools.w_reweight.RWMatrix` method), 150

`add_args()` (`westpa.cli.tools.w_reweight.RWReweight` method), 150

`add_args()` (`westpa.cli.tools.w_reweight.WESTKineticsBase` method), 148

`add_args()` (`westpa.cli.tools.w_reweight.WESTMasterCommand` method), 147

`add_args()` (`westpa.cli.tools.w_reweight.WESTParallelTool` method), 148

`add_args()` (`westpa.cli.tools.w_select.IterRangeSelection` method), 79

`add_args()` (`westpa.cli.tools.w_select.ProgressIndicatorComponent` method), 80

`add_args()` (`westpa.cli.tools.w_select.WESTDataReader` method), 79

`add_args()` (`westpa.cli.tools.w_select.WESTParallelTool` method), 78

`add_args()` (`westpa.cli.tools.w_select.WSelectTool` method), 81

`add_args()` (`westpa.cli.tools.w_stateprobs.WESTMasterCommand` method), 136

`add_args()` (`westpa.cli.tools.w_stateprobs.WESTParallelTool` method), 136

`add_args()` (`westpa.cli.tools.w_trace.WESTDataReader` method), 38

`add_args()` (`westpa.cli.tools.w_trace.WESTTool` method), 37

`add_args()` (`westpa.cli.tools.w_trace.WTraceTool` method), 42

`add_args()` (`westpa.oldtools.iframe.AnalysisMixin` method), 301

`add_args()` (`westpa.oldtools.iframe.atool.WESTAnalysisTool` method), 307

`add_args()` (`westpa.oldtools.iframe.base_mixin.AnalysisMixin` method), 307

`add_args()` (`westpa.oldtools.iframe.BFDataManager` method), 304

`add_args()` (`westpa.oldtools.iframe.binning.AnalysisMixin` method), 308

`add_args()` (`westpa.oldtools.iframe.binning.BinningMixin` method), 308

`add_args()` (`westpa.oldtools.iframe.BinningMixin` method), 304

`add_args()` (`westpa.oldtools.iframe.data_reader.AnalysisMixin` method), 309

`add_args()` (`westpa.oldtools.iframe.data_reader.BFDataManager` method), 311

`add_args()` (`westpa.oldtools.iframe.data_reader.ExtDataReaderMixin` method), 311

`add_args()` (`westpa.oldtools.iframe.data_reader.WESTDataReaderMixin` method), 310

`add_args()` (`westpa.oldtools.iframe.ExtDataReaderMixin` method), 303

`add_args()` (`westpa.oldtools.iframe.iter_range.AnalysisMixin` method), 312

`add_args()` (`westpa.oldtools.iframe.iter_range.IterRangeMixin` method), 312

`add_args()` (`westpa.oldtools.iframe.IterRangeMixin` method), 301

`add_args()` (`westpa.oldtools.iframe.kinetics.AnalysisMixin` method), 313

`add_args()` (`westpa.oldtools.iframe.kinetics.KineticsAnalysisMixin` method), 313

`add_args()` (`westpa.oldtools.iframe.KineticsAnalysisMixin` method), 306

`add_args()` (`westpa.oldtools.iframe.mcbs.AnalysisMixin` method), 313

`add_args()` (`westpa.oldtools.iframe.mcbs.MCBSMixin` method), 313

`add_args()` (`westpa.oldtools.iframe.MCBSMixin` method), 305

`add_args()` (`westpa.oldtools.iframe.output.AnalysisMixin` method), 314

`add_args()` (`westpa.oldtools.iframe.plotting.AnalysisMixin` method), 315

`add_args()` (`westpa.oldtools.iframe.TransitionAnalysisMixin` method), 305

`add_args()` (`westpa.oldtools.iframe.transitions.AnalysisMixin` method), 315

`add_args()` (`westpa.oldtools.iframe.transitions.TransitionAnalysisMixin` method), 317

`add_args()` (`westpa.oldtools.iframe.WESTAnalysisTool` method), 301

`add_args()` (`westpa.oldtools.iframe.WESTDataReaderMixin` method), 302

`add_args()` (`westpa.tools.BinMappingComponent` method), 281

`add_args()` (`westpa.tools.binning.BinMappingComponent` method), 285

`add_args()` (`westpa.tools.binning.WESTToolComponent` method), 283

`add_args()` (`westpa.tools.core.WESTMasterCommand` method), 288

`add_args()` (`westpa.tools.core.WESTMultiTool` method), 287

`add_args()` (`westpa.tools.core.WESTParallelTool` method), 286

`add_args()` (`westpa.tools.core.WESTTool` method), 286

`add_args()` (`westpa.tools.core.WESTToolComponent` method), 285

`add_args()` (`westpa.tools.data_reader.WESTDataReader` method), 289

`add_args()` (`westpa.tools.data_reader.WESTDSSynthesizer` method), 289

`add_args()` (*westpa.tools.data_reader.WESTToolComponent* method), 288

`add_args()` (*westpa.tools.data_reader.WESTWDSSynthesizer* method), 289

`add_args()` (*westpa.tools.iter_range.IterRangeSelection* method), 291

`add_args()` (*westpa.tools.iter_range.WESTToolComponent* method), 290

`add_args()` (*westpa.tools.IterRangeSelection* method), 279

`add_args()` (*westpa.tools.kinetics_tool.AverageCommands* method), 294

`add_args()` (*westpa.tools.kinetics_tool.IterRangeSelection* method), 292

`add_args()` (*westpa.tools.kinetics_tool.ProgressIndicatorComponent* method), 293

`add_args()` (*westpa.tools.kinetics_tool.WESTDataReader* method), 292

`add_args()` (*westpa.tools.kinetics_tool.WESTKineticsBase* method), 293

`add_args()` (*westpa.tools.progress.ProgressIndicatorComponent* method), 295

`add_args()` (*westpa.tools.progress.WESTToolComponent* method), 295

`add_args()` (*westpa.tools.ProgressIndicatorComponent* method), 281

`add_args()` (*westpa.tools.SegSelector* method), 280

`add_args()` (*westpa.tools.selected_segs.SegSelector* method), 296

`add_args()` (*westpa.tools.selected_segs.WESTToolComponent* method), 295

`add_args()` (*westpa.tools.WESTDataReader* method), 278

`add_args()` (*westpa.tools.WESTDSSynthesizer* method), 279

`add_args()` (*westpa.tools.WESTMasterCommand* method), 278

`add_args()` (*westpa.tools.WESTMultiTool* method), 278

`add_args()` (*westpa.tools.WESTParallelTool* method), 276

`add_args()` (*westpa.tools.WESTTool* method), 276

`add_args()` (*westpa.tools.WESTToolComponent* method), 277

`add_args()` (*westpa.tools.WESTWDSSynthesizer* method), 279

`add_common_output_args()` (*westpa.cli.core.w_succ.CommonOutputMixin* method), 59

`add_common_output_args()` (*westpa.oldtools.iframe.CommonOutputMixin* method), 307

`add_common_output_args()` (*westpa.oldtools.iframe.output.CommonOutputMixin* method), 314

`add_initial_states()` (*westpa.core.binning.mab_driver.WEDriver* method), 169

`add_initial_states()` (*westpa.core.we_driver.WEDriver* method), 238

`add_mapper()` (*westpa.core.binning.assign.RecursiveBinMapper* method), 167

`add_mapper()` (*westpa.core.binning.RecursiveBinMapper* method), 163

`add_mcbs_options()` (in module *westpa.oldtools.stats.mcbs*), 319

`add_target_count_args()` (*westpa.cli.tools.w_assign.BinMappingComponent* method), 30

`add_target_count_args()` (*westpa.cli.tools.w_bins.BinMappingComponent* method), 20

`add_target_count_args()` (*westpa.tools.BinMappingComponent* method), 281

`add_target_count_args()` (*westpa.tools.binning.BinMappingComponent* method), 285

`add_timer()` (*westpa.work_managers.zeromq.core.PassiveMultiTimer* method), 260

`add_timer()` (*westpa.work_managers.zeromq.node.PassiveMultiTimer* method), 265

`add_timer()` (*westpa.work_managers.zeromq.work_manager.PassiveMultiTimer* method), 268

`add_timer()` (*westpa.work_managers.zeromq.worker.PassiveMultiTimer* method), 275

`add_to_subparsers()` (*westpa.cli.tools.ploterr.WESTSubcommand* method), 117

`add_to_subparsers()` (*westpa.cli.tools.plothist.WESTSubcommand* method), 113

`add_to_subparsers()` (*westpa.tools.core.WESTSubcommand* method), 287

`add_to_subparsers()` (*westpa.tools.kinetics_tool.WESTSubcommand* method), 293

`add_to_subparsers()` (*westpa.tools.WESTSubcommand* method), 277

`add_wm_args()` (in module *westpa.work_managers.environment*), 245

`add_wm_args()` (*westpa.work_managers.core.WorkManager* class method), 243

`add_wm_args()` (*westpa.work_managers.environment.WMEnvironment* method), 245

`add_wm_args()` (*westpa.work_managers.mpi.WorkManager* method), 245

- class method*), 247
- `add_wm_args()` (*westpa.work_managers.processes.WorkManager* *class method*), 249
- `add_wm_args()` (*westpa.work_managers.serial.WorkManager* *class method*), 252
- `add_wm_args()` (*westpa.work_managers.threads.WorkManager* *class method*), 254
- `add_wm_args()` (*westpa.work_managers.zeromq.work_manager.WorkManager* *class method*), 269
- `add_wm_args()` (*westpa.work_managers.zeromq.work_manager.ZMQWorkManager* *class method*), 271
- `add_wm_args()` (*westpa.work_managers.zeromq.ZMQWorkManager* *class method*), 258
- `alchemicalLambda` (*westpa.core.h5io.Frames* *attribute*), 215
- `all_acquired()` (*westpa.work_managers.core.WMFuture* *static method*), 244
- `all_acquired()` (*westpa.work_managers.mpi.WMFuture* *static method*), 248
- `all_acquired()` (*westpa.work_managers.processes.WMFuture* *static method*), 250
- `all_acquired()` (*westpa.work_managers.serial.WMFuture* *static method*), 253
- `all_acquired()` (*westpa.work_managers.threads.WMFuture* *static method*), 255
- `all_acquired()` (*westpa.work_managers.zeromq.work_manager.WMFuture* *static method*), 270
- `all_initial_states` (*westpa.core.binning.mab_driver.WEDriver* *property*), 169
- `all_initial_states` (*westpa.core.we_driver.WEDriver* *property*), 238
- `AllSegmentSelection` (*class in westpa.tools.selected_segs*), 296
- `AllSegmentSelection` (*class in westpa.trajtree.trajtree*), 300
- `analysis_structure()` (*westpa.cli.tools.w_ipa.WIPI* *method*), 48
- `AnalysisMixin` (*class in westpa.oldtools.iframe*), 301
- `AnalysisMixin` (*class in westpa.oldtools.iframe.base_mixin*), 307
- `AnalysisMixin` (*class in westpa.oldtools.iframe.binning*), 308
- `AnalysisMixin` (*class in westpa.oldtools.iframe.data_reader*), 309
- `AnalysisMixin` (*class in westpa.oldtools.iframe.iter_range*), 312
- `AnalysisMixin` (*class in westpa.oldtools.iframe.kinetics*), 313
- `AnalysisMixin` (*class in westpa.oldtools.iframe.mcbs*), 313
- `AnalysisMixin` (*class in westpa.oldtools.iframe.output*), 314
- `AnalysisMixin` (*class in westpa.oldtools.iframe.plotting*), 315
- `AnalysisMixin` (*class in westpa.oldtools.iframe.transitions*), 315
- `append()` (*westpa.work_managers.mpi.deque* *method*), 246
- `append()` (*westpa.work_managers.zeromq.work_manager.deque* *method*), 271
- `appendleft()` (*westpa.work_managers.mpi.deque* *method*), 246
- `appendleft()` (*westpa.work_managers.zeromq.work_manager.deque* *method*), 271
- `application` (*westpa.core.h5io.HDF5TrajectoryFile* *attribute*), 211
- `application` (*westpa.core.h5io.HDF5TrajectoryFile* *property*), 212
- `apply_down()` (*in module westpa.core.binning.assign*), 166
- `apply_down_argmin_across()` (*in module westpa.core.binning.assign*), 166
- `arg_flag()` (*westpa.work_managers.environment.WMEnvironment* *method*), 245
- `arg_name()` (*westpa.work_managers.environment.WMEnvironment* *method*), 245
- `arg_prefix` (*westpa.work_managers.environment.WMEnvironment* *attribute*), 245
- `ArgumentError`, 301, 307, 312
- `as_completed()` (*westpa.oldtools.stats.edfs.EDF* *method*), 318
- `as_completed()` (*westpa.work_managers.core.WorkManager* *method*), 243
- `as_completed()` (*westpa.work_managers.mpi.WorkManager* *method*), 247
- `as_completed()` (*westpa.work_managers.processes.WorkManager* *method*), 250
- `as_completed()` (*westpa.work_managers.serial.WorkManager* *method*), 252
- `as_completed()` (*westpa.work_managers.threads.WorkManager* *method*), 254
- `as_completed()` (*westpa.work_managers.zeromq.work_manager.WorkManager* *method*), 269
- `as_numpy_record()` (*westpa.cli.core.w_fork.InitialState* *method*), 26
- `as_numpy_record()` (*westpa.cli.core.w_init.BasisState* *method*), 15
- `as_numpy_record()` (*westpa.cli.core.w_states.BasisState* *method*), 86
- `as_numpy_record()` (*westpa.cli.tools.w_trace.InitialState* *method*), 40
- `as_numpy_record()` (*westpa.core.binning.mab_manager.InitialState* *method*), 172
- `as_numpy_record()` (*westpa.core.data_manager.BasisState* *method*), 190
- `as_numpy_record()` (*westpa.core.data_manager.InitialState* *method*), 192
- `as_numpy_record()` (*westpa.core.propagators.executable.BasisState* *method*), 192

`method`), 182
`as_numpy_record()` (`westpa.core.propagators.executable.InitialState` method), 183
`as_numpy_record()` (`westpa.core.sim_manager.InitialState` method), 228
`as_numpy_record()` (`westpa.core.states.BasisState` method), 231
`as_numpy_record()` (`westpa.core.states.InitialState` method), 232
`as_numpy_record()` (`westpa.core.we_driver.InitialState` method), 236
`assign` (`westpa.cli.tools.w_ipa.WIPI` property), 48
`assign` (`westpa.cli.tools.w_ipa.WIPIScheme` property), 47
`assign` (`westpa.tools.wipi.WIPIScheme` property), 297
`assign` (`westpa.tools.WIPIScheme` property), 282
`assign()` (`westpa.core.binning.assign.FuncBinMapper` method), 167
`assign()` (`westpa.core.binning.assign.NopMapper` method), 166
`assign()` (`westpa.core.binning.assign.PiecewiseBinMapper` method), 166
`assign()` (`westpa.core.binning.assign.RectilinearBinMapper` method), 166
`assign()` (`westpa.core.binning.assign.RecursiveBinMapper` method), 167
`assign()` (`westpa.core.binning.assign.VectorizingFuncBinMapper` method), 167
`assign()` (`westpa.core.binning.assign.VoronoiBinMapper` method), 167
`assign()` (`westpa.core.binning.BinlessDriver` method), 164
`assign()` (`westpa.core.binning.FuncBinMapper` method), 162
`assign()` (`westpa.core.binning.mab.FuncBinMapper` method), 168
`assign()` (`westpa.core.binning.mab_driver.MABDriver` method), 170
`assign()` (`westpa.core.binning.mab_driver.WEDriver` method), 169
`assign()` (`westpa.core.binning.MABDriver` method), 164
`assign()` (`westpa.core.binning.NopMapper` method), 162
`assign()` (`westpa.core.binning.PiecewiseBinMapper` method), 162
`assign()` (`westpa.core.binning.RectilinearBinMapper` method), 163
`assign()` (`westpa.core.binning.RecursiveBinMapper` method), 163
`assign()` (`westpa.core.binning.VectorizingFuncBinMapper` method), 163
`assign()` (`westpa.core.binning.VoronoiBinMapper` method), 163
`assign()` (`westpa.core.systems.NopMapper` method), 233
`assign()` (`westpa.core.we_driver.WEDriver` method), 238
`assign()` (`westpa.core.yamlcfg.NopMapper` method), 239
`assign()` (`westpa.tools.binning.RectilinearBinMapper` method), 283
`assign()` (`westpa.westext.adaptvoronoi.adaptVor_driver.VoronoiBinMapper` method), 319
`assign_and_label()` (in module `westpa.cli.tools.w_assign`), 28
`assign_and_label()` (in module `westpa.core.binning`), 164
`assign_iteration()` (`westpa.cli.tools.w_assign.WAssign` method), 35
`assign_to_bins()` (`westpa.oldtools.iframe.binning.BinningMixin` method), 308
`assign_to_bins()` (`westpa.oldtools.iframe.BinningMixin` method), 305
`assignments_list_to_table()` (in module `westpa.cli.tools.w_ntop`), 98
`assignments_list_to_table()` (in module `westpa.core.binning`), 164
`atom_slice()` (`westpa.core.h5io.Trajectory` method), 207
`AtomMapper` (class in `westpa.core.data_manager`), 188
`aux_data_loader()` (in module `westpa.core.propagators.executable`), 185
`auxiliary_data` (`westpa.analysis.core.Iteration` property), 334
`auxiliary_data` (`westpa.analysis.core.Walker` property), 337
`average()` (`westpa.oldtools.stats.accumulator.RunningStatsAccumulator` method), 318
`average()` (`westpa.oldtools.stats.RunningStatsAccumulator` method), 317
`AverageCommands` (class in `westpa.cli.tools.w_direct`), 69
`AverageCommands` (class in `westpa.cli.tools.w_reweight`), 148
`AverageCommands` (class in `westpa.tools.kinetics_tool`), 294
`AveragePlotHist` (class in `westpa.cli.tools.plothist`), 114

B

`BasicMDTrajectory` (class in `westpa.analysis.trajectories`), 339
`basis_state()` (`westpa.analysis.core.Iteration` method), 335
`basis_state_pcoords` (`westpa.analysis.core.Iteration` property), 334
`basis_state_summaries`

(*westpa.analysis.core.Iteration* property), 334

basis_states (*westpa.analysis.core.Iteration* property), 335

BasisState (class in *westpa.cli.core.w_init*), 15

BasisState (class in *westpa.cli.core.w_states*), 85

BasisState (class in *westpa.core.data_manager*), 190

BasisState (class in *westpa.core.propagators.executable*), 181

BasisState (class in *westpa.core.states*), 231

BFDDataManager (class in *westpa.oldtools.iframe*), 303

BFDDataManager (class in *westpa.oldtools.iframe.data_reader*), 311

BFTransitionAnalysisMixin (class in *westpa.oldtools.iframe*), 306

BFTransitionAnalysisMixin (class in *westpa.oldtools.iframe.transitions*), 317

Bin (class in *westpa.analysis.core*), 338

Bin (class in *westpa.core.binning*), 165

Bin (class in *westpa.core.binning.assign*), 165

Bin (class in *westpa.core.binning.bins*), 167

bin() (*westpa.analysis.core.Iteration* method), 335

bin_labels (*westpa.cli.tools.w_ipa.WIPI* property), 48

bin_labels (*westpa.cli.tools.w_ipa.WIPIScheme* property), 47

bin_labels (*westpa.tools.wipi.WIPIScheme* property), 297

bin_labels (*westpa.tools.WIPIScheme* property), 282

bin_mapper (*westpa.analysis.core.Iteration* property), 334

bin_target_counts (*westpa.analysis.core.Iteration* property), 333

bin_target_counts (*westpa.core.systems.WESTSystem* property), 233

bin_target_counts (*westpa.core.yamlcfg.YAMLSystem* property), 240

BinlessDriver (class in *westpa.core.binning*), 164

BinlessMapper (class in *westpa.core.binning*), 163

BinlessSimManager (class in *westpa.core.binning*), 164

BinMapper (class in *westpa.core.binning.assign*), 166

BinMappingComponent (class in *westpa.cli.tools.w_assign*), 30

BinMappingComponent (class in *westpa.cli.tools.w_bins*), 20

BinMappingComponent (class in *westpa.tools*), 280

BinMappingComponent (class in *westpa.tools.binning*), 284

binning_hchunksize (*westpa.core.data_manager.WESTDataManager* attribute), 195

BinningMixin (class in *westpa.oldtools.iframe*), 304

BinningMixin (class in *westpa.oldtools.iframe.binning*), 308

bins (*westpa.analysis.core.Iteration* property), 334

bins_from_yaml_dict() (in module *westpa.westext.weed.weed_driver*), 327

bins_from_yaml_dict() (in module *westpa.westext.wess.wess_driver*), 328

BinUnion (class in *westpa.analysis.core*), 337

blocked_iter() (in module *westpa.core.propagators*), 179

bootstrap_ci() (in module *westpa.oldtools.iframe.mcbs*), 314

bootstrap_ci() (in module *westpa.oldtools.stats.mcbs*), 319

bootstrap_ci_ll() (in module *westpa.oldtools.iframe.mcbs*), 314

boundaries (*westpa.core.binning.assign.RectilinearBinMapper* property), 166

boundaries (*westpa.core.binning.RectilinearBinMapper* property), 163

boundaries (*westpa.tools.binning.RectilinearBinMapper* property), 283

BytesIO (class in *westpa.core.propagators.executable*), 180

C

cache_data() (*westpa.core.h5io.IterBlockedDataset* method), 221

cache_pcoords (*westpa.cli.core.w_succ.WESTDataReaderMixin* property), 58

cache_pcoords (*westpa.oldtools.iframe.data_reader.WESTDataReaderMixin* property), 310

cache_pcoords (*westpa.oldtools.iframe.WESTDataReaderMixin* property), 302

calc_avg_rate() (in module *westpa.cli.tools.w_red*), 106

calc_chunksize() (in module *westpa.core.data_manager*), 197

calc_chunksize() (in module *westpa.core.h5io*), 216

calc_ci_bound_indices() (in module *westpa.oldtools.iframe.mcbs*), 314

calc_ci_bound_indices() (*westpa.oldtools.iframe.mcbs.MCBSMixin* method), 314

calc_ci_bound_indices() (*westpa.oldtools.iframe.MCBSMixin* method), 305

calc_evol_flux() (*westpa.cli.tools.w_fluxanl.WFluxanlTool* method), 162

calc_mcbs_nsets() (in module *westpa.oldtools.iframe.mcbs*), 314

calc_mcbs_nsets() (*westpa.oldtools.iframe.mcbs.MCBSMixin* method), 314

calc_mcbs_nsets() (*westpa.oldtools.iframe.MCBSMixin* method), 305

calc_rate() (*westpa.cli.tools.w_red.RateCalculator* method), 106

calc_rates() (in module *westpa.cli.tools.w_red*), 106

`calc_rates()` (in module `westpa.core.kinetics.rate_averaging`), 177
`calc_rates()` (`westpa.cli.tools.w_red.RateCalculator` method), 106
`calc_stats()` (in module `westpa.core.reweight.matrix`), 187
`calc_store_flux_data()` (`westpa.cli.tools.w_fluxanl.WFluxanlTool` method), 162
`calculate()` (`westpa.core.kinetics.rate_averaging.RateAverager` method), 179
`calculate()` (`westpa.core.kinetics.RateAverager` method), 174
`calculate()` (`westpa.westext.weed.weed_driver.RateAverager` method), 327
`calculate()` (`westpa.westext.wess.wess_driver.RateAverager` method), 328
`calculate_labeled_fluxes()` (in module `westpa.core.kinetics`), 174
`calculate_labeled_fluxes()` (in module `westpa.core.kinetics.matrates`), 176
`calculate_labeled_fluxes_alllags()` (in module `westpa.core.kinetics`), 174
`calculate_labeled_fluxes_alllags()` (in module `westpa.core.kinetics.matrates`), 176
`calculate_state_populations()` (`westpa.cli.tools.w_direct.DStateProbs` method), 73
`calculate_state_populations()` (`westpa.cli.tools.w_stateprobs.DStateProbs` method), 138
`canonicalize_endpoint()` (`westpa.work_managers.zeromq.work_manager.ZMQWorkManager` class method), 272
`canonicalize_endpoint()` (`westpa.work_managers.zeromq.ZMQWorkManager` class method), 258
`cell_angles` (`westpa.core.h5io.Frames` attribute), 215
`cell_lengths` (`westpa.core.h5io.Frames` attribute), 215
`center_coordinates()` (`westpa.core.h5io.Trajectory` method), 207
`change_duration()` (`westpa.work_managers.zeromq.core.ActiveMultiTimer` method), 261
`change_duration()` (`westpa.work_managers.zeromq.node.PassiveMultiTimer` method), 265
`change_duration()` (`westpa.work_managers.zeromq.work_manager.PassiveMultiTimer` method), 268
`change_duration()` (`westpa.work_managers.zeromq.work_manager.ActiveMultiTimer` method), 275
`check_bin_selection()` (`westpa.oldtools.aframe.kinetics.KineticsAnalysisMixIn` method), 313
`check_bin_selection()` (`westpa.oldtools.aframe.KineticsAnalysisMixIn` method), 313
`check_bool()` (in module `westpa.core.propagators.executable`), 184
`check_bool()` (in module `westpa.core.yamlcfg`), 239
`check_bool()` (in module `westpa.westext.adaptvoronoi.adaptVor_driver`), 319
`check_bool()` (in module `westpa.westext.weed.weed_driver`), 327
`check_bool()` (in module `westpa.westext.wess.wess_driver`), 328
`check_data_binhash()` (`westpa.oldtools.aframe.binning.BinningMixIn` method), 308
`check_data_binhash()` (`westpa.oldtools.aframe.BinningMixIn` method), 304
`check_data_iter_range_equal()` (`westpa.cli.tools.w_crawl.IterRangeSelection` method), 64
`check_data_iter_range_equal()` (`westpa.cli.tools.w_fluxanl.IterRangeSelection` method), 161
`check_data_iter_range_equal()` (`westpa.cli.tools.w_ntop.IterRangeSelection` method), 98
`check_data_iter_range_equal()` (`westpa.cli.tools.w_pdist.IterRangeSelection` method), 53
`check_data_iter_range_equal()` (`westpa.cli.tools.w_select.IterRangeSelection` method), 80
`check_data_iter_range_equal()` (`westpa.oldtools.aframe.iter_range.IterRangeMixIn` method), 312
`check_data_iter_range_equal()` (`westpa.oldtools.aframe.IterRangeMixIn` method), 302
`check_data_iter_range_equal()` (`westpa.tools.iter_range.IterRangeSelection` method), 291
`check_data_iter_range_equal()` (`westpa.tools.IterRangeSelection` method), 286
`check_data_iter_range_equal()` (`westpa.tools.iter_range.IterRangeSelection` method), 293
`check_data_iter_range_least()` (`westpa.cli.tools.w_crawl.IterRangeSelection` method), 64
`check_data_iter_range_least()` (`westpa.cli.tools.w_fluxanl.IterRangeSelection` method), 161
`check_data_iter_range_least()` (`westpa.tools.iter_range.IterRangeSelection` method), 291

(westpa.cli.tools.w_ntop.IterRangeSelection method), 97
 check_data_iter_range_least()
(westpa.cli.tools.w_pdist.IterRangeSelection method), 53
 check_data_iter_range_least()
(westpa.cli.tools.w_select.IterRangeSelection method), 80
 check_data_iter_range_least()
(westpa.oldtools.iframe.iter_range.IterRangeMixin method), 312
 check_data_iter_range_least()
(westpa.oldtools.iframe.IterRangeMixin method), 302
 check_data_iter_range_least()
(westpa.tools.iter_range.IterRangeSelection method), 291
 check_data_iter_range_least()
(westpa.tools.IterRangeSelection method), 280
 check_data_iter_range_least()
(westpa.tools.kinetics_tool.IterRangeSelection method), 292
 check_data_iter_step_conformant()
(westpa.cli.tools.w_crawl.IterRangeSelection method), 64
 check_data_iter_step_conformant()
(westpa.cli.tools.w_fluxanl.IterRangeSelection method), 161
 check_data_iter_step_conformant()
(westpa.cli.tools.w_ntop.IterRangeSelection method), 98
 check_data_iter_step_conformant()
(westpa.oldtools.iframe.iter_range.IterRangeMixin method), 312
 check_data_iter_step_conformant()
(westpa.oldtools.iframe.IterRangeMixin method), 301
 check_data_iter_step_conformant()
(westpa.cli.tools.w_pdist.IterRangeSelection method), 53
 check_data_iter_step_conformant()
(westpa.cli.tools.w_select.IterRangeSelection method), 80
 check_data_iter_step_conformant()
(westpa.oldtools.iframe.iter_range.IterRangeMixin method), 312
 check_data_iter_step_conformant()
(westpa.oldtools.iframe.IterRangeMixin method), 302
 check_data_iter_step_conformant()
(westpa.tools.iter_range.IterRangeSelection method), 291
 check_data_iter_step_conformant()
(westpa.tools.IterRangeSelection method), 280
 check_data_iter_step_conformant()
(westpa.tools.kinetics_tool.IterRangeSelection method), 293
 check_data_iter_step_equal()
(westpa.cli.tools.w_crawl.IterRangeSelection method), 64
 check_data_iter_step_equal()
(westpa.cli.tools.w_fluxanl.IterRangeSelection method), 161
 check_data_iter_step_equal()
(westpa.cli.tools.w_ntop.IterRangeSelection method), 98
 check_data_iter_step_equal()
(westpa.cli.tools.w_pdist.IterRangeSelection method), 54
 check_data_iter_step_equal()
(westpa.cli.tools.w_select.IterRangeSelection method), 80
 check_data_iter_step_equal()
(westpa.oldtools.iframe.iter_range.IterRangeMixin method), 313
 check_data_iter_step_equal()
(westpa.oldtools.iframe.IterRangeMixin method), 302
 check_data_iter_step_equal()
(westpa.tools.iter_range.IterRangeSelection method), 291
 check_data_iter_step_equal()
(westpa.tools.IterRangeSelection method), 280
 check_data_iter_step_equal()
(westpa.tools.kinetics_tool.IterRangeSelection method), 293
 check_iter_range()
(westpa.oldtools.iframe.iter_range.IterRangeMixin method), 312
 check_iter_range()
(westpa.oldtools.iframe.IterRangeMixin method), 301
 check_iter_range_equal() (in module *westpa.core.h5io*), 217
 check_iter_range_least() (in module *westpa.core.h5io*), 217
 check_propagation()
(westpa.core.binning.mab_manager.WESimManager method), 171
 check_propagation()
(westpa.core.sim_manager.WESimManager method), 229
 check_threshold_configs()
(westpa.core.binning.mab_driver.WEDriver method), 169
 check_threshold_configs()
(westpa.core.we_driver.WEDriver method), 237
 check_workers() (*westpa.work_managers.zeromq.work_manager.ZMQW*
method), 272
 check_workers() (*westpa.work_managers.zeromq.ZMQWorkManager*
method), 259
 children (*westpa.analysis.core.Walker* property), 336

`clear()` (`westpa.core.binning.mab_driver.WEDriver` method), 169

`clear()` (`westpa.core.progress.ProgressIndicator` method), 224

`clear()` (`westpa.core.we_driver.WEDriver` method), 237

`clear()` (`westpa.oldtools.iframe.TransitionEventAccumulator` method), 306

`clear()` (`westpa.oldtools.iframe.transitions.TransitionEventAccumulator` method), 316

`clear()` (`westpa.tools.progress.ProgressIndicator` method), 294

`clear()` (`westpa.work_managers.mpi.deque` method), 246

`clear()` (`westpa.work_managers.zeromq.work_manager.deque` method), 271

`clear_basis_initial_states()` (`westpa.core.propagators.executable.WESTPropagator` method), 181

`clear_basis_initial_states()` (`westpa.core.propagators.WESTPropagator` method), 179

`clear_run_cache()` (`westpa.cli.core.w_succ.WESTDataReaderMixin` method), 58

`clear_run_cache()` (`westpa.oldtools.iframe.data_reader.WESTDataReaderMixin` method), 310

`clear_run_cache()` (`westpa.oldtools.iframe.WESTDataReaderMixin` method), 302

`clear_state()` (`westpa.oldtools.iframe.TransitionEventAccumulator` method), 306

`clear_state()` (`westpa.oldtools.iframe.transitions.TransitionEventAccumulator` method), 316

`clockIn()` (`westpa.work_managers.mpi.Worker` method), 249

`close()` (`westpa.analysis.core.Run` method), 332

`close()` (`westpa.cli.tools.w_assign.WESTDataReader` method), 29

`close()` (`westpa.cli.tools.w_bins.WESTDataReader` method), 19

`close()` (`westpa.cli.tools.w_crawl.WESTDataReader` method), 63

`close()` (`westpa.cli.tools.w_dumpsegs.WESTDataReader` method), 141

`close()` (`westpa.cli.tools.w_fluxanl.WESTDataReader` method), 160

`close()` (`westpa.cli.tools.w_ipa.WESTDataReader` method), 46

`close()` (`westpa.cli.tools.w_ntop.WESTDataReader` method), 97

`close()` (`westpa.cli.tools.w_pdist.WESTDataReader` method), 52

`close()` (`westpa.cli.tools.w_select.WESTDataReader` method), 79

`close()` (`westpa.cli.tools.w_trace.WESTDataReader` method), 38

`close()` (`westpa.core.h5io.HDF5TrajectoryFile` method), 214

`close()` (`westpa.core.propagators.executable.BytesIO` method), 180

`close()` (`westpa.core.textio.NumericTextOutputFormatter` method), 234

`close()` (`westpa.tools.data_reader.WESTDataReader` method), 289

`close()` (`westpa.tools.kinetics_tool.WESTDataReader` method), 292

`close()` (`westpa.tools.WESTDataReader` method), 279

`close_analysis_backing()` (`westpa.cli.core.w_succ.WESTAnalysisTool` method), 58

`close_analysis_backing()` (`westpa.oldtools.iframe.atool.WESTAnalysisTool` method), 307

`close_analysis_backing()` (`westpa.oldtools.iframe.WESTAnalysisTool` method), 301

`close_backing()` (`westpa.core.data_manager.WESTDataManager` method), 195

`close_bf_h5file()` (`westpa.oldtools.iframe.BFDataManager` method), 312

`close_bf_h5file()` (`westpa.oldtools.iframe.data_reader.BFDataManager` method), 312

`closed` (`westpa.analysis.core.Run` property), 332

`closed` (`westpa.core.data_manager.WESTDataManager` property), 195

`closed` (`westpa.core.propagators.executable.BytesIO` attribute), 180

`ClusterList` (class in `westpa.westext.weed.BinCluster`), 326

`cmd_info()` (`westpa.cli.tools.w_bins.WBinTool` method), 21

`cmd_rebin()` (`westpa.cli.tools.w_bins.WBinTool` method), 21

`cmoment()` (`westpa.oldtools.stats.edfs.EDF` method), 318

`coalesce_announcements()` (`westpa.work_managers.zeromq.core.Message` class method), 260

`coalesce_announcements()` (`westpa.work_managers.zeromq.node.Message` class method), 265

`coalesce_announcements()` (`westpa.work_managers.zeromq.work_manager.Message` class method), 267

`coalesce_announcements()` (`westpa.work_managers.zeromq.worker.Message` class method), 274

`coerce_type_if_present()` (`westpa.core.yamlcfg.YAMLConfig` method), 239

`color_output_filename`

(*westpa.cli.tools.ploterr.DirectStateprobs* attribute), 119

color_output_filename (*westpa.cli.tools.ploterr.ReweightStateprobs* attribute), 119

comm_loop() (*westpa.work_managers.zeromq.node.ZMQNode* method), 265

comm_loop() (*westpa.work_managers.zeromq.work_manager.ZMQNode* method), 269

comm_loop() (*westpa.work_managers.zeromq.work_manager.ZMQWorker* method), 268

comm_loop() (*westpa.work_managers.zeromq.work_manager.ZMQWorkerManager* method), 272

comm_loop() (*westpa.work_managers.zeromq.worker.ZMQExecutor* method), 276

comm_loop() (*westpa.work_managers.zeromq.worker.ZMQWorker* method), 275

comm_loop() (*westpa.work_managers.zeromq.ZMQNode* method), 258

comm_loop() (*westpa.work_managers.zeromq.ZMQWorker* method), 258

comm_loop() (*westpa.work_managers.zeromq.ZMQWorkerManager* method), 259

comment_string (*westpa.core.textio.NumericTextOutputFormat* attribute), 234

CommonOutputMixin (class in *westpa.cli.core.w_succ*), 59

CommonOutputMixin (class in *westpa.oldtools.iframe*), 307

CommonOutputMixin (class in *westpa.oldtools.iframe.output*), 314

CommonPloterrs (class in *westpa.cli.tools.ploterr*), 117

concatenate() (in module *westpa.analysis.trajectories*), 339

concatenate() (*westpa.westext.weed.UncertMath.UncertMath* method), 326

conditional_fluxes (*westpa.cli.tools.w_red.RateCalculator* property), 106

ConfigItemMissing, 239, 319

ConfigItemTypeError, 239

ConfigValueError, 239

ConfigValueWarning, 239

ConsistencyError, 236

ConsistencyWarning, 176

constraints (*westpa.core.h5io.HDF5TrajectoryFile* attribute), 212

constraints (*westpa.core.h5io.HDF5TrajectoryFile* property), 212

construct_bins() (*westpa.cli.tools.w_eddist.WEDDist* method), 94

construct_bins() (*westpa.cli.tools.w_pdist.WPDist* method), 56

construct_bins() (*westpa.core.binning.assign.BinMapper* method), 166

construct_histogram() (*westpa.cli.tools.w_eddist.WEDDist* method), 94

construct_histogram() (*westpa.cli.tools.w_pdist.WPDist* method), 56

construct_next() (*westpa.core.binning.mab_driver.WEDriver* method), 170

construct_next() (*westpa.core.wed_driver.WEDriver* method), 238

contextmanager() (in module *westpa.work_managers.core*), 242

continue_accumulation() (*westpa.oldtools.iframe.TransitionEventAccumulator* method), 306

continue_accumulation() (*westpa.oldtools.iframe.transitions.TransitionEventAccumulator* method), 316

coord_dtype (in module *westpa.core.binning*), 164

coord_dtype (in module *westpa.core.binning.assign*), 166

coordinates (*westpa.core.h5io.Frames* attribute), 215

copy() (*westpa.work_managers.mpi.deque* method), 246

copy() (*westpa.work_managers.zeromq.work_manager.deque* method), 271

correction() (*westpa.cli.tools.w_red.DurationCorrector* method), 105

count (class in *westpa.tools.binning*), 283

count() (*westpa.work_managers.mpi.deque* method), 246

count() (*westpa.work_managers.zeromq.work_manager.deque* method), 271

count_dtype (*westpa.oldtools.iframe.TransitionEventAccumulator* attribute), 306

count_dtype (*westpa.oldtools.iframe.transitions.TransitionEventAccumulator* attribute), 316

create_dataset_from_dsopts() (in module *westpa.core.data_manager*), 197

create_hdf5_group() (in module *westpa.core.h5io*), 216

create_ibstate_group() (*westpa.core.data_manager.WESTDataManager* method), 195

create_ibstate_iter_h5file() (*westpa.core.data_manager.WESTDataManager* method), 196

create_idtype_array() (in module *westpa.cli.tools.w_multi_west*), 103

create_initial_states() (*westpa.core.data_manager.WESTDataManager* method), 196

create_iter_group() (*westpa.cli.tools.w_assign.WESTPAH5File* method), 33

`create_iter_group()`
(*westpa.core.h5io.WESTPAH5File* method), 219

`create_traj_group()`
(*westpa.oldtools.iframe.BFDataManager* method), 304

`create_traj_group()`
(*westpa.oldtools.iframe.data_reader.BFDataManager* method), 311

`cumulative_event_duration_histogram`
(*westpa.cli.tools.w_red.DurationCorrector* property), 105

`current` (*westpa.cli.tools.w_ipa.WIPI* property), 49

`current` (*westpa.cli.tools.w_ipa.WIPIScheme* property), 47

`current` (*westpa.tools.wipi.WIPIScheme* property), 298

`current` (*westpa.tools.WIPIScheme* property), 283

`current_iter_assignments`
(*westpa.core.binning.mab_driver.WEDriver* property), 169

`current_iter_assignments`
(*westpa.core.we_driver.WEDriver* property), 237

`current_iter_segments`
(*westpa.core.binning.mab_driver.WEDriver* property), 169

`current_iter_segments`
(*westpa.core.we_driver.WEDriver* property), 237

`current_iteration` (*westpa.core.data_manager.WESTDataManager* property), 195

D

`DAll` (class in *westpa.cli.tools.w_direct*), 74

`DAverage` (class in *westpa.cli.tools.w_direct*), 74

`days` (*westpa.core.sim_manager.timedelta* attribute), 225

`default_aux_compression_threshold`
(*westpa.core.data_manager.WESTDataManager* attribute), 195

`default_chunksize` (*westpa.oldtools.iframe.data_reader.ExtDataReaderMixin* attribute), 311

`default_chunksize` (*westpa.oldtools.iframe.ExtDataReaderMixin* attribute), 303

`default_comm_mode` (*westpa.work_managers.zeromq.core.ZMQCore* attribute), 261

`default_comm_mode` (*westpa.work_managers.zeromq.node.ZMQCore* attribute), 263

`default_comm_mode` (*westpa.work_managers.zeromq.work_manager.ZMQCore* attribute), 266

`default_comm_mode` (*westpa.work_managers.zeromq.worker.ZMQCore* attribute), 273

`default_comm_mode` (*westpa.work_managers.zeromq.ZMQCore* attribute), 256

`default_flush_period`
(*westpa.core.data_manager.WESTDataManager* attribute), 195

`default_iter_prec` (*westpa.cli.tools.w_assign.WESTPAH5File* attribute), 33

`default_iter_prec` (*westpa.core.data_manager.WESTDataManager* attribute), 194

`default_iter_prec` (*westpa.core.h5io.WESTPAH5File* attribute), 219

`default_kinetics_file`
(*westpa.cli.tools.w_direct.DAll* attribute), 74

`default_kinetics_file`
(*westpa.cli.tools.w_direct.DAverage* attribute), 74

`default_kinetics_file`
(*westpa.cli.tools.w_direct.DKinAvg* attribute), 71

`default_kinetics_file`
(*westpa.cli.tools.w_direct.DKinetics* attribute), 71

`default_kinetics_file`
(*westpa.cli.tools.w_direct.DStateProbs* attribute), 73

`default_kinetics_file`
(*westpa.cli.tools.w_kinavg.DKinAvg* attribute), 124

`default_kinetics_file`
(*westpa.cli.tools.w_kinavg.WKinAvg* attribute), 126

`default_kinetics_file`
(*westpa.cli.tools.w_kinetics.DKinetics* attribute), 130

`default_kinetics_file`
(*westpa.cli.tools.w_postanalysis_matrix.RWMatrix* attribute), 143

`default_kinetics_file`
(*westpa.cli.tools.w_postanalysis_reweight.PAAverage* attribute), 147

`default_kinetics_file`
(*westpa.cli.tools.w_postanalysis_reweight.RWAverage* attribute), 146

`default_kinetics_file`
(*westpa.cli.tools.w_reweight.RWAll* attribute), 155

`default_kinetics_file`
(*westpa.cli.tools.w_reweight.RWAverage* attribute), 155

`default_kinetics_file`
(*westpa.cli.tools.w_reweight.RWMatrix* attribute), 149

`default_kinetics_file`
(*westpa.cli.tools.w_reweight.RWRate* attribute), 151

<code>default_kinetics_file</code> (<i>westpa.cli.tools.w_reweight.RWStateProbs</i> attribute), 153	<code>default_output_file</code> (<i>westpa.cli.tools.w_reweight.RWAll</i> attribute), 155
<code>default_kinetics_file</code> (<i>westpa.cli.tools.w_stateprobs.DStateProbs</i> attribute), 137	<code>default_output_file</code> (<i>westpa.cli.tools.w_reweight.RWAverage</i> attribute), 155
<code>default_kinetics_file</code> (<i>westpa.cli.tools.w_stateprobs.WStateProbs</i> attribute), 139	<code>default_output_file</code> (<i>westpa.cli.tools.w_reweight.RWMatrix</i> at- tribute), 149
<code>default_master_heartbeat</code> (<i>westpa.work_managers.zeromq.core.ZMQCore</i> attribute), 261	<code>default_output_file</code> (<i>westpa.cli.tools.w_reweight.RWRate</i> at- tribute), 151
<code>default_master_heartbeat</code> (<i>westpa.work_managers.zeromq.node.ZMQCore</i> attribute), 263	<code>default_output_file</code> (<i>westpa.cli.tools.w_stateprobs.WStateProbs</i> attribute), 139
<code>default_master_heartbeat</code> (<i>westpa.work_managers.zeromq.work_manager.ZMQCore</i> attribute), 266	<code>default_output_file</code> (<i>westpa.tools.kinetics_tool.AverageCommands</i> attribute), 294
<code>default_master_heartbeat</code> (<i>westpa.work_managers.zeromq.worker.ZMQCore</i> attribute), 273	<code>default_parallel_work_manager</code> (<i>westpa.work_managers.environment.WMEnvironment</i> attribute), 245
<code>default_master_heartbeat</code> (<i>westpa.work_managers.zeromq.ZMQCore</i> attribute), 256	<code>default_shutdown_timeout</code> (<i>westpa.work_managers.zeromq.core.ZMQCore</i> attribute), 261
<code>default_output_file</code> (<i>westpa.cli.tools.w_direct.AverageCommands</i> attribute), 69	<code>default_shutdown_timeout</code> (<i>westpa.work_managers.zeromq.node.ZMQCore</i> attribute), 263
<code>default_output_file</code> (<i>westpa.cli.tools.w_direct.DKinetics</i> attribute), 71	<code>default_shutdown_timeout</code> (<i>westpa.work_managers.zeromq.work_manager.ZMQCore</i> attribute), 266
<code>default_output_file</code> (<i>westpa.cli.tools.w_kinavg.WKinAvg</i> attribute), 126	<code>default_shutdown_timeout</code> (<i>westpa.work_managers.zeromq.worker.ZMQCore</i> attribute), 273
<code>default_output_file</code> (<i>westpa.cli.tools.w_kinetics.DKinetics</i> at- tribute), 130	<code>default_shutdown_timeout</code> (<i>westpa.work_managers.zeromq.ZMQCore</i> attribute), 256
<code>default_output_file</code> (<i>westpa.cli.tools.w_kinetics.WKinetics</i> at- tribute), 131	<code>default_startup_timeout</code> (<i>westpa.work_managers.zeromq.core.ZMQCore</i> attribute), 261
<code>default_output_file</code> (<i>westpa.cli.tools.w_postanalysis_matrix.PAMatrix</i> attribute), 144	<code>default_startup_timeout</code> (<i>westpa.work_managers.zeromq.node.ZMQCore</i> attribute), 263
<code>default_output_file</code> (<i>westpa.cli.tools.w_postanalysis_matrix.RWMatrix</i> attribute), 143	<code>default_startup_timeout</code> (<i>westpa.work_managers.zeromq.work_manager.ZMQCore</i> attribute), 266
<code>default_output_file</code> (<i>westpa.cli.tools.w_postanalysis_reweight.PAAverage</i> attribute), 147	<code>default_startup_timeout</code> (<i>westpa.work_managers.zeromq.worker.ZMQCore</i> attribute), 273
<code>default_output_file</code> (<i>westpa.cli.tools.w_postanalysis_reweight.RWAverage</i> attribute), 146	<code>default_startup_timeout</code> (<i>westpa.work_managers.zeromq.ZMQCore</i> attribute), 256
<code>default_output_file</code> (<i>westpa.cli.tools.w_reweight.AverageCommands</i> attribute), 148	<code>default_timeout_factor</code> (<i>westpa.work_managers.zeromq.core.ZMQCore</i> attribute), 261

<code>default_timeout_factor</code> (<code>westpa.work_managers.zeromq.node.ZMQCore</code> attribute), 263	270 description (<code>westpa.cli.tools.ploterr.DirectKinetics</code> attribute), 118
<code>default_timeout_factor</code> (<code>westpa.work_managers.zeromq.work_manager.ZMQCore</code> attribute), 266	description (<code>westpa.cli.tools.ploterr.DirectStateprobs</code> attribute), 119
<code>default_timeout_factor</code> (<code>westpa.work_managers.zeromq.worker.ZMQCore</code> attribute), 273	description (<code>westpa.cli.tools.ploterr.GenericIntervalSubcommand</code> attribute), 117
<code>default_timeout_factor</code> (<code>westpa.work_managers.zeromq.ZMQCore</code> attribute), 256	description (<code>westpa.cli.tools.ploterr.PloterrsTool</code> attribute), 120
<code>default_we_h5file_driver</code> (<code>westpa.core.data_manager.WESTDataManager</code> attribute), 195	description (<code>westpa.cli.tools.ploterr.WESTSubcommand</code> attribute), 117
<code>default_we_h5filename</code> (<code>westpa.core.data_manager.WESTDataManager</code> attribute), 195	description (<code>westpa.cli.tools.plothist.AveragePlotHist</code> attribute), 114
<code>default_work_manager</code> (<code>westpa.work_managers.environment.WMEnvironment</code> attribute), 245	description (<code>westpa.cli.tools.plothist.EvolutionPlotHist</code> attribute), 115
<code>default_worker_heartbeat</code> (<code>westpa.work_managers.zeromq.core.ZMQCore</code> attribute), 261	description (<code>westpa.cli.tools.plothist.InstantPlotHist</code> attribute), 114
<code>default_worker_heartbeat</code> (<code>westpa.work_managers.zeromq.node.ZMQCore</code> attribute), 263	description (<code>westpa.cli.tools.plothist.PlotHistTool</code> attribute), 115
<code>default_worker_heartbeat</code> (<code>westpa.work_managers.zeromq.work_manager.ZMQCore</code> attribute), 266	description (<code>westpa.cli.tools.plothist.WESTSubcommand</code> attribute), 113
<code>default_worker_heartbeat</code> (<code>westpa.work_managers.zeromq.worker.ZMQCore</code> attribute), 273	description (<code>westpa.cli.tools.w_assign.WAssign</code> attribute), 33
<code>default_worker_heartbeat</code> (<code>westpa.work_managers.zeromq.ZMQCore</code> attribute), 256	description (<code>westpa.cli.tools.w_bins.WBinTool</code> attribute), 21
<code>del_iter_group()</code> (<code>westpa.core.data_manager.WESTDataManager</code> method), 195	description (<code>westpa.cli.tools.w_bins.WESTTool</code> attribute), 19
<code>del_iter_summary()</code> (<code>westpa.core.data_manager.WESTDataManager</code> method), 196	description (<code>westpa.cli.tools.w_crawl.WCrawl</code> attribute), 65
<code>delete_binning_group()</code> (<code>westpa.oldtools.iframe.binning.BinningMixin</code> method), 308	description (<code>westpa.cli.tools.w_direct.DAll</code> attribute), 74
<code>delete_binning_group()</code> (<code>westpa.oldtools.iframe.BinningMixin</code> method), 304	description (<code>westpa.cli.tools.w_direct.DAverage</code> attribute), 74
<code>delete_transitions_group()</code> (<code>westpa.oldtools.iframe.TransitionAnalysisMixin</code> method), 305	description (<code>westpa.cli.tools.w_direct.DKinAvg</code> attribute), 72
<code>delete_transitions_group()</code> (<code>westpa.oldtools.iframe.transitions.TransitionAnalysisMixin</code> method), 317	description (<code>westpa.cli.tools.w_direct.DKinetics</code> attribute), 73
<code>deque</code> (class in <code>westpa.work_managers.mpi</code>), 246	description (<code>westpa.cli.tools.w_dumpsegs.WDumpSegs</code> attribute), 142
<code>deque</code> (class in <code>westpa.work_managers.zeromq.work_manager.ZMQCore</code> attribute), 246	description (<code>westpa.cli.tools.w_dumpsegs.WESTTool</code> attribute), 140
	description (<code>westpa.cli.tools.w_eddist.WEDDist</code> attribute), 92
	description (<code>westpa.cli.tools.w_fluxanl.WESTTool</code> attribute), 160
	description (<code>westpa.cli.tools.w_fluxanl.WFluxanlTool</code> attribute), 162
	description (<code>westpa.cli.tools.w_kinavg.DKinAvg</code> attribute), 124
	description (<code>westpa.cli.tools.w_kinavg.WDirect</code> attribute), 126
	description (<code>westpa.cli.tools.w_kinetics.DKinetics</code> attribute), 126

- [tribute\), 130](#)
- [description \(westpa.cli.tools.w_kinetics.WDirect attribute\), 132](#)
- [description \(westpa.cli.tools.w_multi_west.WESTTool attribute\), 102](#)
- [description \(westpa.cli.tools.w_multi_west.WMultiWest attribute\), 103](#)
- [description \(westpa.cli.tools.w_ntop.WESTTool attribute\), 96](#)
- [description \(westpa.cli.tools.w_ntop.WNTopTool attribute\), 98](#)
- [description \(westpa.cli.tools.w_pdist.WPDist attribute\), 54](#)
- [description \(westpa.cli.tools.w_postanalysis_matrix.RWMKinAvg attribute\), 144](#)
- [description \(westpa.cli.tools.w_postanalysis_matrix.WRedKinetics attribute\), 144](#)
- [description \(westpa.cli.tools.w_postanalysis_reweight.RWReWeight attribute\), 146](#)
- [description \(westpa.cli.tools.w_postanalysis_reweight.WReWeight attribute\), 147](#)
- [description \(westpa.cli.tools.w_red.WRed attribute\), 106](#)
- [description \(westpa.cli.tools.w_reweight.RWAll attribute\), 155](#)
- [description \(westpa.cli.tools.w_reweight.RWAverage attribute\), 155](#)
- [description \(westpa.cli.tools.w_reweight.RWMatrix attribute\), 149](#)
- [description \(westpa.cli.tools.w_reweight.RWRate attribute\), 151](#)
- [description \(westpa.cli.tools.w_reweight.RWStateProbs attribute\), 153](#)
- [description \(westpa.cli.tools.w_select.WSelectTool attribute\), 80](#)
- [description \(westpa.cli.tools.w_stateprobs.DStateProbs attribute\), 137](#)
- [description \(westpa.cli.tools.w_stateprobs.WDirect attribute\), 139](#)
- [description \(westpa.cli.tools.w_trace.WESTTool attribute\), 37](#)
- [description \(westpa.cli.tools.w_trace.WTraceTool attribute\), 41](#)
- [description \(westpa.tools.core.WESTSubcommand attribute\), 287](#)
- [description \(westpa.tools.core.WESTTool attribute\), 286](#)
- [description \(westpa.tools.kinetics_tool.WESTSubcommand attribute\), 293](#)
- [description \(westpa.tools.WESTSubcommand attribute\), 277](#)
- [description \(westpa.tools.WESTTool attribute\), 276](#)
- [dfunc\(\) \(westpa.westext.adaptvoronoi.AdaptiveVoronoiDriver method\), 321](#)
- [dfunc\(\) \(westpa.westext.adaptvoronoi.adaptVor_driver.AdaptiveVoronoiDriver method\), 320](#)
- [direct \(westpa.cli.tools.w_ipa.WIPI property\), 48](#)
- [direct \(westpa.cli.tools.w_ipa.WIPIScheme property\), 47](#)
- [direct \(westpa.tools.wipi.WIPIScheme property\), 297](#)
- [direct \(westpa.tools.WIPIScheme property\), 282](#)
- [DirectKinetics \(class in westpa.cli.tools.ploterr\), 118](#)
- [DirectStateprobs \(class in westpa.cli.tools.ploterr\), 119](#)
- [dirname\(\) \(in module westpa.core.data_manager\), 189](#)
- [distance_unit \(westpa.core.h5io.HDF5TrajectoryFile attribute\), 212](#)
- [DKinAvg \(class in westpa.cli.tools.w_direct\), 71](#)
- [DKinAvg \(class in westpa.cli.tools.w_kinavg\), 124](#)
- [DKinetics \(class in westpa.cli.tools.w_direct\), 70](#)
- [DKinetics \(class in westpa.cli.tools.w_kinetics\), 130](#)
- [do_average_plot_1d\(\) \(westpa.cli.tools.plothist.AveragePlotHist method\), 114](#)
- [do_average_plot_2d\(\) \(westpa.cli.tools.plothist.AveragePlotHist method\), 114](#)
- [do_instant_plot_1d\(\) \(westpa.cli.tools.plothist.InstantPlotHist method\), 114](#)
- [do_instant_plot_2d\(\) \(westpa.cli.tools.plothist.InstantPlotHist method\), 114](#)
- [do_plot\(\) \(westpa.cli.tools.ploterr.CommonPloterr method\), 117](#)
- [done \(westpa.work_managers.core.WMFuture property\), 244](#)
- [done \(westpa.work_managers.mpi.WMFuture property\), 248](#)
- [done \(westpa.work_managers.processes.WMFuture property\), 251](#)
- [done \(westpa.work_managers.serial.WMFuture property\), 253](#)
- [done \(westpa.work_managers.threads.WMFuture property\), 255](#)
- [done \(westpa.work_managers.zeromq.work_manager.WMFuture property\), 270](#)
- [draw\(\) \(westpa.core.progress.ProgressIndicator method\), 224](#)
- [draw\(\) \(westpa.tools.progress.ProgressIndicator method\), 294](#)
- [draw_fancy\(\) \(westpa.core.progress.ProgressIndicator method\), 224](#)
- [draw_fancy\(\) \(westpa.tools.progress.ProgressIndicator method\), 294](#)
- [draw_simple\(\) \(westpa.core.progress.ProgressIndicator method\), 224](#)
- [draw_simple\(\) \(westpa.tools.progress.ProgressIndicator method\), 294](#)

method), 294

drop_cache() (westpa.core.h5io.IterBlockedDataset method), 221

DSSpec (class in westpa.core.h5io), 220

DStateProbs (class in westpa.cli.tools.w_direct), 72

DStateProbs (class in westpa.cli.tools.w_stateprobs), 137

dtau (westpa.cli.tools.w_red.RateCalculator property), 106

duration (westpa.work_managers.zeromq.core.PassiveTime attribute), 260

DurationCorrector (class in westpa.cli.tools.w_red), 105

DurationDataset (class in westpa.cli.tools.w_eddist), 92

E

EDF (class in westpa.oldtools.stats.edfs), 318

emit_header (westpa.core.textio.NumericTextOutputFormatter attribute), 234

emit_trace_h5() (westpa.cli.tools.w_trace.WTraceTool method), 42

emit_trace_text() (westpa.cli.tools.w_trace.WTraceTool method), 42

empty_like() (westpa.core.h5io.IterBlockedDataset class method), 221

endpoint_type_names (westpa.cli.core.w_fork.Segment attribute), 25

endpoint_type_names (westpa.cli.core.w_states.Segment attribute), 85

endpoint_type_names (westpa.cli.core.w_succ.Segment attribute), 58

endpoint_type_names (westpa.cli.tools.w_dumpsegs.Segment attribute), 142

endpoint_type_names (westpa.cli.tools.w_trace.Segment attribute), 39

endpoint_type_names (westpa.core.binning.mab_manager.Segment attribute), 173

endpoint_type_names (westpa.core.data_manager.Segment attribute), 189

endpoint_type_names (westpa.core.propagators.executable.Segment attribute), 184

endpoint_type_names (westpa.core.segment.Segment attribute), 225

endpoint_type_names (westpa.core.sim_manager.Segment attribute), 227

endpoint_type_names (westpa.core.states.Segment attribute), 230

endpoint_type_names (westpa.core.we_driver.Segment attribute), 235

endpoint_type_names (westpa.oldtools.iframe.data_reader.Segment attribute), 309

endpoint_type_text (westpa.cli.core.w_fork.Segment property), 25

endpoint_type_text (westpa.cli.core.w_states.Segment property), 85

endpoint_type_text (westpa.cli.core.w_succ.Segment property), 58

endpoint_type_text (westpa.cli.tools.w_dumpsegs.Segment property), 142

endpoint_type_text (westpa.cli.tools.w_trace.Segment property), 39

endpoint_type_text (westpa.core.binning.mab_manager.Segment property), 174

endpoint_type_text (westpa.core.data_manager.Segment property), 190

endpoint_type_text (westpa.core.propagators.executable.Segment property), 184

endpoint_type_text (westpa.core.segment.Segment property), 225

endpoint_type_text (westpa.core.sim_manager.Segment property), 227

endpoint_type_text (westpa.core.states.Segment property), 231

endpoint_type_text (westpa.core.we_driver.Segment property), 235

endpoint_type_text (westpa.oldtools.iframe.data_reader.Segment property), 309

endpoint_types (westpa.cli.core.w_fork.Segment attribute), 24

endpoint_types (westpa.cli.core.w_states.Segment attribute), 85

endpoint_types (westpa.cli.core.w_succ.Segment attribute), 57

endpoint_types (westpa.cli.tools.w_dumpsegs.Segment attribute), 141

endpoint_types (westpa.cli.tools.w_trace.Segment attribute), 39

endpoint_types (westpa.core.binning.mab_manager.Segment attribute), 173

endpoint_types (westpa.core.data_manager.Segment attribute), 189

endpoint_types (westpa.core.propagators.executable.Segment attribute), 184

endpoint_types (westpa.core.segment.Segment attribute), 225

endpoint_types (westpa.core.sim_manager.Segment attribute), 227

- attribute*), 227
- `endpoint_types` (*westpa.core.states.Segment attribute*), 230
- `endpoint_types` (*westpa.core.we_driver.Segment attribute*), 235
- `endpoint_types` (*westpa.oldtools.iframe.data_reader.Segment attribute*), 309
- `ensure_type()` (*in module westpa.core.h5io*), 210
- `entry_point()` (*in module westpa.cli.core.w_fork*), 26
- `entry_point()` (*in module westpa.cli.core.w_init*), 16
- `entry_point()` (*in module westpa.cli.core.w_run*), 22
- `entry_point()` (*in module westpa.cli.core.w_states*), 87
- `entry_point()` (*in module westpa.cli.core.w_succ*), 59
- `entry_point()` (*in module westpa.cli.core.w_truncate*), 23
- `entry_point()` (*in module westpa.cli.tools.ploterr*), 120
- `entry_point()` (*in module westpa.cli.tools.plothist*), 115
- `entry_point()` (*in module westpa.cli.tools.w_assign*), 35
- `entry_point()` (*in module westpa.cli.tools.w_bins*), 21
- `entry_point()` (*in module westpa.cli.tools.w_crawl*), 65
- `entry_point()` (*in module westpa.cli.tools.w_direct*), 74
- `entry_point()` (*in module westpa.cli.tools.w_dumpsegs*), 142
- `entry_point()` (*in module westpa.cli.tools.w_eddist*), 94
- `entry_point()` (*in module westpa.cli.tools.w_fluxanl*), 162
- `entry_point()` (*in module westpa.cli.tools.w_ipa*), 49
- `entry_point()` (*in module westpa.cli.tools.w_kinavg*), 127
- `entry_point()` (*in module westpa.cli.tools.w_kinetics*), 132
- `entry_point()` (*in module westpa.cli.tools.w_multi_west*), 103
- `entry_point()` (*in module westpa.cli.tools.w_ntop*), 99
- `entry_point()` (*in module westpa.cli.tools.w_pdist*), 56
- `entry_point()` (*in module westpa.cli.tools.w_postanalysis_matrix*), 145
- `entry_point()` (*in module westpa.cli.tools.w_postanalysis_reweight*), 147
- `entry_point()` (*in module westpa.cli.tools.w_red*), 106
- `entry_point()` (*in module westpa.cli.tools.w_reweight*), 155
- `entry_point()` (*in module westpa.cli.tools.w_select*), 81
- `entry_point()` (*in module westpa.cli.tools.w_stateprobs*), 139
- `entry_point()` (*in module westpa.cli.tools.w_trace*), 42
- `ENV_BSTATE_DATA_REF` (*westpa.core.data_manager.ExecutablePropagator attribute*), 192
- `ENV_BSTATE_DATA_REF` (*westpa.core.propagators.executable.ExecutablePropagator attribute*), 185
- `ENV_BSTATE_ID` (*westpa.core.data_manager.ExecutablePropagator attribute*), 192
- `ENV_BSTATE_ID` (*westpa.core.propagators.executable.ExecutablePropagator attribute*), 185
- `ENV_CURRENT_ITER` (*westpa.core.data_manager.ExecutablePropagator attribute*), 192
- `ENV_CURRENT_ITER` (*westpa.core.propagators.executable.ExecutablePropagator attribute*), 185
- `ENV_CURRENT_SEG_DATA_REF` (*westpa.core.data_manager.ExecutablePropagator attribute*), 192
- `ENV_CURRENT_SEG_DATA_REF` (*westpa.core.propagators.executable.ExecutablePropagator attribute*), 185
- `ENV_CURRENT_SEG_ID` (*westpa.core.data_manager.ExecutablePropagator attribute*), 192
- `ENV_CURRENT_SEG_ID` (*westpa.core.propagators.executable.ExecutablePropagator attribute*), 185
- `ENV_CURRENT_SEG_INITPOINT` (*westpa.core.data_manager.ExecutablePropagator attribute*), 192
- `ENV_CURRENT_SEG_INITPOINT` (*westpa.core.propagators.executable.ExecutablePropagator attribute*), 185
- `ENV_ISTATE_DATA_REF` (*westpa.core.data_manager.ExecutablePropagator attribute*), 192
- `ENV_ISTATE_DATA_REF` (*westpa.core.propagators.executable.ExecutablePropagator attribute*), 185
- `ENV_ISTATE_ID` (*westpa.core.data_manager.ExecutablePropagator attribute*), 192
- `ENV_ISTATE_ID` (*westpa.core.propagators.executable.ExecutablePropagator attribute*), 185
- `env_name()` (*westpa.work_managers.environment.WMEnvironment method*), 245
- `ENV_PARENT_DATA_REF` (*westpa.core.data_manager.ExecutablePropagator attribute*), 192
- `ENV_PARENT_DATA_REF` (*westpa.core.propagators.executable.ExecutablePropagator attribute*), 185
- `ENV_PARENT_SEG_ID` (*westpa.core.data_manager.ExecutablePropagator attribute*), 192
- `ENV_PARENT_SEG_ID` (*westpa.core.propagators.executable.ExecutablePropagator attribute*), 185
- `env_prefix` (*westpa.work_managers.environment.WMEnvironment attribute*), 245

ENV_RAND128 (*westpa.core.data_manager.ExecutablePropagator* property), 255
attribute), 193 exception (*westpa.work_managers.zeromq.work_manager.WMFuture*
ENV_RAND128 (*westpa.core.propagators.executable.ExecutablePropagator* property), 270
attribute), 186 exclude_arg() (*westpa.tools.binning.WESTToolComponent*
ENV_RAND16 (*westpa.core.data_manager.ExecutablePropagator* method), 283
attribute), 192 exclude_arg() (*westpa.tools.core.WESTToolComponent*
ENV_RAND16 (*westpa.core.propagators.executable.ExecutablePropagator* method), 285
attribute), 185 exclude_arg() (*westpa.tools.data_reader.WESTToolComponent*
ENV_RAND32 (*westpa.core.data_manager.ExecutablePropagator* method), 288
attribute), 192 exclude_arg() (*westpa.tools.iter_range.WESTToolComponent*
ENV_RAND32 (*westpa.core.propagators.executable.ExecutablePropagator* method), 290
attribute), 185 exclude_arg() (*westpa.tools.progress.WESTToolComponent*
ENV_RAND64 (*westpa.core.data_manager.ExecutablePropagator* method), 295
attribute), 192 exclude_arg() (*westpa.tools.selected_segs.WESTToolComponent*
ENV_RAND64 (*westpa.core.propagators.executable.ExecutablePropagator* method), 295
attribute), 185 exclude_arg() (*westpa.tools.WESTToolComponent*
ENV_RANDFLOAT (*westpa.core.data_manager.ExecutablePropagator* method), 277
attribute), 193 exec_child() (*westpa.core.data_manager.ExecutablePropagator*
ENV_RANDFLOAT (*westpa.core.propagators.executable.ExecutablePropagator* method), 193
attribute), 186 exec_child() (*westpa.core.propagators.executable.ExecutablePropagator*
ENV_STRUCT_DATA_REF method), 186
(*westpa.core.data_manager.ExecutablePropagator* exec_child_from_child_info()
attribute), 192 (*westpa.core.data_manager.ExecutablePropagator*
ENV_STRUCT_DATA_REF method), 193
(*westpa.core.propagators.executable.ExecutablePropagator* exec_child_from_child_info()
attribute), 185 (*westpa.core.propagators.executable.ExecutablePropagator*
epilog (*westpa.cli.tools.w_bins.WESTTool* attribute), 19 method), 186
epilog (*westpa.cli.tools.w_dumpsegs.WESTTool* attribute), 140
epilog (*westpa.cli.tools.w_fluxanl.WESTTool* attribute),
160
epilog (*westpa.cli.tools.w_multi_west.WESTTool* attribute), 102
epilog (*westpa.cli.tools.w_ntop.WESTTool* attribute), 96
epilog (*westpa.cli.tools.w_trace.WESTTool* attribute),
37
epilog (*westpa.tools.core.WESTTool* attribute), 286
epilog (*westpa.tools.WESTTool* attribute), 276
estimate_rates() (in module
westpa.core.kinetics.matrates), 176
event_duration_histogram
(*westpa.cli.tools.w_red.DurationCorrector*
property), 105
EvolutionPlotHist (class in *westpa.cli.tools.plothist*),
114
exception (*westpa.work_managers.core.WMFuture* property), 244
exception (*westpa.work_managers.mpi.WMFuture* property), 248
exception (*westpa.work_managers.processes.WMFuture* property), 250
exception (*westpa.work_managers.serial.WMFuture* property), 253
exception (*westpa.work_managers.threads.WMFuture* property), 255
exec_for_basis_state()
(*westpa.core.data_manager.ExecutablePropagator*
method), 193
exec_for_basis_state()
(*westpa.core.propagators.executable.ExecutablePropagator*
method), 186
exec_for_initial_state()
(*westpa.core.data_manager.ExecutablePropagator*
method), 193
exec_for_initial_state()
(*westpa.core.propagators.executable.ExecutablePropagator*
method), 186
exec_for_iteration()
(*westpa.core.data_manager.ExecutablePropagator*
method), 193
exec_for_iteration()
(*westpa.core.propagators.executable.ExecutablePropagator*
method), 186
exec_for_segment() (*westpa.core.data_manager.ExecutablePropagator*
method), 193
exec_for_segment() (*westpa.core.propagators.executable.ExecutablePropagator*
method), 186
ExecutablePropagator (class in
westpa.core.data_manager), 192
ExecutablePropagator (class in
westpa.core.propagators.executable), 185
execute() (*westpa.work_managers.zeromq.core.Task*

- method), 260
- execute() (westpa.work_managers.zeromq.work_manager.Task method), 267
- execute() (westpa.work_managers.zeromq.worker.Task method), 275
- expandvars() (in module westpa.core.binning.mab), 168
- expired (westpa.work_managers.zeromq.core.PassiveTime property), 260
- expired() (westpa.work_managers.zeromq.core.PassiveMultiTask method), 261
- expired() (westpa.work_managers.zeromq.node.PassiveMultiTask method), 265
- expired() (westpa.work_managers.zeromq.work_manager.PassiveMultiTask method), 268
- expired() (westpa.work_managers.zeromq.worker.PassiveMultiTask method), 275
- expires_in (westpa.work_managers.zeromq.core.PassiveTime property), 260
- expiring_flushing_lock (class in westpa.core.data_manager), 194
- expiring_flushing_lock() (westpa.core.data_manager.WESTDataManager method), 195
- ExtDataReaderMixin (class in westpa.oldtools.iframe), 303
- ExtDataReaderMixin (class in westpa.oldtools.iframe.data_reader), 311
- extend() (westpa.work_managers.mpi.dequeue method), 246
- extend() (westpa.work_managers.zeromq.work_manager.dequeue method), 271
- extendleft() (westpa.work_managers.mpi.dequeue method), 246
- extendleft() (westpa.work_managers.zeromq.work_manager.dequeue method), 271
- extent (westpa.core.progress.ProgressIndicator property), 224
- extent (westpa.tools.progress.ProgressIndicator property), 295
- extract_data() (westpa.core.kinetics.rate_averaging.RateAverager method), 194
- extract_data() (westpa.core.kinetics.RateAverager method), 174
- extract_data() (westpa.westext.weed.weed_driver.RateAverager method), 327
- extract_data() (westpa.westext.wess.wess_driver.RateAverager method), 328
- extract_fluxes() (in module westpa.cli.tools.w_fluxanl), 161
- fconcat (westpa.analysis.trajectories.Trajectory property), 338
- fftconvolve() (in module westpa.cli.tools.w_fluxanl), 157
- fget (westpa.analysis.trajectories.Trajectory property), 338
- FileLinkedDSSpec (class in westpa.core.h5io), 221
- final_pcoord() (westpa.cli.core.w_fork.Segment static method), 25
- final_pcoord() (westpa.cli.core.w_states.Segment static method), 85
- final_pcoord() (westpa.cli.core.w_succ.Segment static method), 58
- final_pcoord() (westpa.cli.tools.w_dumpsegs.Segment static method), 142
- final_pcoord() (westpa.cli.tools.w_trace.Segment static method), 39
- final_pcoord() (westpa.core.binning.mab_manager.Segment static method), 173
- final_pcoord() (westpa.core.data_manager.Segment static method), 190
- final_pcoord() (westpa.core.propagators.executable.Segment static method), 184
- final_pcoord() (westpa.core.segment.Segment static method), 225
- final_pcoord() (westpa.core.sim_manager.Segment static method), 227
- final_pcoord() (westpa.core.states.Segment static method), 230
- final_pcoord() (westpa.core.we_driver.Segment static method), 235
- final_pcoord() (westpa.oldtools.iframe.data_reader.Segment static method), 309
- finalize() (westpa.cli.tools.w_crawl.WESTPACrawler method), 65
- finalize_iteration() (westpa.core.binning.mab_manager.WESimManager method), 171
- finalize_iteration() (westpa.core.data_manager.ExecutablePropagator method), 187
- finalize_iteration() (westpa.core.propagators.executable.ExecutablePropagator method), 181
- finalize_iteration() (westpa.core.propagators.WESTPropagator method), 179
- finalize_iteration() (westpa.core.sim_manager.WESimManager method), 229
- finalize_run() (westpa.core.binning.mab_manager.WESimManager

method), 171

finalize_run() (westpa.core.data_manager.WESTDataManager method), 197

finalize_run() (westpa.core.sim_manager.WESimManager method), 229

finalize_run() (westpa.core.systems.WESTSystem method), 233

finalize_run() (westpa.core.yamlcfg.YAMLSystem method), 240

find_bin_mapper() (westpa.core.data_manager.WESTDataManager method), 197

find_ibstate_group() (westpa.core.data_manager.WESTDataManager method), 195

find_macrostate_transitions() (in module westpa.core.kinetics), 175

find_macrostate_transitions() (in module westpa.core.kinetics.events), 175

find_successful_trajs() (westpa.cli.core.w_succ.WSucc method), 59

find_transitions() (westpa.oldtools.iframe.BFTransitionAnalysisMixin method), 306

find_transitions() (westpa.oldtools.iframe.TransitionAnalysisMixin method), 306

find_transitions() (westpa.oldtools.iframe.transitions.BFTransitionAnalysisMixin method), 317

find_transitions() (westpa.oldtools.iframe.transitions.TransitionAnalysisMixin method), 317

find_tstate_group() (westpa.core.data_manager.WESTDataManager method), 195

flat_to_nested_matrix() (in module westpa.core.kinetics), 174

flat_to_nested_vector() (in module westpa.core.kinetics), 175

flat_to_nested_vector() (in module westpa.core.kinetics.matrates), 176

flush() (westpa.core.h5io.HDF5TrajectoryFile method), 214

flush() (westpa.core.propagators.executable.BytesIO method), 180

flush_backing() (westpa.core.data_manager.WESTDataManager method), 195

flush_transition_data() (westpa.oldtools.iframe.TransitionEventAccumulator method), 306

flush_transition_data() (westpa.oldtools.iframe.transitions.TransitionEventAccumulator method), 316

flushing_lock (class in westpa.core.data_manager), 194

flushing_lock() (westpa.core.data_manager.WESTDataManager method), 195

flux_assign() (in module westpa.core.kinetics.rate_averaging), 177

flux_output_filename (westpa.cli.tools.ploterr.DirectKinetics attribute), 118

flux_output_filename (westpa.cli.tools.ploterr.ReweightKinetics attribute), 119

FluxMatrix (class in westpa.cli.tools.w_reweight), 149

FluxMatrix (class in westpa.core.reweight), 187

FluxMatrix (class in westpa.core.reweight.matrix), 187

FnDSSpec (class in westpa.core.h5io), 221

FnDSSpec (class in westpa.tools.data_reader), 288

forcefield (westpa.core.h5io.HDF5TrajectoryFile attribute), 211

forcefield (westpa.core.h5io.HDF5TrajectoryFile property), 212

Frames (class in westpa.core.h5io), 215

from_array() (westpa.oldtools.stats.edfs.EDF static method), 318

from_arrays() (westpa.oldtools.stats.edfs.EDF static method), 318

from_data_manager() (westpa.cli.tools.w_trace.Trace class method), 40

from_environ() (westpa.work_managers.core.WorkManager class method), 243

from_environ() (westpa.work_managers.mpi.MPIWorkManager class method), 248

from_environ() (westpa.work_managers.mpi.WorkManager class method), 247

from_environ() (westpa.work_managers.processes.ProcessWorkManager class method), 251

from_environ() (westpa.work_managers.processes.WorkManager class method), 249

from_environ() (westpa.work_managers.ProcessWorkManager class method), 242

from_environ() (westpa.work_managers.serial.SerialWorkManager class method), 253

from_environ() (westpa.work_managers.serial.WorkManager class method), 252

from_environ() (westpa.work_managers.SerialWorkManager class method), 241

from_environ() (westpa.work_managers.threads.ThreadsWorkManager class method), 255

from_environ() (westpa.work_managers.threads.WorkManager class method), 254

from_environ() (westpa.work_managers.ThreadsWorkManager class method), 241

from_environ() (westpa.work_managers.zeromq.work_manager.WorkManager class method), 269

from_environ() (westpa.work_managers.zeromq.work_manager.ZMQWorkManager class method), 271

from_environ() (westpa.work_managers.zeromq.ZMQWorkManager class method), 258

`from_iter()` (`westpa.tools.selected_segs.AllSegmentSelection` method), 296
`from_iter()` (`westpa.tools.selected_segs.SegmentSelection` method), 296
`from_iter()` (`westpa.trajtree.trajtree.AllSegmentSelection` method), 300
`from_kinetics_file()` (`westpa.cli.tools.w_red.DurationCorrector` static method), 105
`from_string()` (`westpa.core.h5io.SingleDSSpec` class method), 221
`from_text()` (`westpa.tools.selected_segs.SegmentSelection` class method), 296
`fstate` (`westpa.cli.tools.w_red.RateCalculator` property), 106
`FuncBinMapper` (class in `westpa.core.binning`), 162
`FuncBinMapper` (class in `westpa.core.binning.assign`), 167
`FuncBinMapper` (class in `westpa.core.binning.mab`), 168
`future` (`westpa.cli.tools.w_ipa.WIPI` property), 49
`FutureWatcher` (class in `westpa.work_managers.core`), 244

G

`gen_istate()` (in module `westpa.core.wm_ops`), 238
`gen_istate()` (`westpa.core.data_manager.ExecutablePropagator` method), 194
`gen_istate()` (`westpa.core.propagators.executable.ExecutablePropagator` method), 187
`gen_istate()` (`westpa.core.propagators.executable.WESTPropagator` method), 181
`gen_istate()` (`westpa.core.propagators.WESTPropagator` method), 179
`generate_file_list()` (`westpa.cli.tools.w_multi_west.WESTMultiTool` method), 103
`generate_file_list()` (`westpa.tools.core.WESTMultiTool` method), 287
`generate_file_list()` (`westpa.tools.WESTMultiTool` method), 278
`generate_future()` (in module `westpa.cli.tools.w_reweight`), 149
`generate_future()` (in module `westpa.tools.kinetics_tool`), 293
`generate_reweight_data()` (`westpa.cli.tools.w_reweight.RWReweight` method), 150
`GenericIntervalSubcommand` (class in `westpa.cli.tools.ploterr`), 117
`get()` (`westpa.core.yamlcfg.YAMLConfig` method), 239
`get_all_parent_ids()` (`westpa.core.data_manager.WESTDataManager` method), 197
`get_basis_states()` (`westpa.core.data_manager.WESTDataManager` method), 196
`get_bin_assignments()` (`westpa.oldtools.iframe.binning.BinningMixin` method), 308
`get_bin_assignments()` (`westpa.oldtools.iframe.BinningMixin` method), 305
`get_bin_mapper()` (in module `westpa.cli.tools.w_multi_west`), 103
`get_bin_mapper()` (`westpa.core.data_manager.WESTDataManager` method), 197
`get_bin_populations()` (`westpa.oldtools.iframe.binning.BinningMixin` method), 308
`get_bin_populations()` (`westpa.oldtools.iframe.BinningMixin` method), 305
`get_bssize()` (in module `westpa.mclib`), 298
`get_bssize()` (in module `westpa.oldtools.stats.mcbs`), 319
`get_bstate_pcoords()` (`westpa.core.binning.mab_manager.WESimManager` method), 170
`get_bstate_pcoords()` (`westpa.core.sim_manager.WESimManager` method), 229
`get_child_ids()` (`westpa.core.data_manager.WESTDataManager` method), 197
`get_children()` (`westpa.cli.core.w_succ.WESTDataReaderMixin` method), 59
`get_children()` (`westpa.core.data_manager.WESTDataManager` method), 197
`get_children()` (`westpa.oldtools.iframe.data_reader.WESTDataReaderMixin` method), 310
`get_children()` (`westpa.oldtools.iframe.WESTDataReaderMixin` method), 302
`get_choice()` (`westpa.core.yamlcfg.YAMLConfig` method), 240
`get_created_seg_ids()` (`westpa.cli.core.w_succ.WESTDataReaderMixin` method), 59
`get_created_seg_ids()` (`westpa.oldtools.iframe.data_reader.WESTDataReaderMixin` method), 310
`get_created_seg_ids()` (`westpa.oldtools.iframe.WESTDataReaderMixin` method), 303
`get_creator_data()` (in module `westpa.core.h5io`), 216
`get_dfunc_method()` (`westpa.westext.adaptvoronoi.AdaptiveVoronoiDriver` method), 321
`get_dfunc_method()` (`westpa.westext.adaptvoronoi.adaptVor_driver.AdaptiveVoronoiDriver` method), 320

<code>get_exception()</code> (<code>westpa.work_managers.core.WMFuture</code> method), 244	<code>get_iter_data()</code> (<code>westpa.core.h5io.DSSpec</code> method), 221
<code>get_exception()</code> (<code>westpa.work_managers.mpi.WMFuture</code> method), 248	<code>get_iter_data()</code> (<code>westpa.core.h5io.FnDSSpec</code> method), 221
<code>get_exception()</code> (<code>westpa.work_managers.processes.WMFuture</code> method), 250	<code>get_iter_data()</code> (<code>westpa.core.h5io.MultiDSSpec</code> method), 221
<code>get_exception()</code> (<code>westpa.work_managers.serial.WMFuture</code> method), 253	<code>get_iter_data()</code> (<code>westpa.core.h5io.SingleIterDSSpec</code> method), 221
<code>get_exception()</code> (<code>westpa.work_managers.threads.WMFuture</code> method), 255	<code>get_iter_data()</code> (<code>westpa.core.h5io.SingleSegmentDSSpec</code> method), 221
<code>get_exception()</code> (<code>westpa.work_managers.zeromq.work_manager.WMFuture</code> method), 270	<code>get_iter_data()</code> (<code>westpa.tools.data_reader.FnDSSpec</code> method), 288
<code>get_extent()</code> (<code>westpa.cli.tools.plothist.NonUniformImage</code> method), 111	<code>get_iter_data()</code> (<code>westpa.tools.data_reader.MultiDSSpec</code> method), 288
<code>get_identification()</code> (<code>westpa.work_managers.zeromq.core.ZMQCore</code> method), 261	<code>get_iter_data()</code> (<code>westpa.tools.data_reader.SingleIterDSSpec</code> method), 289
<code>get_identification()</code> (<code>westpa.work_managers.zeromq.node.ZMQCore</code> method), 263	<code>get_iter_data()</code> (<code>westpa.tools.data_reader.SingleSegmentDSSpec</code> method), 289
<code>get_identification()</code> (<code>westpa.work_managers.zeromq.work_manager.ZMQCore</code> method), 266	<code>get_iter_group()</code> (<code>westpa.cli.core.w_succ.WESTDataReaderMixin</code> method), 58
<code>get_identification()</code> (<code>westpa.work_managers.zeromq.worker.ZMQCore</code> method), 273	<code>get_iter_group()</code> (<code>westpa.cli.tools.w_assign.WESTPAH5File</code> method), 33
<code>get_identification()</code> (<code>westpa.work_managers.zeromq.ZMQCore</code> method), 257	<code>get_iter_group()</code> (<code>westpa.core.data_manager.WESTDataManager</code> method), 195
<code>get_initial_centers()</code> (<code>westpa.westext.adaptvoronoi.AdaptiveVoronoiDriver</code> method), 321	<code>get_iter_group()</code> (<code>westpa.core.h5io.WESTPAH5File</code> method), 219
<code>get_initial_centers()</code> (<code>westpa.westext.adaptvoronoi.adaptVor_driver.AdaptiveVoronoiDriver</code> method), 320	<code>get_iter_group()</code> (<code>westpa.oldtools.aframe.data_reader.WESTDataReaderMixin</code> method), 310
<code>get_initial_nodes()</code> (<code>westpa.oldtools.aframe.TrajWalker</code> method), 305	<code>get_iter_group()</code> (<code>westpa.oldtools.aframe.WESTDataReaderMixin</code> method), 302
<code>get_initial_nodes()</code> (<code>westpa.oldtools.aframe.trajwalker.TrajWalker</code> method), 315	<code>get_iter_range()</code> (in module <code>westpa.core.h5io</code>), 217
<code>get_initial_nodes()</code> (<code>westpa.oldtools.aframe.transitions.TrajWalker</code> method), 316	<code>get_iter_step()</code> (in module <code>westpa.core.h5io</code>), 217
<code>get_initial_states()</code> (<code>westpa.core.data_manager.WESTDataManager</code> method), 196	<code>get_iter_summary()</code> (<code>westpa.core.data_manager.WESTDataManager</code> method), 196
<code>get_istate_futures()</code> (<code>westpa.core.binning.mab_manager.WESimManager</code> method), 171	<code>get_iteration_entry()</code> (in module <code>westpa.core.h5io</code>), 217
<code>get_istate_futures()</code> (<code>westpa.core.sim_manager.WESimManager</code> method), 229	<code>get_iteration_slice()</code> (in module <code>westpa.core.h5io</code>), 217
<code>get_iter_data()</code> (<code>westpa.cli.tools.w_eddist.DurationData</code> method), 221	<code>get_macrostate_rates()</code> (in module <code>westpa.core.kinetics.matrates</code>), 176
	<code>get_mapper_func()</code> (<code>westpa.westext.adaptvoronoi.AdaptiveVoronoiDriver</code> method), 321
	<code>get_mapper_func()</code> (<code>westpa.westext.adaptvoronoi.adaptVor_driver.AdaptiveVoronoiDriver</code> method), 320
	<code>get_max_traj_len()</code> (<code>westpa.oldtools.aframe.BFDDataManager</code> method), 304
	<code>get_max_traj_len()</code> (<code>westpa.oldtools.aframe.data_reader.BFDDataManager</code> method), 311
	<code>get_n_traj()</code> (<code>westpa.oldtools.aframe.BFDDataManager</code> method), 304
	<code>get_n_traj()</code> (<code>westpa.oldtools.aframe.data_reader.BFDDataManager</code> method), 311
	<code>get_new_weight_data()</code>

(*westpa.core.data_manager.WESTDataManager* method), 197

`get_object()` (in module *westpa.cli.tools.plothist*), 113

`get_object()` (in module *westpa.cli.tools.w_assign*), 33

`get_object()` (in module *westpa.cli.tools.w_crawl*), 65

`get_object()` (in module *westpa.cli.tools.w_select*), 80

`get_object()` (in module *westpa.core.extloader*), 198

`get_object()` (in module *westpa.core.propagators.executable*), 181

`get_object()` (in module *westpa.tools.binning*), 283

`get_object()` (in module *westpa.tools.data_reader*), 288

`get_parent_array()` (*westpa.cli.core.w_succ.WESTDataReaderMixin* method), 59

`get_parent_array()` (*westpa.oldtools.iframe.data_reader.WESTDataReaderMixin* method), 310

`get_parent_array()` (*westpa.oldtools.iframe.WESTDataReaderMixin* method), 303

`get_parent_ids()` (*westpa.core.data_manager.WESTDataManager* method), 197

`get_path()` (*westpa.core.yamlcfg.YAMLConfig* method), 239

`get_pathlist()` (*westpa.core.yamlcfg.YAMLConfig* method), 239

`get_pcoord()` (in module *westpa.core.wm_ops*), 238

`get_pcoord()` (*westpa.core.data_manager.ExecutablePropagator* method), 193

`get_pcoord()` (*westpa.core.propagators.executable.ExecutablePropagator* method), 186

`get_pcoord()` (*westpa.core.propagators.executable.WESTPropagator* method), 181

`get_pcoord()` (*westpa.core.propagators.WESTPropagator* method), 179

`get_pcoord_array()` (*westpa.cli.core.w_succ.WESTDataReaderMixin* method), 59

`get_pcoord_array()` (*westpa.oldtools.iframe.BFDataManager* method), 304

`get_pcoord_array()` (*westpa.oldtools.iframe.data_reader.WESTDataReaderMixin* method), 312

`get_pcoord_array()` (*westpa.oldtools.iframe.data_reader.WESTDataReaderMixin* method), 310

`get_pcoord_array()` (*westpa.oldtools.iframe.WESTDataReaderMixin* method), 303

`get_pcoord_dataset()` (*westpa.cli.core.w_succ.WESTDataReaderMixin* method), 59

`get_pcoord_dataset()` (*westpa.oldtools.iframe.BFDataManager* method), 304

`get_pcoord_dataset()` (*westpa.oldtools.iframe.data_reader.WESTDataReaderMixin* method), 312

`get_pcoord_dataset()` (*westpa.oldtools.iframe.data_reader.WESTDataReaderMixin* method), 303

`get_pcoord_dataset()` (*westpa.oldtools.iframe.WESTDataReaderMixin* method), 310

`get_pcoord_len()` (*westpa.cli.core.w_succ.WESTDataReaderMixin* method), 59

`get_pcoord_len()` (*westpa.oldtools.iframe.data_reader.WESTDataReaderMixin* method), 310

`get_pcoord_len()` (*westpa.oldtools.iframe.WESTDataReaderMixin* method), 303

`get_pcoords()` (*westpa.cli.core.w_succ.WESTDataReaderMixin* method), 59

`get_pcoords()` (*westpa.oldtools.iframe.data_reader.WESTDataReaderMixin* method), 310

`get_pcoords()` (*westpa.oldtools.iframe.WESTDataReaderMixin* method), 303

`get_python_object()` (*westpa.core.yamlcfg.YAMLConfig* method), 239

`get_rates()` (*westpa.westext.weed.weed_driver.WEEDDriver* method), 327

`get_rates()` (*westpa.westext.weed.WEEDDriver* method), 328

`get_rates()` (*westpa.westext.wess.wess_driver.WESSDriver* method), 329

`get_rates()` (*westpa.westext.wess.WESSDriver* method), 329

`get_rates()` (in module *westpa.cli.tools.w_red*), 105

`get_result()` (*westpa.work_managers.core.WMFuture* method), 244

`get_result()` (*westpa.work_managers.mpi.WMFuture* method), 248

`get_result()` (*westpa.work_managers.processes.WMFuture* method), 250

`get_result()` (*westpa.work_managers.serial.WMFuture* method), 253

`get_result()` (*westpa.work_managers.threads.WMFuture* method), 255

`get_result()` (*westpa.work_managers.zeromq.work_manager.WMFuture* method), 270

`get_root_indices()` (*westpa.trajtree.trajtree.TrajTreeSet* method), 300

`get_root_indices()` (*westpa.trajtree.TrajTreeSet* method), 299

`get_roots()` (*westpa.trajtree.trajtree.TrajTreeSet* method), 300

`get_roots()` (*westpa.trajtree.TrajTreeSet* method), 299

`get_seg_ids()` (*westpa.cli.core.w_succ.WESTDataReaderMixin* method), 59

`get_seg_ids()` (*westpa.oldtools.iframe.data_reader.WESTDataReaderMixin* method), 310

`get_seg_ids()` (*westpa.oldtools.iframe.WESTDataReaderMixin* method), 303

`get_seg_index()` (`westpa.cli.core.w_succ.WESTDataReaderMixin` method), 59

`get_seg_index()` (`westpa.core.data_manager.WESTDataManager` method), 195

`get_seg_index()` (`westpa.oldtools.iframe.data_reader.WESTDataReaderMixin` method), 310

`get_seg_index()` (`westpa.oldtools.iframe.WESTDataReaderMixin` method), 311

`get_seg_index()` (`westpa.oldtools.iframe.WESTDataReaderMixin` method), 302

`get_segment_data()` (`westpa.core.h5io.DSSpec` method), 221

`get_segment_data()` (`westpa.core.h5io.SingleSegmentDSSpec` method), 221

`get_segment_data()` (`westpa.tools.data_reader.SingleSegmentDSSpec` method), 289

`get_segment_data_slice()` (`westpa.cli.tools.w_trace.Trace` method), 41

`get_segment_initial_states()` (`westpa.core.data_manager.WESTDataManager` method), 196

`get_segments()` (`westpa.analysis.trajectories.SegmentCollector` method), 339

`get_segments()` (`westpa.cli.core.w_succ.WESTDataReaderMixin` method), 59

`get_segments()` (`westpa.core.data_manager.WESTDataManager` method), 196

`get_segments()` (`westpa.oldtools.iframe.data_reader.WESTDataReaderMixin` method), 310

`get_segments()` (`westpa.oldtools.iframe.WESTDataReaderMixin` method), 311

`get_segments()` (`westpa.oldtools.iframe.WESTDataReaderMixin` method), 302

`get_segments_by_id()` (`westpa.cli.core.w_succ.WESTDataReaderMixin` method), 59

`get_segments_by_id()` (`westpa.oldtools.iframe.data_reader.WESTDataReaderMixin` method), 310

`get_segments_by_id()` (`westpa.oldtools.iframe.WESTDataReaderMixin` method), 302

`get_state()` (`westpa.oldtools.iframe.TransitionEventAccumulator` method), 306

`get_state()` (`westpa.oldtools.iframe.transitions.TransitionEventAccumulator` method), 316

`get_steady_state()` (in module `westpa.core.kinetics.matrates`), 176

`get_summary_table()` (`westpa.cli.core.w_succ.WESTDataReaderMixin` method), 58

`get_summary_table()` (`westpa.oldtools.iframe.data_reader.WESTDataReaderMixin` method), 310

`get_summary_table()` (`westpa.oldtools.iframe.WESTDataReaderMixin` method), 302

`get_target_states()` (`westpa.core.data_manager.WESTDataManager` method), 195

`get_total_time()` (`westpa.cli.core.w_succ.WESTDataReaderMixin` method), 59

`get_total_time()` (`westpa.oldtools.iframe.data_reader.WESTDataReaderMixin` method), 311

`get_total_time()` (`westpa.oldtools.iframe.WESTDataReaderMixin` method), 303

`get_traceback()` (`westpa.work_managers.core.WMFuture` method), 244

`get_traceback()` (`westpa.work_managers.mpi.WMFuture` method), 248

`get_traceback()` (`westpa.work_managers.processes.WMFuture` method), 251

`get_traceback()` (`westpa.work_managers.serial.WMFuture` method), 253

`get_traceback()` (`westpa.work_managers.threads.WMFuture` method), 255

`get_traceback()` (`westpa.work_managers.zeromq.work_manager.WMFuture` method), 255

`get_traj_group()` (`westpa.oldtools.iframe.BFDataManager` method), 304

`get_traj_group()` (`westpa.oldtools.iframe.data_reader.BFDataManager` method), 304

`get_traj_len()` (`westpa.oldtools.iframe.BFDataManager` method), 304

`get_traj_len()` (`westpa.oldtools.iframe.data_reader.BFDataManager` method), 304

`get_trajectory_roots()` (`westpa.oldtools.iframe.TrajWalker` method), 305

`get_trajectory_roots()` (`westpa.oldtools.iframe.trajwalker.TrajWalker` method), 315

`get_trajectory_roots()` (`westpa.oldtools.iframe.transitions.TrajWalker` method), 316

`get_transitions_ds()` (`westpa.oldtools.iframe.TransitionAnalysisMixin` method), 305

`get_transitions_ds()` (`westpa.oldtools.iframe.transitions.TransitionAnalysisMixin` method), 317

`get_typed()` (`westpa.core.yamlcfg.YAMLConfig` method), 239

`get_unused_initial_states()` (`westpa.core.data_manager.WESTDataManager` method), 196

`get_work_manager()` (`westpa.work_managers.environment.WMEnvironment` method), 245

`get_weights()` (`westpa.core.data_manager.WESTDataManager` method), 197

`get_wtg_parent_array()`

(*westpa.cli.core.w_succ.WESTDataReaderMixin* *go()* (*westpa.cli.tools.w_eddist.WESTParallelTool* method), 59 method), 91

get_wtg_parent_array() (*westpa.cli.tools.w_fluxanl.WESTTool* method), 160
(*westpa.oldtools.iframe.data_reader.WESTDataReaderMixin* *go()* (*westpa.cli.tools.w_fluxanl.WFluxanlTool* method), method), 310 method), 162

get_wtg_parent_array() (*westpa.cli.tools.w_ipa.WESTParallelTool* method),
(*westpa.oldtools.iframe.WESTDataReaderMixin* *go()* method), 302 method), 46

getbuffer() (*westpa.core.propagators.executable.BytesIO* *go()* (*westpa.cli.tools.w_ipa.WIPI* method), 49
method), 180 *go()* (*westpa.cli.tools.w_kinavg.DKinAvg* method), 125

getvalue() (*westpa.core.propagators.executable.BytesIO* *go()* (*westpa.cli.tools.w_kinavg.WESTMasterCommand* method), 180 method), 124

go() (*westpa.cli.tools.ploterr.DirectKinetics* method), *go()* (*westpa.cli.tools.w_kinavg.WESTParallelTool* method), 119 method), 124

go() (*westpa.cli.tools.ploterr.DirectStateprobs* method), *go()* (*westpa.cli.tools.w_kinetics.DKinetics* method), 131
119 *go()* (*westpa.cli.tools.w_kinetics.WESTMasterCommand* method), 129

go() (*westpa.cli.tools.ploterr.GenericIntervalSubcommand* *go()* (*westpa.cli.tools.w_kinetics.WESTParallelTool* method), 118 method), 130

go() (*westpa.cli.tools.ploterr.WESTMasterCommand* *go()* (*westpa.cli.tools.w_multi_west.WESTMultiTool* method), 116 method), 103

go() (*westpa.cli.tools.ploterr.WESTSubcommand* *go()* (*westpa.cli.tools.w_multi_west.WESTTool* method), method), 117 method), 102

go() (*westpa.cli.tools.plothist.AveragePlotHist* method), *go()* (*westpa.cli.tools.w_multi_west.WMultiWest* method), 114 method), 103

go() (*westpa.cli.tools.plothist.EvolutionPlotHist* *go()* (*westpa.cli.tools.w_ntop.WESTTool* method), 96
method), 115 *go()* (*westpa.cli.tools.w_ntop.WNTopTool* method), 99

go() (*westpa.cli.tools.plothist.InstantPlotHist* method), *go()* (*westpa.cli.tools.w_pdist.WESTParallelTool* method), 114 method), 52

go() (*westpa.cli.tools.plothist.WESTMasterCommand* *go()* (*westpa.cli.tools.w_pdist.WPDist* method), 56
method), 113 *go()* (*westpa.cli.tools.w_postanalysis_matrix.RWMatrix* method), 144

go() (*westpa.cli.tools.plothist.WESTSubcommand* *go()* (*westpa.cli.tools.w_postanalysis_matrix.WESTMasterCommand* method), 113 method), 143

go() (*westpa.cli.tools.w_assign.WAssign* method), 35 *go()* (*westpa.cli.tools.w_postanalysis_matrix.WESTParallelTool* method), 29 method), 143

go() (*westpa.cli.tools.w_bins.WBinTool* method), 21 *go()* (*westpa.cli.tools.w_postanalysis_reweight.RWAverage* method), 147

go() (*westpa.cli.tools.w_bins.WESTTool* method), 19 *go()* (*westpa.cli.tools.w_postanalysis_reweight.WESTMasterCommand* method), 145

go() (*westpa.cli.tools.w_crawl.WCrawl* method), 65 *go()* (*westpa.cli.tools.w_postanalysis_reweight.WESTParallelTool* method), 146

go() (*westpa.cli.tools.w_crawl.WESTParallelTool* *go()* (*westpa.cli.tools.w_red.WESTParallelTool* method),
method), 63 method), 105

go() (*westpa.cli.tools.w_direct.DAll* method), 74 *go()* (*westpa.cli.tools.w_red.WRed* method), 106

go() (*westpa.cli.tools.w_direct.DAverage* method), 74 *go()* (*westpa.cli.tools.w_reweight.RWAll* method), 155

go() (*westpa.cli.tools.w_direct.DKinAvg* method), 72 *go()* (*westpa.cli.tools.w_reweight.RWAverage* method), 155

go() (*westpa.cli.tools.w_direct.DKinetics* method), 71 *go()* (*westpa.cli.tools.w_reweight.RWMatrix* method), 150

go() (*westpa.cli.tools.w_direct.DStateProbs* method), 74 *go()* (*westpa.cli.tools.w_reweight.RWRate* method), 152

go() (*westpa.cli.tools.w_direct.WESTMasterCommand* *go()* (*westpa.cli.tools.w_reweight.RWStateProbs* method), 68 method), 154

go() (*westpa.cli.tools.w_direct.WESTParallelTool* *go()* (*westpa.cli.tools.w_reweight.WESTMasterCommand* method), 69 method), 147

go() (*westpa.cli.tools.w_dumpsegs.WDumpSegs* *go()* (*westpa.cli.tools.w_reweight.WEDDist* method), 94
method), 142

go() (*westpa.cli.tools.w_dumpsegs.WESTTool* method), 140

[go\(\)](#) ([westpa.cli.tools.w_reweight.WESTParallelTool method](#)), 148
[go\(\)](#) ([westpa.cli.tools.w_select.WESTParallelTool method](#)), 78
[go\(\)](#) ([westpa.cli.tools.w_select.WSelectTool method](#)), 81
[go\(\)](#) ([westpa.cli.tools.w_stateprobs.DStateProbs method](#)), 138
[go\(\)](#) ([westpa.cli.tools.w_stateprobs.WESTMasterCommand method](#)), 136
[go\(\)](#) ([westpa.cli.tools.w_stateprobs.WESTParallelTool method](#)), 136
[go\(\)](#) ([westpa.cli.tools.w_trace.WESTTool method](#)), 38
[go\(\)](#) ([westpa.cli.tools.w_trace.WTraceTool method](#)), 42
[go\(\)](#) ([westpa.tools.core.WESTMasterCommand method](#)), 288
[go\(\)](#) ([westpa.tools.core.WESTMultiTool method](#)), 287
[go\(\)](#) ([westpa.tools.core.WESTParallelTool method](#)), 286
[go\(\)](#) ([westpa.tools.core.WESTSubcommand method](#)), 287
[go\(\)](#) ([westpa.tools.core.WESTTool method](#)), 286
[go\(\)](#) ([westpa.tools.kinetics_tool.WESTSubcommand method](#)), 293
[go\(\)](#) ([westpa.tools.WESTMasterCommand method](#)), 278
[go\(\)](#) ([westpa.tools.WESTMultiTool method](#)), 278
[go\(\)](#) ([westpa.tools.WESTParallelTool method](#)), 277
[go\(\)](#) ([westpa.tools.WESTSubcommand method](#)), 277
[go\(\)](#) ([westpa.tools.WESTTool method](#)), 276
[group_description](#) ([westpa.work_managers.environment.WMEnvironment attribute](#)), 245
[group_name](#) ([westpa.cli.tools.w_assign.WESTDSSynthesizer attribute](#)), 30
[group_name](#) ([westpa.cli.tools.w_pdist.WESTDSSynthesizer attribute](#)), 52
[group_name](#) ([westpa.cli.tools.w_pdist.WESTWDSSynthesizer attribute](#)), 53
[group_name](#) ([westpa.tools.data_reader.WESTDSSynthesizer attribute](#)), 289
[group_name](#) ([westpa.tools.data_reader.WESTWDSSynthesizer attribute](#)), 289
[group_name](#) ([westpa.tools.WESTDSSynthesizer attribute](#)), 279
[group_name](#) ([westpa.tools.WESTWDSSynthesizer attribute](#)), 279
[group_title](#) ([westpa.work_managers.environment.WMEnvironment attribute](#)), 245
[grouper\(\)](#) (in module [westpa.core.binning.mab_manager](#)), 171
[grouper\(\)](#) (in module [westpa.core.kinetics.rate_averaging](#)), 178
[grouper\(\)](#) (in module [westpa.core.sim_manager](#)), 228

H

[H5File](#) (in module [westpa.cli.tools.w_red](#)), 105
[h5file](#) ([westpa.core.h5io.FileLinkedDSSpec property](#)), 221
[h5group](#) ([westpa.analysis.core.Iteration property](#)), 333
[handle_reconfigure_timeout\(\)](#) ([westpa.work_managers.zeromq.work_manager.ZMQWorker method](#)), 268
[handle_reconfigure_timeout\(\)](#) ([westpa.work_managers.zeromq.worker.ZMQWorker method](#)), 275
[handle_reconfigure_timeout\(\)](#) ([westpa.work_managers.zeromq.ZMQWorker method](#)), 258
[handle_result\(\)](#) ([westpa.work_managers.zeromq.work_manager.ZMQWorker method](#)), 268
[handle_result\(\)](#) ([westpa.work_managers.zeromq.work_manager.ZMQWorker method](#)), 272
[handle_result\(\)](#) ([westpa.work_managers.zeromq.worker.ZMQWorker method](#)), 275
[handle_result\(\)](#) ([westpa.work_managers.zeromq.ZMQWorker method](#)), 258
[handle_result\(\)](#) ([westpa.work_managers.zeromq.ZMQWorkManager method](#)), 259
[handle_task_request\(\)](#) ([westpa.work_managers.zeromq.work_manager.ZMQWorkManager method](#)), 272
[handle_task_request\(\)](#) ([westpa.work_managers.zeromq.ZMQWorkManager method](#)), 259
[has_pointer\(\)](#) ([westpa.core.h5io.WESTIterationFile method](#)), 220
[has_restart\(\)](#) ([westpa.core.h5io.WESTIterationFile method](#)), 220
[has_target_states](#) ([westpa.analysis.core.Iteration property](#)), 335
[has_topology\(\)](#) ([westpa.core.h5io.WESTIterationFile method](#)), 220
[hash_args\(\)](#) ([westpa.cli.tools.w_ipa.WIPI method](#)), 48
[hashfunc\(\)](#) ([westpa.core.binning.assign.BinMapper method](#)), 166
[HDF5MDTrajectory](#) (class in [westpa.analysis.trajectories](#)), 339
[HDF5TrajectoryFile](#) (class in [westpa.core.h5io](#)), 211
[help](#) ([westpa.cli.tools.w_ipa.WIPI property](#)), 49
[help_text](#) ([westpa.cli.tools.ploterr.DirectKinetics attribute](#)), 118
[help_text](#) ([westpa.cli.tools.ploterr.DirectStateprobs attribute](#)), 119
[help_text](#) ([westpa.cli.tools.ploterr.GenericIntervalSubcommand attribute](#)), 118
[help_text](#) ([westpa.cli.tools.ploterr.ReweightKinetics attribute](#)), 119
[help_text](#) ([westpa.cli.tools.ploterr.ReweightStateprobs attribute](#)), 119
[help_text](#) ([westpa.cli.tools.ploterr.WESTSubcommand attribute](#)), 119

- attribute), 117
- help_text (westpa.cli.tools.plothist.AveragePlotHist attribute), 114
- help_text (westpa.cli.tools.plothist.EvolutionPlotHist attribute), 115
- help_text (westpa.cli.tools.plothist.InstantPlotHist attribute), 114
- help_text (westpa.cli.tools.plothist.WESTSubcommand attribute), 113
- help_text (westpa.cli.tools.w_direct.DAll attribute), 74
- help_text (westpa.cli.tools.w_direct.DAverage attribute), 74
- help_text (westpa.cli.tools.w_direct.DKinAvg attribute), 71
- help_text (westpa.cli.tools.w_direct.DKinetics attribute), 71
- help_text (westpa.cli.tools.w_direct.DStateProbs attribute), 73
- help_text (westpa.cli.tools.w_kinavg.DKinAvg attribute), 124
- help_text (westpa.cli.tools.w_kinavg.WKinAvg attribute), 126
- help_text (westpa.cli.tools.w_kinetics.DKinetics attribute), 130
- help_text (westpa.cli.tools.w_kinetics.WKinetics attribute), 131
- help_text (westpa.cli.tools.w_postanalysis_matrix.PAMatrix attribute), 144
- help_text (westpa.cli.tools.w_postanalysis_matrix.RWMatrix attribute), 143
- help_text (westpa.cli.tools.w_postanalysis_reweight.PAAverage attribute), 147
- help_text (westpa.cli.tools.w_postanalysis_reweight.RWAverage attribute), 146
- help_text (westpa.cli.tools.w_reweight.RWAll attribute), 155
- help_text (westpa.cli.tools.w_reweight.RWAverage attribute), 155
- help_text (westpa.cli.tools.w_reweight.RWMatrix attribute), 149
- help_text (westpa.cli.tools.w_reweight.RWRate attribute), 151
- help_text (westpa.cli.tools.w_reweight.RWReweight attribute), 150
- help_text (westpa.cli.tools.w_reweight.RWStateProbs attribute), 153
- help_text (westpa.cli.tools.w_stateprobs.DStateProbs attribute), 137
- help_text (westpa.cli.tools.w_stateprobs.WStateProbs attribute), 138
- help_text (westpa.tools.core.WESTSubcommand attribute), 287
- help_text (westpa.tools.kinetics_tool.WESTSubcommand attribute), 293
- help_text (westpa.tools.WESTSubcommand attribute), 277
- histnd() (in module westpa.cli.tools.w_eddist), 92
- histnd() (in module westpa.cli.tools.w_pdist), 54
- histnd() (in module westpa.fasthist), 298
- |
- idempotent_announcement_messages (westpa.work_managers.zeromq.core.Message attribute), 260
- idempotent_announcement_messages (westpa.work_managers.zeromq.node.Message attribute), 264
- idempotent_announcement_messages (westpa.work_managers.zeromq.work_manager.Message attribute), 267
- idempotent_announcement_messages (westpa.work_managers.zeromq.worker.Message attribute), 274
- IDENTIFY (westpa.work_managers.zeromq.core.Message attribute), 260
- IDENTIFY (westpa.work_managers.zeromq.node.Message attribute), 264
- IDENTIFY (westpa.work_managers.zeromq.work_manager.Message attribute), 267
- IDENTIFY (westpa.work_managers.zeromq.worker.Message attribute), 274
- identify() (westpa.work_managers.zeromq.work_manager.ZMQWorker method), 268
- identify() (westpa.work_managers.zeromq.worker.ZMQWorker method), 275
- identify() (westpa.work_managers.zeromq.ZMQWorker method), 258
- image_molecules() (westpa.core.h5io.Trajectory method), 209
- import_() (in module westpa.core.h5io), 210
- in_units_of() (in module westpa.core.h5io), 209
- include_arg() (westpa.tools.binning.WESTToolComponent method), 283
- include_arg() (westpa.tools.core.WESTToolComponent method), 285
- include_arg() (westpa.tools.data_reader.WESTToolComponent method), 288
- include_arg() (westpa.tools.iter_range.WESTToolComponent method), 290
- include_arg() (westpa.tools.progress.WESTToolComponent method), 295
- include_arg() (westpa.tools.selected_segs.WESTToolComponent method), 295
- include_arg() (westpa.tools.WESTToolComponent method), 277
- include_help_command (westpa.cli.tools.ploterr.WESTMasterCommand attribute), 116

`include_help_command` (static method), 85
(`westpa.cli.tools.plothist.WESTMasterCommand` attribute), 112

`include_help_command` (`westpa.cli.tools.w_direct.WESTMasterCommand` attribute), 68

`include_help_command` (`westpa.cli.tools.w_kinavg.WESTMasterCommand` attribute), 123

`include_help_command` (`westpa.cli.tools.w_kinetics.WESTMasterCommand` attribute), 129

`include_help_command` (`westpa.cli.tools.w_postanalysis_matrix.WESTMasterCommand` attribute), 142

`include_help_command` (`westpa.cli.tools.w_postanalysis_reweight.WESTMasterCommand` attribute), 145

`include_help_command` (`westpa.cli.tools.w_reweight.WESTMasterCommand` attribute), 147

`include_help_command` (`westpa.cli.tools.w_stateprobs.WESTMasterCommand` attribute), 136

`include_help_command` (`westpa.tools.core.WESTMasterCommand` attribute), 288

`include_help_command` (`westpa.tools.WESTMasterCommand` attribute), 278

`incorporate()` (`westpa.oldtools.stats.accumulator.RunningStatsAccumulator` method), 318

`incorporate()` (`westpa.oldtools.stats.RunningStatsAccumulator` method), 317

`index()` (`westpa.work_managers.mpi.deque` method), 246

`index()` (`westpa.work_managers.zeromq.work_manager.deque` method), 271

`index_dtype` (in module `westpa.cli.tools.w_assign`), 28

`index_dtype` (in module `westpa.core.binning`), 165

`index_dtype` (in module `westpa.core.binning.assign`), 166

`index_dtype` (in module `westpa.core.kinetics.events`), 175

`index_dtype` (in module `westpa.core.reweight.matrix`), 187

`index_dtype` (`westpa.oldtools.iframe.TransitionEventAccumulator` attribute), 306

`index_dtype` (`westpa.oldtools.iframe.transitions.TransitionEventAccumulator` attribute), 316

`initial` (`westpa.analysis.core.Walker` property), 337

`initial_pcoord()` (`westpa.cli.core.w_fork.Segment` static method), 25

`initial_pcoord()` (`westpa.cli.core.w_states.Segment` static method), 85

`initial_pcoord()` (`westpa.cli.core.w_succ.Segment` static method), 58

`initial_pcoord()` (`westpa.cli.tools.w_dumpsegs.Segment` static method), 142

`initial_pcoord()` (`westpa.cli.tools.w_trace.Segment` static method), 39

`initial_pcoord()` (`westpa.core.binning.mab_manager.Segment` static method), 173

`initial_pcoord()` (`westpa.core.data_manager.Segment` static method), 190

`initial_pcoord()` (`westpa.core.propagators.executable.Segment` static method), 184

`initial_pcoord()` (`westpa.core.segment.Segment` static method), 225

`initial_pcoord()` (`westpa.core.sim_manager.Segment` static method), 227

`initial_pcoord()` (`westpa.core.states.Segment` static method), 230

`initial_pcoord()` (`westpa.core.we_driver.Segment` static method), 235

`initial_pcoord()` (`westpa.oldtools.iframe.data_reader.Segment` static method), 309

`initial_state_id` (`westpa.cli.core.w_fork.Segment` property), 25

`initial_state_id` (`westpa.cli.core.w_states.Segment` property), 85

`initial_state_id` (`westpa.cli.core.w_succ.Segment` property), 58

`initial_state_id` (`westpa.cli.tools.w_dumpsegs.Segment` property), 142

`initial_state_id` (`westpa.cli.tools.w_trace.Segment` property), 39

`initial_state_id` (`westpa.core.binning.mab_manager.Segment` property), 173

`initial_state_id` (`westpa.core.data_manager.Segment` property), 190

`initial_state_id` (`westpa.core.propagators.executable.Segment` property), 184

`initial_state_id` (`westpa.core.segment.Segment` property), 225

`initial_state_id` (`westpa.core.sim_manager.Segment` property), 227

`initial_state_id` (`westpa.core.states.Segment` property), 230

`initial_state_id` (`westpa.core.we_driver.Segment` property), 235

`initial_state_id` (`westpa.oldtools.iframe.data_reader.Segment` property), 309

`initial_walkers` (`westpa.analysis.core.Iteration` property), 334

`initial_walkers` (`westpa.analysis.core.Run` property), 332

`initialize()` (in module `westpa.cli.core.w_init`), 16

[initialize\(\) \(in module westpa.cli.core.w_states\), 87](#)
[initialize\(\) \(westpa.cli.tools.w_crawl.WESTPACrawler method\), 65](#)
[initialize\(\) \(westpa.core.systems.WESTSystem method\), 233](#)
[initialize\(\) \(westpa.core.yamlcfg.YAMLSystem method\), 240](#)
[initialize_simulation\(\) \(westpa.core.binning.BinlessSimManager method\), 164](#)
[initialize_simulation\(\) \(westpa.core.binning.mab_manager.MABSimManager method\), 174](#)
[initialize_simulation\(\) \(westpa.core.binning.mab_manager.WESimManager method\), 170](#)
[initialize_simulation\(\) \(westpa.core.binning.MABSimManager method\), 164](#)
[initialize_simulation\(\) \(westpa.core.sim_manager.WESimManager method\), 229](#)
[InitialState \(class in westpa.cli.core.w_fork\), 25](#)
[InitialState \(class in westpa.cli.tools.w_trace\), 39](#)
[InitialState \(class in westpa.core.binning.mab_manager\), 171](#)
[InitialState \(class in westpa.core.data_manager\), 191](#)
[InitialState \(class in westpa.core.propagators.executable\), 182](#)
[InitialState \(class in westpa.core.sim_manager\), 227](#)
[InitialState \(class in westpa.core.states\), 231](#)
[InitialState \(class in westpa.core.we_driver\), 235](#)
[initpoint_type \(westpa.cli.core.w_fork.Segment property\), 25](#)
[initpoint_type \(westpa.cli.core.w_states.Segment property\), 85](#)
[initpoint_type \(westpa.cli.core.w_succ.Segment property\), 58](#)
[initpoint_type \(westpa.cli.tools.w_dumpsegs.Segment property\), 142](#)
[initpoint_type \(westpa.cli.tools.w_trace.Segment property\), 39](#)
[initpoint_type \(westpa.core.binning.mab_manager.Segment property\), 173](#)
[initpoint_type \(westpa.core.data_manager.Segment property\), 190](#)
[initpoint_type \(westpa.core.propagators.executable.Segment property\), 184](#)
[initpoint_type \(westpa.core.segment.Segment property\), 225](#)
[initpoint_type \(westpa.core.sim_manager.Segment property\), 227](#)
[initpoint_type \(westpa.core.states.Segment property\), 230](#)
[initpoint_type \(westpa.core.we_driver.Segment property\), 235](#)
[initpoint_type \(westpa.oldtools.iframe.data_reader.Segment property\), 309](#)
[initpoint_type_names \(westpa.cli.core.w_fork.Segment attribute\), 24](#)
[initpoint_type_names \(westpa.cli.core.w_states.Segment attribute\), 85](#)
[initpoint_type_names \(westpa.cli.core.w_succ.Segment attribute\), 58](#)
[initpoint_type_names \(westpa.cli.tools.w_dumpsegs.Segment attribute\), 141](#)
[initpoint_type_names \(westpa.cli.tools.w_trace.Segment attribute\), 39](#)
[initpoint_type_names \(westpa.core.binning.mab_manager.Segment attribute\), 173](#)
[initpoint_type_names \(westpa.core.data_manager.Segment attribute\), 189](#)
[initpoint_type_names \(westpa.core.propagators.executable.Segment attribute\), 184](#)
[initpoint_type_names \(westpa.core.segment.Segment attribute\), 225](#)
[initpoint_type_names \(westpa.core.sim_manager.Segment attribute\), 227](#)
[initpoint_type_names \(westpa.core.states.Segment attribute\), 230](#)
[initpoint_type_names \(westpa.core.we_driver.Segment attribute\), 235](#)
[initpoint_type_names \(westpa.oldtools.iframe.data_reader.Segment attribute\), 309](#)
[initpoint_types \(westpa.cli.core.w_fork.Segment attribute\), 24](#)
[initpoint_types \(westpa.cli.core.w_states.Segment attribute\), 85](#)
[initpoint_types \(westpa.cli.core.w_succ.Segment attribute\), 57](#)
[initpoint_types \(westpa.cli.tools.w_dumpsegs.Segment attribute\), 141](#)
[initpoint_types \(westpa.cli.tools.w_trace.Segment attribute\), 39](#)
[initpoint_types \(westpa.core.binning.mab_manager.Segment attribute\), 173](#)

`initpoint_types` (`westpa.core.data_manager.Segment` attribute), 189

`initpoint_types` (`westpa.core.propagators.executable.Segment` attribute), 184

`initpoint_types` (`westpa.core.segment.Segment` attribute), 225

`initpoint_types` (`westpa.core.sim_manager.Segment` attribute), 227

`initpoint_types` (`westpa.core.states.Segment` attribute), 230

`initpoint_types` (`westpa.core.we_driver.Segment` attribute), 235

`initpoint_types` (`westpa.oldtools.iframe.data_reader.Segment` attribute), 309

`input_filename` (`westpa.cli.tools.ploterr.DirectKinetics` attribute), 118

`input_filename` (`westpa.cli.tools.ploterr.DirectStateprobs` attribute), 119

`input_filename` (`westpa.cli.tools.ploterr.ReweightKinetics` attribute), 119

`input_filename` (`westpa.cli.tools.ploterr.ReweightStateprobs` attribute), 119

`insert()` (`westpa.work_managers.mpi.dequeue` method), 246

`insert()` (`westpa.work_managers.zeromq.work_manager.dequeue` method), 271

`install_sigint_handler()` (`westpa.work_managers.core.WorkManager` method), 243

`install_sigint_handler()` (`westpa.work_managers.mpi.WorkManager` method), 247

`install_sigint_handler()` (`westpa.work_managers.processes.WorkManager` method), 249

`install_sigint_handler()` (`westpa.work_managers.serial.WorkManager` method), 252

`install_sigint_handler()` (`westpa.work_managers.threads.WorkManager` method), 254

`install_sigint_handler()` (`westpa.work_managers.zeromq.core.ZMQCore` method), 262

`install_sigint_handler()` (`westpa.work_managers.zeromq.node.ZMQCore` method), 264

`install_sigint_handler()` (`westpa.work_managers.zeromq.work_manager.WorkManager` method), 269

`install_sigint_handler()` (`westpa.work_managers.zeromq.work_manager.ZMQCore` method), 267

`install_sigint_handler()`

`install_sigint_handler()` (`westpa.work_managers.zeromq.worker.ZMQCore` method), 274

`install_sigint_handler()` (`westpa.work_managers.zeromq.ZMQCore` method), 257

`install_signal_handlers()` (`westpa.work_managers.zeromq.ZMQCore` method), 257

`install_signal_handlers()` (`westpa.work_managers.zeromq.core.ZMQCore` method), 262

`install_signal_handlers()` (`westpa.work_managers.zeromq.node.ZMQCore` method), 264

`install_signal_handlers()` (`westpa.work_managers.zeromq.work_manager.ZMQCore` method), 267

`install_signal_handlers()` (`westpa.work_managers.zeromq.work_manager.ZMQWorker` method), 269

`install_signal_handlers()` (`westpa.work_managers.zeromq.worker.ZMQCore` method), 274

`install_signal_handlers()` (`westpa.work_managers.zeromq.worker.ZMQWorker` method), 275

`install_signal_handlers()` (`westpa.work_managers.zeromq.ZMQCore` method), 257

`install_signal_handlers()` (`westpa.work_managers.zeromq.ZMQWorker` method), 258

`InstantPlotHist` (class in `westpa.cli.tools.plothist`), 114

`internal_transport` (`westpa.work_managers.zeromq.core.ZMQCore` attribute), 261

`internal_transport` (`westpa.work_managers.zeromq.node.ZMQCore` attribute), 263

`internal_transport` (`westpa.work_managers.zeromq.work_manager.ZMQCore` attribute), 266

`internal_transport` (`westpa.work_managers.zeromq.worker.ZMQCore` attribute), 273

`internal_transport` (`westpa.work_managers.zeromq.ZMQCore` attribute), 256

`intersection()` (`westpa.analysis.core.BinUnion` method), 337

`introduction` (`westpa.cli.tools.w_ipa.WIPI` property), 49

`invoke_callbacks()` (`westpa.core.binning.mab_manager.WESimManager` method), 170

`invoke_callbacks()` (`westpa.core.sim_manager.WESimManager` method), 228

`is_done()` (`westpa.work_managers.core.WMFuture` method), 244

`is_done()` (`westpa.work_managers.mpi.WMFuture` method), 248

`is_done()` (`westpa.work_managers.processes.WMFuture` method), 248

<i>method</i>), 251	ISTATE_STATUS_FAILED
<code>is_done()</code> (<i>westpa.work_managers.serial.WMFuture</i> <i>method</i>), 253	(<i>westpa.cli.core.w_fork.InitialState</i> <i>attribute</i>), 26
<code>is_done()</code> (<i>westpa.work_managers.threads.WMFuture</i> <i>method</i>), 255	ISTATE_STATUS_FAILED
<code>is_done()</code> (<i>westpa.work_managers.zeromq.work_manager.WMFuture</i> <i>method</i>), 270	(<i>westpa.cli.tools.w_trace.InitialState</i> <i>attribute</i>), 40
<code>is_master</code> (<i>westpa.work_managers.core.WorkManager</i> <i>property</i>), 244	ISTATE_STATUS_FAILED
<code>is_master</code> (<i>westpa.work_managers.mpi.Worker</i> <i>property</i>), 249	(<i>westpa.core.binning.mab_manager.InitialState</i> <i>attribute</i>), 172
<code>is_master</code> (<i>westpa.work_managers.mpi.WorkManager</i> <i>property</i>), 247	ISTATE_STATUS_FAILED
<code>is_master</code> (<i>westpa.work_managers.processes.WorkManager</i> <i>property</i>), 250	(<i>westpa.core.data_manager.InitialState</i> <i>attribute</i>), 192
<code>is_master</code> (<i>westpa.work_managers.serial.WorkManager</i> <i>property</i>), 252	ISTATE_STATUS_FAILED
<code>is_master</code> (<i>westpa.work_managers.threads.WorkManager</i> <i>property</i>), 254	(<i>westpa.core.propagators.executable.InitialState</i> <i>attribute</i>), 183
<code>is_master</code> (<i>westpa.work_managers.zeromq.node.ZMQNode</i> <i>property</i>), 265	ISTATE_STATUS_FAILED
<code>is_master</code> (<i>westpa.work_managers.zeromq.work_manager.WorkManager</i> <i>property</i>), 270	(<i>westpa.core.sim_manager.InitialState</i> <i>attribute</i>), 228
<code>is_master</code> (<i>westpa.work_managers.zeromq.work_manager.ZMQNode</i> <i>property</i>), 269	ISTATE_STATUS_FAILED (<i>westpa.core.states.InitialState</i> <i>attribute</i>), 232
<code>is_master</code> (<i>westpa.work_managers.zeromq.work_manager.ZMQWorker</i> <i>property</i>), 268	ISTATE_STATUS_FAILED
<code>is_master</code> (<i>westpa.work_managers.zeromq.worker.ZMQWorker</i> <i>property</i>), 275	(<i>westpa.core.we_driver.InitialState</i> <i>attribute</i>), 236
<code>is_master</code> (<i>westpa.work_managers.zeromq.ZMQNode</i> <i>property</i>), 258	<i>istate_status_names</i>
<code>is_master</code> (<i>westpa.work_managers.zeromq.ZMQWorker</i> <i>property</i>), 258	(<i>westpa.cli.core.w_fork.InitialState</i> <i>attribute</i>), 26
<code>is_numpy()</code> (<i>westpa.oldtools.iframe.data_reader.ExtDataReaderMixin</i> <i>method</i>), 311	<i>istate_status_names</i>
<code>is_numpy()</code> (<i>westpa.oldtools.iframe.ExtDataReaderMixin</i> <i>method</i>), 303	(<i>westpa.cli.tools.w_trace.InitialState</i> <i>attribute</i>), 40
<code>is_within_directory()</code> (<i>in module westpa.core.h5io</i>), 216	<i>istate_status_names</i>
<code>isatty()</code> (<i>westpa.core.propagators.executable.BytesIO</i> <i>method</i>), 180	(<i>westpa.core.binning.mab_manager.InitialState</i> <i>attribute</i>), 172
<code>isiterable()</code> (<i>in module westpa.cli.tools.w_eddist</i>), 92	<i>istate_status_names</i>
<code>isiterable()</code> (<i>in module westpa.cli.tools.w_pdist</i>), 54	(<i>westpa.core.data_manager.InitialState</i> <i>attribute</i>), 192
<code>islice</code> (<i>class in westpa.work_managers.core</i>), 242	<i>istate_status_names</i>
<code>IsNode</code> (<i>class in westpa.work_managers.zeromq.core</i>), 262	(<i>westpa.core.propagators.executable.InitialState</i> <i>attribute</i>), 183
<code>IsNode</code> (<i>class in westpa.work_managers.zeromq.node</i>), 265	<i>istate_status_names</i>
<code>IsNode</code> (<i>class in westpa.work_managers.zeromq.work_manager</i>), 268	(<i>westpa.core.sim_manager.InitialState</i> <i>attribute</i>), 228
<code>istate</code> (<i>westpa.cli.tools.w_red.RateCalculator</i> <i>property</i>), 106	<i>istate_status_names</i> (<i>westpa.core.states.InitialState</i> <i>attribute</i>), 232
<code>istate_status_dtype</code> (<i>in module westpa.core.data_manager</i>), 194	<i>istate_status_names</i>
	(<i>westpa.core.we_driver.InitialState</i> <i>attribute</i>), 236
	ISTATE_STATUS_PENDING
	(<i>westpa.cli.core.w_fork.InitialState</i> <i>attribute</i>), 25
	ISTATE_STATUS_PENDING
	(<i>westpa.cli.tools.w_trace.InitialState</i> <i>attribute</i>), 40
	ISTATE_STATUS_PENDING
	(<i>westpa.core.binning.mab_manager.InitialState</i> <i>attribute</i>), 172

attribute), 172
 ISTATE_STATUS_PENDING (westpa.core.data_manager.InitialState attribute), 192
 ISTATE_STATUS_PENDING (westpa.core.propagators.executable.InitialState attribute), 183
 ISTATE_STATUS_PENDING (westpa.core.sim_manager.InitialState attribute), 228
 ISTATE_STATUS_PENDING (westpa.core.states.InitialState attribute), 232
 ISTATE_STATUS_PENDING (westpa.core.we_driver.InitialState attribute), 236
 ISTATE_STATUS_PREPARED (westpa.cli.core.w_fork.InitialState attribute), 25
 ISTATE_STATUS_PREPARED (westpa.cli.tools.w_trace.InitialState attribute), 40
 ISTATE_STATUS_PREPARED (westpa.core.binning.mab_manager.InitialState attribute), 172
 ISTATE_STATUS_PREPARED (westpa.core.data_manager.InitialState attribute), 192
 ISTATE_STATUS_PREPARED (westpa.core.propagators.executable.InitialState attribute), 183
 ISTATE_STATUS_PREPARED (westpa.core.sim_manager.InitialState attribute), 228
 ISTATE_STATUS_PREPARED (westpa.core.states.InitialState attribute), 232
 ISTATE_STATUS_PREPARED (westpa.core.we_driver.InitialState attribute), 236
 istate_statuses (westpa.cli.core.w_fork.InitialState attribute), 26
 istate_statuses (westpa.cli.tools.w_trace.InitialState attribute), 40
 istate_statuses (westpa.core.binning.mab_manager.InitialState attribute), 172
 istate_statuses (westpa.core.data_manager.InitialState attribute), 192
 istate_statuses (westpa.core.propagators.executable.InitialState attribute), 183
 istate_statuses (westpa.core.sim_manager.InitialState attribute), 228
 istate_statuses (westpa.core.states.InitialState attribute), 232
 istate_statuses (westpa.core.we_driver.InitialState attribute), 236
 ISTATE_TYPE_BASIS (westpa.cli.core.w_fork.InitialState attribute), 25
 ISTATE_TYPE_BASIS (westpa.cli.tools.w_trace.InitialState attribute), 40
 ISTATE_TYPE_BASIS (westpa.core.binning.mab_manager.InitialState attribute), 172
 ISTATE_TYPE_BASIS (westpa.core.data_manager.InitialState attribute), 191
 ISTATE_TYPE_BASIS (westpa.core.propagators.executable.InitialState attribute), 183
 ISTATE_TYPE_BASIS (westpa.core.sim_manager.InitialState attribute), 228
 ISTATE_TYPE_BASIS (westpa.core.states.InitialState attribute), 232
 ISTATE_TYPE_BASIS (westpa.core.we_driver.InitialState attribute), 236
 istate_type_dtype (in module westpa.core.data_manager), 194
 ISTATE_TYPE_GENERATED (westpa.cli.core.w_fork.InitialState attribute), 25
 ISTATE_TYPE_GENERATED (westpa.cli.tools.w_trace.InitialState attribute), 40
 ISTATE_TYPE_GENERATED (westpa.core.binning.mab_manager.InitialState attribute), 172
 ISTATE_TYPE_GENERATED (westpa.core.data_manager.InitialState attribute), 191
 ISTATE_TYPE_GENERATED (westpa.core.propagators.executable.InitialState attribute), 183
 ISTATE_TYPE_GENERATED (westpa.core.sim_manager.InitialState attribute), 228
 ISTATE_TYPE_GENERATED (westpa.core.states.InitialState attribute), 232
 ISTATE_TYPE_GENERATED (westpa.core.we_driver.InitialState attribute), 236
 istate_type_names (westpa.cli.core.w_fork.InitialState attribute), 26
 istate_type_names (westpa.cli.tools.w_trace.InitialState attribute), 40
 istate_type_names (westpa.core.binning.mab_manager.InitialState attribute), 172
 istate_type_names (westpa.core.data_manager.InitialState attribute), 192
 istate_type_names (westpa.core.propagators.executable.InitialState attribute), 183

`istate_type_names` (`westpa.core.sim_manager.InitialState` attribute), 183
`istate_type_names` (`westpa.core.sim_manager.InitialState` attribute), 228
`istate_type_names` (`westpa.core.states.InitialState` attribute), 232
`istate_type_names` (`westpa.core.we_driver.InitialState` attribute), 236
`ISTATE_TYPE_RESTART` (`westpa.cli.core.w_fork.InitialState` attribute), 25
`ISTATE_TYPE_RESTART` (`westpa.cli.tools.w_trace.InitialState` attribute), 40
`ISTATE_TYPE_RESTART` (`westpa.core.binning.mab_manager.InitialState` attribute), 172
`ISTATE_TYPE_RESTART` (`westpa.core.data_manager.InitialState` attribute), 191
`ISTATE_TYPE_RESTART` (`westpa.core.propagators.executable.InitialState` attribute), 183
`ISTATE_TYPE_RESTART` (`westpa.core.sim_manager.InitialState` attribute), 228
`ISTATE_TYPE_RESTART` (`westpa.core.states.InitialState` attribute), 232
`ISTATE_TYPE_RESTART` (`westpa.core.we_driver.InitialState` attribute), 236
`ISTATE_TYPE_START` (`westpa.cli.core.w_fork.InitialState` attribute), 25
`ISTATE_TYPE_START` (`westpa.cli.tools.w_trace.InitialState` attribute), 40
`ISTATE_TYPE_START` (`westpa.core.binning.mab_manager.InitialState` attribute), 172
`ISTATE_TYPE_START` (`westpa.core.data_manager.InitialState` attribute), 191
`ISTATE_TYPE_START` (`westpa.core.propagators.executable.InitialState` attribute), 183
`ISTATE_TYPE_START` (`westpa.core.sim_manager.InitialState` attribute), 228
`ISTATE_TYPE_START` (`westpa.core.states.InitialState` attribute), 232
`ISTATE_TYPE_START` (`westpa.core.we_driver.InitialState` attribute), 236
`ISTATE_TYPE_UNSET` (`westpa.cli.core.w_fork.InitialState` attribute), 25
`ISTATE_TYPE_UNSET` (`westpa.cli.tools.w_trace.InitialState` attribute), 40
`ISTATE_TYPE_UNSET` (`westpa.core.binning.mab_manager.InitialState` attribute), 172
`ISTATE_TYPE_UNSET` (`westpa.core.data_manager.InitialState` attribute), 191
`ISTATE_TYPE_UNSET` (`westpa.core.propagators.executable.InitialState` attribute), 183
`ISTATE_TYPE_UNSET` (`westpa.core.sim_manager.InitialState` attribute), 228
`ISTATE_TYPE_UNSET` (`westpa.core.states.InitialState` attribute), 232
`ISTATE_TYPE_UNSET` (`westpa.core.we_driver.InitialState` attribute), 236
`iter_block_iter()` (`westpa.cli.tools.w_crawl.IterRangeSelection` method), 64
`iter_block_iter()` (`westpa.cli.tools.w_fluxanl.IterRangeSelection` method), 161
`iter_block_iter()` (`westpa.cli.tools.w_ntop.IterRangeSelection` method), 97
`iter_block_iter()` (`westpa.cli.tools.w_pdist.IterRangeSelection` method), 53
`iter_block_iter()` (`westpa.cli.tools.w_select.IterRangeSelection` method), 79
`iter_block_iter()` (`westpa.oldtools.iframe.iter_range.IterRangeMixin` method), 312
`iter_block_iter()` (`westpa.oldtools.iframe.IterRangeMixin` method), 301
`iter_block_iter()` (`westpa.tools.iter_range.IterRangeSelection` method), 301

method), 291

iter_block_iter() (westpa.tools.IterRangeSelection method), 280

iter_block_iter() (westpa.tools.kinetics_tool.IterRangeSelection method), 292

iter_entry() (westpa.core.h5io.IterBlockedDataset method), 221

iter_group_name() (westpa.core.data_manager.WESTPADataManager method), 195

iter_label_values() (westpa.core.h5io.WESTTrajectory method), 215

iter_labels (westpa.core.h5io.WESTTrajectory property), 215

iter_object_name() (westpa.cli.tools.w_assign.WESTPAH5File method), 33

iter_object_name() (westpa.core.h5io.WESTPAH5File method), 219

iter_range() (westpa.cli.tools.w_crawl.IterRangeSelection method), 64

iter_range() (westpa.cli.tools.w_fluxanl.IterRangeSelection method), 161

iter_range() (westpa.cli.tools.w_ntop.IterRangeSelection method), 98

iter_range() (westpa.cli.tools.w_pdist.IterRangeSelection method), 54

iter_range() (westpa.cli.tools.w_select.IterRangeSelection method), 80

iter_range() (westpa.oldtools.iframe.iter_range.IterRangeSelection method), 313

iter_range() (westpa.oldtools.iframe.IterRangeMixin method), 302

iter_range() (westpa.tools.iter_range.IterRangeSelection method), 291

iter_range() (westpa.tools.IterRangeSelection method), 280

iter_range() (westpa.tools.kinetics_tool.IterRangeSelection method), 293

iter_slice() (westpa.core.h5io.IterBlockedDataset method), 222

Iteration (class in westpa.analysis.core), 333

iteration (westpa.cli.tools.w_ipa.WIPI property), 49

iteration (westpa.cli.tools.w_ipa.WIPIScheme property), 47

iteration (westpa.tools.wipi.WIPIScheme property), 297

iteration (westpa.tools.WIPIScheme property), 282

iteration() (westpa.analysis.core.Run method), 333

iterations (westpa.analysis.core.Run property), 332

IterBlockedDataset (class in westpa.core.h5io), 221

IterRangeMixin (class in westpa.oldtools.iframe), 301

IterRangeMixin (class in westpa.oldtools.iframe.iter_range), 312

IterRangeSelection (class in westpa.cli.tools.w_crawl), 63

IterRangeSelection (class in westpa.cli.tools.w_fluxanl), 160

IterRangeSelection (class in westpa.cli.tools.w_ntop), 97

IterRangeSelection (class in westpa.cli.tools.w_pdist), 53

IterRangeSelection (class in westpa.cli.tools.w_select), 79

IterRangeSelection (class in westpa.tools), 279

IterRangeSelection (class in westpa.tools.iter_range), 290

IterRangeSelection (class in westpa.tools.kinetics_tool), 292

J

join() (westpa.core.h5io.Trajectory method), 201

join() (westpa.core.h5io.WESTTrajectory method), 216

join() (westpa.westext.weed.BinCluster.ClusterList method), 326

join() (westpa.work_managers.zeromq.core.ZMQCore method), 262

join() (westpa.work_managers.zeromq.node.ZMQCore method), 264

join() (westpa.work_managers.zeromq.work_manager.ZMQCore method), 267

join() (westpa.work_managers.zeromq.worker.ZMQCore method), 274

join() (westpa.work_managers.zeromq.ZMQCore method), 257

join_simple() (westpa.westext.weed.BinCluster.ClusterList method), 326

join_traj() (in module westpa.core.h5io), 209

K

keys() (westpa.cli.tools.w_ipa.WIPIDataset method), 46

keys() (westpa.tools.KineticsIteration method), 282

keys() (westpa.tools.wipi.KineticsIteration method), 297

keys() (westpa.tools.wipi.WIPIDataset method), 297

keys() (westpa.tools.WIPIDataset method), 282

kineticEnergy (westpa.core.h5io.Frames attribute), 215

KineticsAnalysisMixin (class in westpa.oldtools.iframe), 306

KineticsAnalysisMixin (class in westpa.oldtools.iframe.kinetics), 313

KineticsIteration (class in westpa.tools), 282

KineticsIteration (class in westpa.tools.wipi), 297

L

label_axes() (in module westpa.core.h5io), 217

label_values (westpa.core.h5io.WESTTrajectory property), 215

labeled_flux_to_rate()	(in module <code>westpa.core.kinetics</code>), 174	M1 (<code>westpa.core.kinetics.rate_averaging.StreamingStatsTuple</code> attribute), 178
labeled_flux_to_rate()	(in module <code>westpa.core.kinetics.matrates</code>), 176	M2 (<code>westpa.core.kinetics.rate_averaging.StreamingStats1D</code> attribute), 177
labels	(<code>westpa.core.binning.assign.RecursiveBinMapper</code> property), 167	M2 (<code>westpa.core.kinetics.rate_averaging.StreamingStats2D</code> attribute), 178
labels	(<code>westpa.core.binning.RecursiveBinMapper</code> property), 163	M2 (<code>westpa.core.kinetics.rate_averaging.StreamingStatsTuple</code> attribute), 178
largest_allowed_weight	(<code>westpa.core.binning.mab_driver.WEDriver</code> attribute), 168	MABBinMapper (class in <code>westpa.core.binning</code>), 163
largest_allowed_weight	(<code>westpa.core.we_driver.WEDriver</code> attribute), 237	MABBinMapper (class in <code>westpa.core.binning.mab</code>), 168
linregress()	(in module <code>westpa.core.progress</code>), 222	MABBinMapper (class in <code>westpa.core.binning.mab_manager</code>), 170
list_schemes	(<code>westpa.cli.tools.w_ipa.WIPI</code> property), 49	MABDriver (class in <code>westpa.core.binning</code>), 163
list_schemes	(<code>westpa.cli.tools.w_ipa.WIPIScheme</code> property), 47	MABDriver (class in <code>westpa.core.binning.mab_driver</code>), 170
list_schemes	(<code>westpa.tools.wipi.WIPIScheme</code> property), 297	MABSimManager (class in <code>westpa.core.binning</code>), 164
list_schemes	(<code>westpa.tools.WIPIScheme</code> property), 282	MABSimManager (class in <code>westpa.core.binning.mab_manager</code>), 174
load()	(<code>westpa.core.h5io.Trajectory</code> static method), 203	main() (<code>westpa.cli.tools.w_assign.WESTParallelTool</code> method), 29
load_and_validate_data()	(<code>westpa.cli.tools.ploterr.GenericIntervalSubcommand</code> method), 118	main() (<code>westpa.cli.tools.w_bins.WESTTool</code> method), 19
load_config_from_west()	(<code>westpa.cli.tools.w_assign.WAssign</code> method), 35	main() (<code>westpa.cli.tools.w_crawl.WESTParallelTool</code> method), 63
load_module()	(in module <code>westpa.core.extloader</code>), 198	main() (<code>westpa.cli.tools.w_direct.WESTParallelTool</code> method), 69
load_numpy_or_text()	(in module <code>westpa.oldtools.files</code>), 300	main() (<code>westpa.cli.tools.w_dumpsegs.WESTTool</code> method), 140
load_numpy_or_text()	(<code>westpa.oldtools.aframe.data_reader.ExtDataReaderMixin</code> method), 311	main() (<code>westpa.cli.tools.w_eddist.WESTParallelTool</code> method), 91
load_numpy_or_text()	(<code>westpa.oldtools.aframe.ExtDataReaderMixin</code> method), 303	main() (<code>westpa.cli.tools.w_fluxanl.WESTTool</code> method), 160
load_plugins()	(<code>westpa.core.binning.mab_manager.WESimManager</code> method), 170	main() (<code>westpa.cli.tools.w_ipa.WESTParallelTool</code> method), 146
load_plugins()	(<code>westpa.core.sim_manager.WESimManager</code> method), 228	main() (<code>westpa.cli.tools.w_kinavg.WESTParallelTool</code> method), 124
load_state_file()	(<code>westpa.cli.tools.w_assign.WAssign</code> method), 35	main() (<code>westpa.cli.tools.w_kinetics.WESTParallelTool</code> method), 130
load_states_from_function()	(<code>westpa.cli.tools.w_assign.WAssign</code> method), 35	main() (<code>westpa.cli.tools.w_multi_west.WESTMultiTool</code> method), 103
load_trajectory()	(in module <code>westpa.core.propagators.executable</code>), 184	main() (<code>westpa.cli.tools.w_multi_west.WESTTool</code> method), 102
load_west()	(in module <code>westpa.core.h5io</code>), 217	main() (<code>westpa.cli.tools.w_ntop.WESTTool</code> method), 96
M		main() (<code>westpa.cli.tools.w_pdist.WESTParallelTool</code> method), 52
M1 (<code>westpa.core.kinetics.rate_averaging.StreamingStats1D</code> attribute), 177		main() (<code>westpa.cli.tools.w_postanalysis_matrix.WESTParallelTool</code> method), 143
M1 (<code>westpa.core.kinetics.rate_averaging.StreamingStats2D</code> attribute), 178		main() (<code>westpa.cli.tools.w_postanalysis_reweight.WESTParallelTool</code> method), 146
		main() (<code>westpa.cli.tools.w_red.WESTParallelTool</code> method), 105
		main() (<code>westpa.cli.tools.w_reweight.WESTParallelTool</code> method), 148
		main() (<code>westpa.cli.tools.w_select.WESTParallelTool</code> method), 78

`main()` (*westpa.cli.tools.w_stateprobs.WESTParallelTool method*), 136

`main()` (*westpa.cli.tools.w_trace.WESTTool method*), 38

`main()` (*westpa.tools.core.WESTMultiTool method*), 287

`main()` (*westpa.tools.core.WESTParallelTool method*), 286

`main()` (*westpa.tools.core.WESTTool method*), 286

`main()` (*westpa.tools.WESTMultiTool method*), 278

`main()` (*westpa.tools.WESTParallelTool method*), 277

`main()` (*westpa.tools.WESTTool method*), 276

`make_image()` (*westpa.cli.tools.plothist.NonUniformImage method*), 111

`make_internal_endpoint()` (*westpa.work_managers.zeromq.core.ZMQCore class method*), 261

`make_internal_endpoint()` (*westpa.work_managers.zeromq.node.ZMQCore class method*), 263

`make_internal_endpoint()` (*westpa.work_managers.zeromq.work_manager.ZMQCore class method*), 266

`make_internal_endpoint()` (*westpa.work_managers.zeromq.worker.ZMQCore class method*), 273

`make_internal_endpoint()` (*westpa.work_managers.zeromq.ZMQCore class method*), 256

`make_ipc_endpoint()` (*westpa.work_managers.zeromq.core.ZMQCore class method*), 261

`make_ipc_endpoint()` (*westpa.work_managers.zeromq.node.ZMQCore class method*), 263

`make_ipc_endpoint()` (*westpa.work_managers.zeromq.work_manager.ZMQCore class method*), 266

`make_ipc_endpoint()` (*westpa.work_managers.zeromq.worker.ZMQCore class method*), 273

`make_ipc_endpoint()` (*westpa.work_managers.zeromq.ZMQCore class method*), 256

`make_molecules_whole()` (*westpa.core.h5io.Trajectory method*), 208

`make_parser()` (*westpa.cli.tools.w_bins.WESTTool method*), 19

`make_parser()` (*westpa.cli.tools.w_dumpsegs.WESTTool method*), 140

`make_parser()` (*westpa.cli.tools.w_fluxanl.WESTTool method*), 160

`make_parser()` (*westpa.cli.tools.w_multi_west.WESTTool method*), 102

`make_parser()` (*westpa.cli.tools.w_ntop.WESTTool method*), 96

`make_parser()` (*westpa.cli.tools.w_trace.WESTTool method*), 38

`make_parser()` (*westpa.tools.core.WESTTool method*), 286

`make_parser()` (*westpa.tools.WESTTool method*), 276

`make_parser_and_process()` (*westpa.cli.tools.w_assign.WESTParallelTool method*), 29

`make_parser_and_process()` (*westpa.cli.tools.w_bins.WESTTool method*), 19

`make_parser_and_process()` (*westpa.cli.tools.w_crawl.WESTParallelTool method*), 63

`make_parser_and_process()` (*westpa.cli.tools.w_direct.WESTParallelTool method*), 69

`make_parser_and_process()` (*westpa.cli.tools.w_dumpsegs.WESTTool method*), 140

`make_parser_and_process()` (*westpa.cli.tools.w_eddist.WESTParallelTool method*), 91

`make_parser_and_process()` (*westpa.cli.tools.w_fluxanl.WESTTool method*), 160

`make_parser_and_process()` (*westpa.cli.tools.w_ipa.WESTParallelTool method*), 45

`make_parser_and_process()` (*westpa.cli.tools.w_kinavg.WESTParallelTool method*), 124

`make_parser_and_process()` (*westpa.cli.tools.w_kinetics.WESTParallelTool method*), 130

`make_parser_and_process()` (*westpa.cli.tools.w_multi_west.WESTMultiTool method*), 102

`make_parser_and_process()` (*westpa.cli.tools.w_multi_west.WESTTool method*), 102

`make_parser_and_process()` (*westpa.cli.tools.w_ntop.WESTTool method*), 96

`make_parser_and_process()` (*westpa.cli.tools.w_pdist.WESTParallelTool method*), 52

`make_parser_and_process()` (*westpa.cli.tools.w_postanalysis_matrix.WESTParallelTool method*), 143

`make_parser_and_process()` (*westpa.cli.tools.w_postanalysis_reweight.WESTParallelTool method*), 146

`make_parser_and_process()`

<i>(westpa.cli.tools.w_red.WESTParallelTool method)</i> , 105	<i>make_work_manager()</i> <i>(westpa.work_managers.environment.WMEnvironment method)</i> , 245
<i>make_parser_and_process()</i> <i>(westpa.cli.tools.w_reweight.WESTParallelTool method)</i> , 148	<i>makepath()</i> (in module <i>westpa.core.data_manager</i>), 194
<i>make_parser_and_process()</i> <i>(westpa.cli.tools.w_select.WESTParallelTool method)</i> , 78	<i>makepath()</i> (<i>westpa.core.data_manager.ExecutablePropagator</i> static method), 193
<i>make_parser_and_process()</i> <i>(westpa.cli.tools.w_stateprobs.WESTParallelTool method)</i> , 136	<i>makepath()</i> (<i>westpa.core.propagators.executable.ExecutablePropagator</i> static method), 186
<i>make_parser_and_process()</i> <i>(westpa.cli.tools.w_trace.WESTTool method)</i> , 38	<i>Manager</i> (class in <i>westpa.work_managers.mpi</i>), 249
<i>make_parser_and_process()</i> <i>(westpa.tools.core.WESTMultiTool method)</i> , 287	<i>map_binless()</i> (in module <i>westpa.core.binning</i>), 163
<i>make_parser_and_process()</i> <i>(westpa.tools.core.WESTParallelTool method)</i> , 286	<i>map_mab()</i> (in module <i>westpa.core.binning</i>), 163
<i>make_parser_and_process()</i> <i>(westpa.tools.core.WESTTool method)</i> , 286	<i>map_mab()</i> (in module <i>westpa.core.binning.mab</i>), 168
<i>make_parser_and_process()</i> <i>(westpa.tools.WESTMultiTool method)</i> , 278	<i>mapper_from_dict()</i> (in module <i>westpa.tools</i>), 281
<i>make_parser_and_process()</i> <i>(westpa.tools.WESTParallelTool method)</i> , 276	<i>mapper_from_dict()</i> (in module <i>westpa.tools.binning</i>), 284
<i>make_parser_and_process()</i> <i>(westpa.tools.WESTTool method)</i> , 276	<i>mapper_from_expr()</i> (in module <i>westpa.tools.binning</i>), 284
<i>make_tcp_endpoint()</i> <i>(westpa.work_managers.zeromq.core.ZMQCore class method)</i> , 261	<i>mapper_from_expr()</i> (<i>westpa.oldtools.iframe.binning.BinningMixin</i> method), 308
<i>make_tcp_endpoint()</i> <i>(westpa.work_managers.zeromq.node.ZMQCore class method)</i> , 263	<i>mapper_from_expr()</i> (<i>westpa.oldtools.iframe.BinningMixin</i> method), 304
<i>make_tcp_endpoint()</i> <i>(westpa.work_managers.zeromq.work_manager.ZMQCore class method)</i> , 266	<i>mapper_from_function()</i> (in module <i>westpa.tools.binning</i>), 284
<i>make_tcp_endpoint()</i> <i>(westpa.work_managers.zeromq.worker.ZMQCore class method)</i> , 273	<i>mapper_from_hdf5()</i> (in module <i>westpa.tools.binning</i>), 284
<i>make_tcp_endpoint()</i> <i>(westpa.work_managers.zeromq.ZMQCore class method)</i> , 256	<i>mapper_from_system()</i> (in module <i>westpa.tools.binning</i>), 284
<i>make_work_manager()</i> (in module <i>westpa.cli.core.w_init</i>), 16	<i>mapper_from_yaml()</i> (in module <i>westpa.tools.binning</i>), 284
<i>make_work_manager()</i> (in module <i>westpa.cli.core.w_run</i>), 22	<i>MASTER_BEACON</i> (<i>westpa.work_managers.zeromq.core.Message</i> attribute), 260
<i>make_work_manager()</i> (in module <i>westpa.cli.core.w_states</i>), 84	<i>MASTER_BEACON</i> (<i>westpa.work_managers.zeromq.node.Message</i> attribute), 264
<i>make_work_manager()</i> (in module <i>westpa.work_managers</i>), 242	<i>MASTER_BEACON</i> (<i>westpa.work_managers.zeromq.work_manager.Message</i> attribute), 267
<i>make_work_manager()</i> (in module <i>westpa.work_managers.environment</i>), 245	<i>MASTER_BEACON</i> (<i>westpa.work_managers.zeromq.worker.Message</i> attribute), 274
	<i>max</i> (<i>westpa.core.sim_manager.timedelta</i> attribute), 226
	<i>max_acc</i> (<i>westpa.oldtools.iframe.TransitionEventAccumulator</i> attribute), 306
	<i>max_acc</i> (<i>westpa.oldtools.iframe.transitions.TransitionEventAccumulator</i> attribute), 316
	<i>max_iter_segs_in_range()</i> <i>(westpa.cli.core.w_succ.WESTDataReaderMixin method)</i> , 59
	<i>max_iter_segs_in_range()</i> <i>(westpa.oldtools.iframe.data_reader.WESTDataReaderMixin method)</i> , 310
	<i>max_iter_segs_in_range()</i> <i>(westpa.oldtools.iframe.WESTDataReaderMixin method)</i> , 303
	<i>maxlen</i> (<i>westpa.work_managers.mpi.deque</i> attribute), 246
	<i>maxlen</i> (<i>westpa.work_managers.zeromq.work_manager.deque</i>

attribute), 271
`mcbs_ci()` (in module `westpa.mclib`), 298
`mcbs_ci_correl()` (in module `westpa.cli.tools.w_direct`), 70
`mcbs_ci_correl()` (in module `westpa.cli.tools.w_reweight`), 149
`mcbs_ci_correl()` (in module `westpa.mclib`), 299
`mcbs_correltime()` (in module `westpa.mclib`), 298
`MCBSMixin` (class in `westpa.oldtools.iframe`), 305
`MCBSMixin` (class in `westpa.oldtools.iframe.mcbs`), 313
`mean` (`westpa.core.kinetics.rate_averaging.StreamingStats1D` attribute), 177
`mean` (`westpa.core.kinetics.rate_averaging.StreamingStats2D` attribute), 178
`mean()` (`westpa.oldtools.stats.accumulator.RunningStatsAccumulator` method), 318
`mean()` (`westpa.oldtools.stats.edfs.EDF` method), 318
`mean()` (`westpa.oldtools.stats.RunningStatsAccumulator` method), 317
`median()` (`westpa.oldtools.stats.edfs.EDF` method), 318
`Message` (class in `westpa.work_managers.zeromq.core`), 260
`Message` (class in `westpa.work_managers.zeromq.node`), 264
`Message` (class in `westpa.work_managers.zeromq.work_manager`), 267
`Message` (class in `westpa.work_managers.zeromq.worker`), 274
`message_validation()` (`westpa.work_managers.zeromq.core.ZMQCore` method), 261
`message_validation()` (`westpa.work_managers.zeromq.node.ZMQCore` method), 263
`message_validation()` (`westpa.work_managers.zeromq.work_manager.ZMQCore` method), 266
`message_validation()` (`westpa.work_managers.zeromq.worker.ZMQCore` method), 273
`message_validation()` (`westpa.work_managers.zeromq.ZMQCore` method), 257
`microseconds` (`westpa.core.sim_manager.timedelta` attribute), 226
`min` (`westpa.core.sim_manager.timedelta` attribute), 226
module
 `westpa.analysis.core`, 332
 `westpa.analysis.statistics`, 340
 `westpa.analysis.trajectories`, 338
 `westpa.cli.core.w_fork`, 24
 `westpa.cli.core.w_init`, 15
 `westpa.cli.core.w_run`, 22
 `westpa.cli.core.w_states`, 84
 `westpa.cli.core.w_succ`, 57
 `westpa.cli.core.w_truncate`, 23
 `westpa.cli.tools.ploterr`, 116
 `westpa.cli.tools.plothist`, 111
 `westpa.cli.tools.w_assign`, 28
 `westpa.cli.tools.w_bins`, 19
 `westpa.cli.tools.w_crawl`, 63
 `westpa.cli.tools.w_direct`, 68
 `westpa.cli.tools.w_dumpsegs`, 140
 `westpa.cli.tools.w_eddist`, 91
 `westpa.cli.tools.w_fluxanl`, 157
 `westpa.cli.tools.w_ipa`, 45
 `westpa.cli.tools.w_kinavg`, 123
 `westpa.cli.tools.w_kinetics`, 129
 `westpa.cli.tools.w_multi_west`, 102
 `westpa.cli.tools.w_ntop`, 96
 `westpa.cli.tools.w_pdlist`, 52
 `westpa.cli.tools.w_postanalysis_matrix`, 142
 `westpa.cli.tools.w_postanalysis_reweight`, 145
 `westpa.cli.tools.w_red`, 105
 `westpa.cli.tools.w_reweight`, 147
 `westpa.cli.tools.w_select`, 78
 `westpa.cli.tools.w_stateprobs`, 136
 `westpa.cli.tools.w_trace`, 37
 `westpa.core`, 188
 `westpa.core.binning`, 162
 `westpa.core.binning.assign`, 165, 349
 `westpa.core.binning.bins`, 167, 349
 `westpa.core.binning.mab`, 168
 `westpa.core.binning.mab_driver`, 168
 `westpa.core.binning.mab_manager`, 170
 `westpa.core.data_manager`, 188
 `westpa.core.extloader`, 198
 `westpa.core.h5io`, 198
 `westpa.core.kinetics`, 174
 `westpa.core.kinetics.events`, 175
 `westpa.core.kinetics.matrates`, 175
 `westpa.core.kinetics.rate_averaging`, 177
 `westpa.core.progress`, 222
 `westpa.core.propagators`, 179
 `westpa.core.propagators.executable`, 180
 `westpa.core.reweight`, 187
 `westpa.core.reweight.matrix`, 187
 `westpa.core.segment`, 224
 `westpa.core.sim_manager`, 225
 `westpa.core.states`, 230
 `westpa.core.systems`, 233
 `westpa.core.textio`, 234
 `westpa.core.we_driver`, 234
 `westpa.core.wm_ops`, 238
 `westpa.core.yamlcfg`, 239, 349
 `westpa.fasthist`, 298

- westpa.mclib, 298
 - westpa.oldtools, 300
 - westpa.oldtools.iframe, 301
 - westpa.oldtools.iframe.atool, 307
 - westpa.oldtools.iframe.base_mixin, 307
 - westpa.oldtools.iframe.binning, 308
 - westpa.oldtools.iframe.data_reader, 308
 - westpa.oldtools.iframe.iter_range, 312
 - westpa.oldtools.iframe.kinetics, 313
 - westpa.oldtools.iframe.mcbs, 313
 - westpa.oldtools.iframe.output, 314
 - westpa.oldtools.iframe.plotting, 315
 - westpa.oldtools.iframe.trajwalker, 315
 - westpa.oldtools.iframe.transitions, 315
 - westpa.oldtools.cmds, 317
 - westpa.oldtools.files, 300
 - westpa.oldtools.miscfn, 301
 - westpa.oldtools.stats, 317
 - westpa.oldtools.stats.accumulator, 317
 - westpa.oldtools.stats.edfs, 318
 - westpa.oldtools.stats.mcbs, 319
 - westpa.tools, 276
 - westpa.tools.binning, 283
 - westpa.tools.core, 285
 - westpa.tools.data_reader, 288
 - westpa.tools.dtypes, 290
 - westpa.tools.iter_range, 290
 - westpa.tools.kinetics_tool, 292
 - westpa.tools.plot, 294
 - westpa.tools.progress, 294
 - westpa.tools.selected_segs, 295
 - westpa.tools.wipi, 297
 - westpa.trajtree, 299
 - westpa.trajtree.trajtree, 300
 - westpa.westext, 329
 - westpa.westext.adaptvoronoi, 320
 - westpa.westext.adaptvoronoi.adaptVor_driver, 319
 - westpa.westext.weed, 327
 - westpa.westext.weed.BinCluster, 326
 - westpa.westext.weed.ProbAdjustEquil, 326
 - westpa.westext.weed.UncertMath, 326
 - westpa.westext.weed.weed_driver, 327
 - westpa.westext.wess, 329
 - westpa.westext.wess.ProbAdjust, 328
 - westpa.westext.wess.wess_driver, 328
 - westpa.work_managers, 241
 - westpa.work_managers.core, 242
 - westpa.work_managers.environment, 245
 - westpa.work_managers.mpi, 246
 - westpa.work_managers.processes, 249
 - westpa.work_managers.serial, 252
 - westpa.work_managers.threads, 254
 - westpa.work_managers.zeromq, 256
 - westpa.work_managers.zeromq.core, 259
 - westpa.work_managers.zeromq.node, 263
 - westpa.work_managers.zeromq.work_manager, 265
 - westpa.work_managers.zeromq.worker, 272
 - moment() (*westpa.oldtools.stats.edfs.EDF method*), 318
 - mouseover (*westpa.cli.tools.plothist.NonUniformImage attribute*), 111
 - MPIWorkManager (*class in westpa.work_managers.mpi*), 248
 - MultiDSSpec (*class in westpa.core.h5io*), 221
 - MultiDSSpec (*class in westpa.tools.data_reader*), 288
- ## N
- n (*westpa.core.kinetics.rate_averaging.StreamingStats1D attribute*), 178
 - n (*westpa.core.kinetics.rate_averaging.StreamingStats2D attribute*), 178
 - n (*westpa.core.kinetics.rate_averaging.StreamingStatsTuple attribute*), 178
 - n_atoms (*westpa.core.h5io.Trajectory attribute*), 199
 - n_atoms (*westpa.core.h5io.Trajectory property*), 199
 - n_chains (*westpa.core.h5io.Trajectory property*), 200
 - n_frames (*westpa.core.h5io.Trajectory attribute*), 198
 - n_frames (*westpa.core.h5io.Trajectory property*), 199
 - n_istates_needed (*westpa.core.binning.mab_driver.WEDriver property*), 169
 - n_istates_needed (*westpa.core.we_driver.WEDriver property*), 237
 - n_iter (*westpa.trajtree.trajtree.trajnode attribute*), 300
 - n_iter_blocks() (*westpa.cli.tools.w_crawl.IterRangeSelection method*), 64
 - n_iter_blocks() (*westpa.cli.tools.w_fluxanl.IterRangeSelection method*), 161
 - n_iter_blocks() (*westpa.cli.tools.w_ntop.IterRangeSelection method*), 97
 - n_iter_blocks() (*westpa.cli.tools.w_pdist.IterRangeSelection method*), 53
 - n_iter_blocks() (*westpa.cli.tools.w_select.IterRangeSelection method*), 79
 - n_iter_blocks() (*westpa.oldtools.iframe.iter_range.IterRangeMixin method*), 312
 - n_iter_blocks() (*westpa.oldtools.iframe.IterRangeMixin method*), 301
 - n_iter_blocks() (*westpa.tools.iter_range.IterRangeSelection method*), 291
 - n_iter_blocks() (*westpa.tools.IterRangeSelection method*), 280
 - n_iter_blocks() (*westpa.tools.kinetics_tool.IterRangeSelection method*), 292
 - n_iter_dtype (*in module westpa.cli.core.w_fork*), 26
 - n_iter_dtype (*in module westpa.cli.tools.w_fluxanl*), 159

`n_iter_dtype` (in module `westpa.cli.tools.w_multi_west`), 102
`n_iter_dtype` (in module `westpa.cli.tools.w_ntop`), 98
`n_iter_dtype` (in module `westpa.cli.tools.w_select`), 80
`n_iter_dtype` (in module `westpa.cli.tools.w_trace`), 40
`n_iter_dtype` (in module `westpa.core.data_manager`), 194
`n_iter_dtype` (in module `westpa.tools.dtypes`), 290
`n_iters` (`westpa.cli.tools.w_red.RateCalculator` property), 106
`n_recycled_segs` (`westpa.core.binning.mab_driver.WEDriver` property), 169
`n_recycled_segs` (`westpa.core.we_driver.WEDriver` property), 237
`n_residues` (`westpa.core.h5io.Trajectory` attribute), 199
`n_residues` (`westpa.core.h5io.Trajectory` property), 200
`n_workers` (`westpa.work_managers.zeromq.work_manager.ZMQWorker` property), 272
`n_workers` (`westpa.work_managers.zeromq.ZMQWorkManager` property), 258
`NAK` (`westpa.work_managers.zeromq.core.Message` attribute), 260
`NAK` (`westpa.work_managers.zeromq.node.Message` attribute), 264
`NAK` (`westpa.work_managers.zeromq.work_manager.Message` attribute), 267
`NAK` (`westpa.work_managers.zeromq.worker.Message` attribute), 274
`namedtuple()` (in module `westpa.core.kinetics.rate_averaging`), 177
`nested_to_flat_matrix()` (in module `westpa.core.kinetics`), 174
`nested_to_flat_matrix()` (in module `westpa.core.kinetics.matrates`), 176
`nested_to_flat_vector()` (in module `westpa.core.kinetics`), 174
`nested_to_flat_vector()` (in module `westpa.core.kinetics.matrates`), 176
`new_iteration()` (`westpa.core.binning.mab_driver.WEDriver` method), 169
`new_iteration()` (`westpa.core.we_driver.WEDriver` method), 237
`new_operation()` (`westpa.core.progress.ProgressIndicator` method), 224
`new_operation()` (`westpa.tools.progress.ProgressIndicator` method), 295
`new_pcoord_array()` (`westpa.core.systems.WESTSystem` method), 234
`new_pcoord_array()` (`westpa.core.yamlcfg.YAMLSystem` method), 240
`new_region_set()` (`westpa.core.systems.WESTSystem` method), 234
`new_region_set()` (`westpa.core.yamlcfg.YAMLSystem` method), 240
`NewWeightEntry` (class in `westpa.cli.tools.w_fluxanl`), 159
`NewWeightEntry` (class in `westpa.core.data_manager`), 192
`NewWeightEntry` (class in `westpa.core.we_driver`), 236
`next` (`westpa.analysis.core.Iteration` property), 333
`next_expiration()` (`westpa.work_managers.zeromq.core.PassiveMultiTimer` method), 261
`next_expiration()` (`westpa.work_managers.zeromq.node.PassiveMultiTimer` method), 265
`next_expiration()` (`westpa.work_managers.zeromq.work_manager.PassiveMultiTimer` method), 268
`next_expiration()` (`westpa.work_managers.zeromq.worker.PassiveMultiTimer` method), 275
`next_expiration_in()` (`westpa.work_managers.zeromq.core.PassiveMultiTimer` method), 261
`next_expiration_in()` (`westpa.work_managers.zeromq.node.PassiveMultiTimer` method), 265
`next_expiration_in()` (`westpa.work_managers.zeromq.work_manager.PassiveMultiTimer` method), 268
`next_expiration_in()` (`westpa.work_managers.zeromq.worker.PassiveMultiTimer` method), 275
`next_iter_assignments` (`westpa.core.binning.mab_driver.WEDriver` property), 169
`next_iter_assignments` (`westpa.core.we_driver.WEDriver` property), 237
`next_iter_segments` (`westpa.core.binning.mab_driver.WEDriver` property), 168
`next_iter_segments` (`westpa.core.we_driver.WEDriver` property), 237
`NonUniformImage` (class in `westpa.cli.tools.plothist`), 111
`nop()` (in module `westpa.core.progress`), 224
`NopMapper` (class in `westpa.core.binning`), 162
`NopMapper` (class in `westpa.core.binning.assign`), 166
`NopMapper` (class in `westpa.core.systems`), 233
`NopMapper` (class in `westpa.core.yamlcfg`), 239
`normalize_dataset_options()` (in module `westpa.core.data_manager`), 197
`normhistnd()` (in module `westpa.cli.tools.plothist`), 113
`normhistnd()` (in module `westpa.cli.tools.w_eddist`), 92
`normhistnd()` (in module `westpa.cli.tools.w_pdist`), 54
`normhistnd()` (in module `westpa.fasthist`), 298
`np_data_loader()` (in module `westpa.core.propagators.executable`), 185
`np_to_h5dataset()` (`westpa.oldtools.iframe.data_reader.ExtDataReader` method), 311
`np_to_h5dataset()` (`westpa.oldtools.iframe.ExtDataReaderMixin`

method), 303
 num_bins (*westpa.analysis.core.Iteration* property), 334
 num_iterations (*westpa.analysis.core.Run* property), 332
 num_segments (*westpa.analysis.core.Iteration* property), 334
 num_segments (*westpa.analysis.core.Run* property), 332
 num_snapshots (*westpa.analysis.core.Walker* property), 336
 num_walkers (*westpa.analysis.core.Iteration* property), 334
 num_walkers (*westpa.analysis.core.Run* property), 332
 NumericTextOutputFormatter (class in *westpa.core.textio*), 234
 nw_source_dtype (in module *westpa.core.data_manager*), 194
 NW_SOURCE_RECYCLED (*westpa.cli.tools.w_fluxanl.NewWeightedEnsemble* attribute), 159
 NW_SOURCE_RECYCLED (*westpa.core.data_manager.NewWeightedEnsemble* attribute), 192
 NW_SOURCE_RECYCLED (*westpa.core.we_driver.NewWeightedEnsemble* attribute), 236

O

open() (*westpa.analysis.core.Run* class method), 332
 open() (*westpa.cli.tools.w_assign.WESTDataReader* method), 29
 open() (*westpa.cli.tools.w_bins.WESTDataReader* method), 19
 open() (*westpa.cli.tools.w_crawl.WESTDataReader* method), 63
 open() (*westpa.cli.tools.w_dumpsegs.WESTDataReader* method), 141
 open() (*westpa.cli.tools.w_fluxanl.WESTDataReader* method), 160
 open() (*westpa.cli.tools.w_ipa.WESTDataReader* method), 46
 open() (*westpa.cli.tools.w_ntop.WESTDataReader* method), 97
 open() (*westpa.cli.tools.w_pdist.WESTDataReader* method), 52
 open() (*westpa.cli.tools.w_select.WESTDataReader* method), 79
 open() (*westpa.cli.tools.w_trace.WESTDataReader* method), 38
 open() (*westpa.tools.data_reader.WESTDataReader* method), 289
 open() (*westpa.tools.kinetics_tool.WESTDataReader* method), 292
 open() (*westpa.tools.WESTDataReader* method), 279
 open_analysis_backing() (*westpa.cli.core.w_succ.WESTAnalysisTool* method), 58
 open_analysis_backing() (*westpa.oldtools.iframe.atool.WESTAnalysisTool* method), 307
 open_analysis_backing() (*westpa.oldtools.iframe.WESTAnalysisTool* method), 301
 open_assignments() (*westpa.cli.tools.w_direct.AverageCommands* method), 70
 open_assignments() (*westpa.cli.tools.w_reweight.AverageCommands* method), 148
 open_assignments() (*westpa.tools.kinetics_tool.AverageCommands* method), 294
 open_backing() (*westpa.core.data_manager.WESTDataManager* method), 195
 open_files() (*westpa.cli.tools.w_direct.AverageCommands* method), 70
 open_files() (*westpa.cli.tools.w_direct.DKinetics* method), 71
 open_files() (*westpa.cli.tools.w_kinetics.DKinetics* method), 131
 open_files() (*westpa.cli.tools.w_multi_west.WMultiWest* method), 103
 open_files() (*westpa.cli.tools.w_reweight.AverageCommands* method), 148
 open_files() (*westpa.tools.kinetics_tool.AverageCommands* method), 294
 openmm_boxes() (*westpa.core.h5io.Trajectory* method), 203
 openmm_positions() (*westpa.core.h5io.Trajectory* method), 203
 operation (*westpa.core.progress.ProgressIndicator* property), 224
 operation (*westpa.tools.progress.ProgressIndicator* property), 294
 output_format_version (*westpa.cli.tools.w_fluxanl.WFluxanlTool* attribute), 162
 output_map() (in module *westpa.core.binning.assign*), 165
 output_tdat_chunksize (*westpa.oldtools.iframe.TransitionEventAccumulator* attribute), 306
 output_tdat_chunksize (*westpa.oldtools.iframe.transitions.TransitionEventAccumulator* attribute), 316

P

PAAverage (class in *westpa.cli.tools.w_postanalysis_reweight*), 147
 PAMatrix (class in *westpa.cli.tools.w_postanalysis_matrix*), 144
 pare_basis_initial_states() (in module *westpa.core.binning.mab_manager*), 172

`pare_basis_initial_states()` (in module `westpa.oldtools.iframe.data_reader`), 310
`westpa.core.states`, 233
`parent` (`westpa.analysis.core.Walker` property), 336
`parent_id_dsspec` (`westpa.cli.tools.w_assign.WESTDataReader` property), 29
`parent_id_dsspec` (`westpa.cli.tools.w_bins.WESTDataReader` property), 19
`parent_id_dsspec` (`westpa.cli.tools.w_crawl.WESTDataReader` property), 63
`parent_id_dsspec` (`westpa.cli.tools.w_dumpsegs.WESTDataReader` property), 141
`parent_id_dsspec` (`westpa.cli.tools.w_fluxanl.WESTDataReader` property), 160
`parent_id_dsspec` (`westpa.cli.tools.w_ipa.WESTDataReader` property), 46
`parent_id_dsspec` (`westpa.cli.tools.w_ntop.WESTDataReader` property), 97
`parent_id_dsspec` (`westpa.cli.tools.w_pdist.WESTDataReader` property), 52
`parent_id_dsspec` (`westpa.cli.tools.w_select.WESTDataReader` property), 79
`parent_id_dsspec` (`westpa.cli.tools.w_trace.WESTDataReader` property), 38
`parent_id_dsspec` (`westpa.tools.data_reader.WESTDataReader` property), 289
`parent_id_dsspec` (`westpa.tools.kinetics_tool.WESTDataReader` property), 292
`parent_id_dsspec` (`westpa.tools.WESTDataReader` property), 279
`parent_ids` (`westpa.core.h5io.WESTTrajectory` property), 216
`parse_bin_range()` (`westpa.oldtools.iframe.kinetics.KineticsAnalysisMixin` method), 313
`parse_bin_range()` (`westpa.oldtools.iframe.KineticsAnalysisMixin` method), 306
`parse_binspec()` (`westpa.cli.tools.w_eddist.WEDist` static method), 94
`parse_binspec()` (`westpa.cli.tools.w_pdist.WPDist` static method), 56
`parse_cmdline_states()` (`westpa.cli.tools.w_assign.WAssign` method), 35
`parse_dataset_string()` (`westpa.cli.tools.w_trace.WTraceTool` method), 42
`parse_dimspec()` (`westpa.cli.tools.plothist.PlotHistBase` method), 113
`parse_from_yaml()` (`westpa.cli.tools.w_multi_west.WESTMultiTool` method), 102
`parse_from_yaml()` (`westpa.tools.core.WESTMultiTool` method), 287
`parse_from_yaml()` (`westpa.tools.WESTMultiTool` method), 278
`parse_int_list()` (in module `westpa.oldtools.miscfn`), 301
`parse_pcoord_value()` (in module `westpa.cli.tools.w_assign`), 33
`parse_range()` (`westpa.cli.tools.ploterr.CommonPloterr` method), 117
`parse_range()` (`westpa.cli.tools.plothist.PlotHistBase` method), 113
`parse_segssel_file()` (`westpa.tools.SegSelector` method), 280
`parse_segssel_file()` (`westpa.tools.selected_segs.SegSelector` method), 296
`PassiveMultiTimer` (class in `westpa.work_managers.zeromq.core`), 260
`PassiveMultiTimer` (class in `westpa.work_managers.zeromq.node`), 265
`PassiveMultiTimer` (class in `westpa.work_managers.zeromq.work_manager`), 268
`PassiveMultiTimer` (class in `westpa.work_managers.zeromq.worker`), 274
`PassiveTimer` (class in `westpa.work_managers.zeromq.core`), 260
`past` (`westpa.cli.tools.w_ipa.WIPI` property), 49
`past` (`westpa.cli.tools.w_ipa.WIPIScheme` property), 47
`past` (`westpa.tools.wipi.WIPIScheme` property), 298
`past` (`westpa.tools.WIPIScheme` property), 283
`pcoord_formats` (`westpa.cli.tools.w_trace.WTraceTool` attribute), 42
`pcoord_loader()` (in module `westpa.core.propagators.executable`), 184
`pcoords` (`westpa.analysis.core.Iteration` property), 333
`pcoords` (`westpa.analysis.core.Walker` property), 336
`pcoords` (`westpa.core.h5io.WESTTrajectory` property), 216
`pickle_and_hash()` (`westpa.core.binning.assign.BinMapper` method), 166
`pickle_data_loader()` (in module `westpa.core.propagators.executable`), 185
`PickleError`, 226, 283
`PiecewiseBinMapper` (class in `westpa.core.binning`), 162
`PiecewiseBinMapper` (class in `westpa.core.binning.assign`), 166
`Plotter` (`westpa.cli.tools.ploterr.Plotter` method), 117
`plot()` (`westpa.cli.tools.w_ipa.Plotter` method), 46
`plot()` (`westpa.tools.plot.Plotter` method), 294
`plot()` (`westpa.tools.Plotter` method), 282
`plot()` (`westpa.tools.wipi.Plotter` method), 297
`plot_color()` (`westpa.cli.tools.ploterr.DirectStateprobs` method), 119

`plot_flux()` (*westpa.cli.tools.ploterr.DirectKinetics* method), 118
`plot_pop()` (*westpa.cli.tools.ploterr.DirectStateprobs* method), 119
`plot_rate()` (*westpa.cli.tools.ploterr.DirectKinetics* method), 118
`PloterrsTool` (class in *westpa.cli.tools.ploterr*), 120
`PlotHistBase` (class in *westpa.cli.tools.plothist*), 113
`PlotHistTool` (class in *westpa.cli.tools.plothist*), 115
`PlotSupports2D` (class in *westpa.cli.tools.plothist*), 113
`Plotter` (class in *westpa.cli.tools.ploterr*), 117
`Plotter` (class in *westpa.cli.tools.w_ipa*), 46
`Plotter` (class in *westpa.tools*), 282
`Plotter` (class in *westpa.tools.plot*), 294
`Plotter` (class in *westpa.tools.wipi*), 297
`PlottingMixin` (class in *westpa.oldtools.iframe*), 307
`PlottingMixin` (class in *westpa.oldtools.plotting*), 315
`pop()` (*westpa.work_managers.mpi.deque* method), 246
`pop()` (*westpa.work_managers.zeromq.work_manager.deque* method), 271
`pop_assign()` (in module *westpa.core.kinetics.rate_averaging*), 177
`pop_output_filename` (*westpa.cli.tools.ploterr.DirectStateprobs* attribute), 119
`pop_output_filename` (*westpa.cli.tools.ploterr.ReweightStateprobs* attribute), 119
`popleft()` (*westpa.work_managers.mpi.deque* method), 246
`popleft()` (*westpa.work_managers.zeromq.work_manager.deque* method), 271
`populate_initial()` (*westpa.core.binning.mab_driver.WEDriver* method), 169
`populate_initial()` (*westpa.core.we_driver.WEDriver* method), 238
`populations` (*westpa.cli.tools.w_red.RateCalculator* property), 106
`post_iter()` (in module *westpa.core.wm_ops*), 238
`post_propagation()` (*westpa.core.binning.mab_manager.WESimManager* method), 171
`post_propagation()` (*westpa.core.sim_manager.WESimManager* method), 229
`post_we()` (*westpa.core.binning.mab_manager.WESimManager* method), 171
`post_we()` (*westpa.core.sim_manager.WESimManager* method), 229
`potentialEnergy` (*westpa.core.h5io.Frames* attribute), 215
`pre_propagation()` (*westpa.core.binning.mab_manager.WESimManager* method), 171
`pre_propagation()` (*westpa.core.sim_manager.WESimManager* method), 229
`pre_we()` (*westpa.core.binning.mab_manager.WESimManager* method), 171
`pre_we()` (*westpa.core.sim_manager.WESimManager* method), 229
`preload_config_files` (*westpa.core.yamlcfg.YAMLConfig* attribute), 239
`prep_iter()` (in module *westpa.core.wm_ops*), 238
`prepare_backing()` (*westpa.core.data_manager.WESTDataManager* method), 195
`prepare_file_system()` (*westpa.core.data_manager.ExecutablePropagator* method), 193
`prepare_file_system()` (*westpa.core.propagators.executable.ExecutablePropagator* method), 186
`prepare_iteration()` (*westpa.core.binning.BinlessSimManager* method), 164
`prepare_iteration()` (*westpa.core.binning.mab_manager.MABSimManager* method), 174
`prepare_iteration()` (*westpa.core.binning.mab_manager.WESimManager* method), 171
`prepare_iteration()` (*westpa.core.binning.MABSimManager* method), 164
`prepare_iteration()` (*westpa.core.data_manager.ExecutablePropagator* method), 194
`prepare_iteration()` (*westpa.core.data_manager.WESTDataManager* method), 196
`prepare_iteration()` (*westpa.core.propagators.executable.ExecutablePropagator* method), 187
`prepare_iteration()` (*westpa.core.propagators.executable.WESTPropagator* method), 181
`prepare_iteration()` (*westpa.core.propagators.WESTPropagator* method), 179
`prepare_iteration()` (*westpa.core.sim_manager.WESimManager* method), 229
`prepare_new_iteration()` (*westpa.core.binning.mab_manager.WESimManager* method), 171
`prepare_new_iteration()` (*westpa.core.sim_manager.WESimManager* method), 229
`prepare_new_iteration()` (*westpa.westtext.adaptvoronoi.AdaptiveVoronoiDriver* method), 171

method), 321

prepare_new_iteration()
(westpa.westext.adaptvoronoi.adaptVor_driver.AdaptiveVoronoiDriver method), 320

prepare_new_iteration()
(westpa.westext.weed.weed_driver.WEEDDriver method), 327

prepare_new_iteration()
(westpa.westext.weed.WEEDDriver method), 328

prepare_new_iteration()
(westpa.westext.wess.wess_driver.WESSDriver method), 329

prepare_new_iteration()
(westpa.westext.wess.WESSDriver method), 329

prepare_run() (westpa.core.binning.mab_manager.WESimManager method), 171

prepare_run() (westpa.core.data_manager.WESTDataManager method), 197

prepare_run() (westpa.core.sim_manager.WESimManager method), 229

prepare_run() (westpa.core.systems.WESTSystem method), 233

prepare_run() (westpa.core.yamlcfg.YAMLSystem method), 240

prepare_segment_restarts()
(westpa.core.data_manager.WESTDataManager method), 196

prev (westpa.analysis.core.Iteration property), 333

print_averages() (westpa.cli.tools.w_direct.AverageCommands method), 70

print_averages() (westpa.cli.tools.w_reweight.AverageCommands method), 148

print_averages() (westpa.tools.kinetics_tool.AverageCommands method), 294

prob_adjust() (in module westpa.westext.wess), 329

prob_adjust() (in module westpa.westext.wess.ProbAdjust), 328

prob_adjust() (in module westpa.westext.wess.wess_driver), 328

probAdjustEquil() (in module westpa.westext.weed), 327

probAdjustEquil() (in module westpa.westext.weed.ProbAdjustEquil), 326

probAdjustEquil() (in module westpa.westext.weed.weed_driver), 327

process_all_args() (westpa.tools.binning.WESTToolComponent method), 284

process_all_args() (westpa.tools.core.WESTToolComponent method), 286

process_all_args() (westpa.tools.data_reader.WESTToolComponent method), 288

process_all_args() (westpa.tools.iter_range.WESTToolComponent method), 290

process_all_args() (westpa.tools.progress.WESTToolComponent method), 295

process_all_args() (westpa.tools.selected_segs.WESTToolComponent method), 296

process_all_args() (westpa.tools.WESTToolComponent method), 277

process_args() (westpa.cli.core.w_succ.WESTAnalysisTool method), 58

process_args() (westpa.cli.core.w_succ.WESTDataReaderMixin method), 58

process_args() (westpa.cli.tools.ploterr.CommonPloterrs method), 117

process_args() (westpa.cli.tools.ploterr.DirectKinetics method), 118

process_args() (westpa.cli.tools.ploterr.DirectStateprobs method), 119

process_args() (westpa.cli.tools.ploterr.GenericIntervalSubcommand method), 118

process_args() (westpa.cli.tools.ploterr.ProgressIndicatorComponent method), 117

process_args() (westpa.cli.tools.ploterr.WESTMasterCommand method), 116

process_args() (westpa.cli.tools.plothist.AveragePlotHist method), 114

process_args() (westpa.cli.tools.plothist.EvolutionPlotHist method), 115

process_args() (westpa.cli.tools.plothist.InstantPlotHist method), 114

process_args() (westpa.cli.tools.plothist.PlotHistBase method), 113

process_args() (westpa.cli.tools.plothist.PlotSupports2D method), 113

process_args() (westpa.cli.tools.plothist.WESTMasterCommand method), 113

process_args() (westpa.cli.tools.w_assign.BinMappingComponent method), 30

process_args() (westpa.cli.tools.w_assign.ProgressIndicatorComponent method), 31

process_args() (westpa.cli.tools.w_assign.WAssign method), 35

process_args() (westpa.cli.tools.w_assign.WESTDataReader method), 29

process_args() (westpa.cli.tools.w_assign.WESTDSSynthesizer method), 30

process_args() (westpa.cli.tools.w_assign.WESTParallelTool method), 29

process_args() (westpa.cli.tools.w_bins.BinMappingComponent method), 20

process_args() (westpa.cli.tools.w_bins.WBinTool method), 21

process_args() (westpa.cli.tools.w_bins.WESTDataReader method), 19

process_args() (westpa.cli.tools.w_bins.WESTToolComponent method), 20

method), 19

`process_args()` (`westpa.cli.tools.w_crawl.IterRangeSelection` method), 64

`process_args()` (`westpa.cli.tools.w_crawl.ProgressIndicatorComponent` method), 64

`process_args()` (`westpa.cli.tools.w_crawl.WCrawl` method), 65

`process_args()` (`westpa.cli.tools.w_crawl.WESTDataReader` method), 63

`process_args()` (`westpa.cli.tools.w_crawl.WESTParallelTool` method), 63

`process_args()` (`westpa.cli.tools.w_direct.AverageCommands` method), 70

`process_args()` (`westpa.cli.tools.w_direct.WESTKineticsBase` method), 69

`process_args()` (`westpa.cli.tools.w_direct.WESTMasterCommand` method), 68

`process_args()` (`westpa.cli.tools.w_direct.WESTParallelTool` method), 69

`process_args()` (`westpa.cli.tools.w_dumpsegs.WDumpSegs` method), 142

`process_args()` (`westpa.cli.tools.w_dumpsegs.WESTDataReader` method), 141

`process_args()` (`westpa.cli.tools.w_dumpsegs.WESTTool` method), 140

`process_args()` (`westpa.cli.tools.w_eddist.ProgressIndicatorComponent` method), 92

`process_args()` (`westpa.cli.tools.w_eddist.WEDDist` method), 94

`process_args()` (`westpa.cli.tools.w_eddist.WESTParallelTool` method), 91

`process_args()` (`westpa.cli.tools.w_fluxanl.IterRangeSelection` method), 161

`process_args()` (`westpa.cli.tools.w_fluxanl.WESTDataReader` method), 160

`process_args()` (`westpa.cli.tools.w_fluxanl.WESTTool` method), 160

`process_args()` (`westpa.cli.tools.w_fluxanl.WFluxanlTool` method), 162

`process_args()` (`westpa.cli.tools.w_ipa.ProgressIndicatorComponent` method), 46

`process_args()` (`westpa.cli.tools.w_ipa.WESTDataReader` method), 46

`process_args()` (`westpa.cli.tools.w_ipa.WESTParallelTool` method), 46

`process_args()` (`westpa.cli.tools.w_ipa.WIPI` method), 48

`process_args()` (`westpa.cli.tools.w_kinavg.WESTMasterCommand` method), 123

`process_args()` (`westpa.cli.tools.w_kinavg.WESTParallelTool` method), 124

`process_args()` (`westpa.cli.tools.w_kinetics.WESTMasterCommand` method), 129

`process_args()` (`westpa.cli.tools.w_kinetics.WESTParallelTool` method), 130

`process_args()` (`westpa.cli.tools.w_multi_west.ProgressIndicatorComponent` method), 102

`process_args()` (`westpa.cli.tools.w_multi_west.WESTMultiTool` method), 103

`process_args()` (`westpa.cli.tools.w_multi_west.WESTTool` method), 102

`process_args()` (`westpa.cli.tools.w_multi_west.WMultiWest` method), 103

`process_args()` (`westpa.cli.tools.w_ntop.IterRangeSelection` method), 97

`process_args()` (`westpa.cli.tools.w_ntop.ProgressIndicatorComponent` method), 98

`process_args()` (`westpa.cli.tools.w_ntop.WESTDataReader` method), 97

`process_args()` (`westpa.cli.tools.w_ntop.WESTTool` method), 96

`process_args()` (`westpa.cli.tools.w_ntop.WNTopTool` method), 99

`process_args()` (`westpa.cli.tools.w_pdist.IterRangeSelection` method), 53

`process_args()` (`westpa.cli.tools.w_pdist.ProgressIndicatorComponent` method), 54

`process_args()` (`westpa.cli.tools.w_pdist.WESTDataReader` method), 52

`process_args()` (`westpa.cli.tools.w_pdist.WESTDSSynthesizer` method), 52

`process_args()` (`westpa.cli.tools.w_pdist.WESTParallelTool` method), 52

`process_args()` (`westpa.cli.tools.w_pdist.WESTWDSSynthesizer` method), 53

`process_args()` (`westpa.cli.tools.w_pdist.WPDist` method), 56

`process_args()` (`westpa.cli.tools.w_postanalysis_matrix.RWMatrix` method), 144

`process_args()` (`westpa.cli.tools.w_postanalysis_matrix.WESTMasterCommand` method), 143

`process_args()` (`westpa.cli.tools.w_postanalysis_matrix.WESTParallelTool` method), 143

`process_args()` (`westpa.cli.tools.w_postanalysis_reweight.WESTMasterCommand` method), 145

`process_args()` (`westpa.cli.tools.w_postanalysis_reweight.WESTParallelTool` method), 146

`process_args()` (`westpa.cli.tools.w_red.WESTParallelTool` method), 105

`process_args()` (`westpa.cli.tools.w_reweight.AverageCommands` method), 148

`process_args()` (`westpa.cli.tools.w_reweight.RWMatrix` method), 150

`process_args()` (`westpa.cli.tools.w_reweight.RWReweight` method), 150

`process_args()` (`westpa.cli.tools.w_reweight.WESTKineticsBase` method), 148

`process_args()` (`westpa.cli.tools.w_reweight.WESTMasterCommand` method), 148

method), 147

`process_args()` (`westpa.cli.tools.w_reweight.WESTParallelTool`), 148

`process_args()` (`westpa.cli.tools.w_select.IterRangeSelection`), 79

`process_args()` (`westpa.cli.tools.w_select.ProgressIndicator`), 80

`process_args()` (`westpa.cli.tools.w_select.WESTDataReader`), 79

`process_args()` (`westpa.cli.tools.w_select.WESTParallelTool`), 78

`process_args()` (`westpa.cli.tools.w_select.WSelectTool`), 81

`process_args()` (`westpa.cli.tools.w_stateprobs.WESTMasterCommand`), 136

`process_args()` (`westpa.cli.tools.w_stateprobs.WESTParallelTool`), 136

`process_args()` (`westpa.cli.tools.w_trace.WESTDataReader`), 38

`process_args()` (`westpa.cli.tools.w_trace.WESTTool`), 37

`process_args()` (`westpa.cli.tools.w_trace.WTraceTool`), 42

`process_args()` (`westpa.oldtools.iframe.AnalysisMixin`), 301

`process_args()` (`westpa.oldtools.iframe.atool.WESTAnalysisTool`), 307

`process_args()` (`westpa.oldtools.iframe.base_mixin.AnalysisMixin`), 307

`process_args()` (`westpa.oldtools.iframe.BFDDataManager`), 304

`process_args()` (`westpa.oldtools.iframe.binning.AnalysisMixin`), 308

`process_args()` (`westpa.oldtools.iframe.binning.BinningMixin`), 308

`process_args()` (`westpa.oldtools.iframe.BinningMixin`), 304

`process_args()` (`westpa.oldtools.iframe.data_reader.AnalysisMixin`), 309

`process_args()` (`westpa.oldtools.iframe.data_reader.BFDDataManager`), 311

`process_args()` (`westpa.oldtools.iframe.data_reader.ExtDataReaderMixin`), 311

`process_args()` (`westpa.oldtools.iframe.data_reader.WESTDSSynthesizer`), 310

`process_args()` (`westpa.oldtools.iframe.ExtDataReaderMixin`), 303

`process_args()` (`westpa.oldtools.iframe.iter_range.AnalysisMixin`), 312

`process_args()` (`westpa.oldtools.iframe.iter_range.IterRangeSelection`), 312

`process_args()` (`westpa.oldtools.iframe.IterRangeMixin`), 301

`process_args()` (`westpa.oldtools.iframe.kinetics.AnalysisMixin`), 313

`process_args()` (`westpa.oldtools.iframe.kinetics.KineticsAnalysisMixin`), 313

`process_args()` (`westpa.oldtools.iframe.KineticsAnalysisMixin`), 306

`process_args()` (`westpa.oldtools.iframe.mcbs.AnalysisMixin`), 313

`process_args()` (`westpa.oldtools.iframe.mcbs.MCBSMixin`), 313

`process_args()` (`westpa.oldtools.iframe.MCBSMixin`), 305

`process_args()` (`westpa.oldtools.iframe.output.AnalysisMixin`), 314

`process_args()` (`westpa.oldtools.iframe.plotting.AnalysisMixin`), 315

`process_args()` (`westpa.oldtools.iframe.TransitionAnalysisMixin`), 306

`process_args()` (`westpa.oldtools.iframe.transitions.AnalysisMixin`), 315

`process_args()` (`westpa.oldtools.iframe.transitions.TransitionAnalysisMixin`), 317

`process_args()` (`westpa.oldtools.iframe.WESTAnalysisTool`), 301

`process_args()` (`westpa.oldtools.iframe.WESTDataReaderMixin`), 302

`process_args()` (`westpa.tools.BinMappingComponent`), 281

`process_args()` (`westpa.tools.binning.BinMappingComponent`), 285

`process_args()` (`westpa.tools.binning.WESTToolComponent`), 283

`process_args()` (`westpa.tools.core.WESTMasterCommand`), 288

`process_args()` (`westpa.tools.core.WESTMultiTool`), 287

`process_args()` (`westpa.tools.core.WESTParallelTool`), 286

`process_args()` (`westpa.tools.core.WESTTool`), 286

`process_args()` (`westpa.tools.core.WESTToolComponent`), 285

`process_args()` (`westpa.tools.data_reader.WESTDataReader`), 289

`process_args()` (`westpa.tools.data_reader.WESTDSSynthesizer`), 289

`process_args()` (`westpa.tools.data_reader.WESTToolComponent`), 288

`process_args()` (`westpa.tools.data_reader.WESTWDSSynthesizer`), 290

`process_args()` (`westpa.tools.iter_range.IterRangeSelection`), 291

`process_args()` (`westpa.tools.iter_range.WESTToolComponent`), 290

`process_args()` (`westpa.tools.IterRangeSelection`), 290

method), 279

process_args() (westpa.tools.kinetics_tool.AverageCommand method), 294

process_args() (westpa.tools.kinetics_tool.IterRangeSelection method), 292

process_args() (westpa.tools.kinetics_tool.ProgressIndicator method), 293

process_args() (westpa.tools.kinetics_tool.WESTDataReader method), 292

process_args() (westpa.tools.kinetics_tool.WESTKineticsBase method), 294

process_args() (westpa.tools.progress.ProgressIndicatorComponent method), 295

process_args() (westpa.tools.progress.WESTToolComponent method), 295

process_args() (westpa.tools.ProgressIndicatorComponent method), 281

process_args() (westpa.tools.SegSelector method), 280

process_args() (westpa.tools.selected_segs.SegSelector method), 296

process_args() (westpa.tools.selected_segs.WESTToolComponent method), 296

process_args() (westpa.tools.WESTDataReader method), 279

process_args() (westpa.tools.WESTDSSynthesizer method), 279

process_args() (westpa.tools.WESTMasterCommand method), 278

process_args() (westpa.tools.WESTMultiTool method), 278

process_args() (westpa.tools.WESTParallelTool method), 277

process_args() (westpa.tools.WESTTool method), 276

process_args() (westpa.tools.WESTToolComponent method), 277

process_args() (westpa.tools.WESTWDSSynthesizer method), 279

process_common_output_args() (westpa.cli.core.w_succ.CommonOutputMixin method), 59

process_common_output_args() (westpa.oldtools.iframe.CommonOutputMixin method), 307

process_common_output_args() (westpa.oldtools.iframe.output.CommonOutputMixin method), 314

process_config() (westpa.core.binning.mab_driver.WEDriver method), 168

process_config() (westpa.core.binning.mab_manager.WESimManager method), 170

process_config() (westpa.core.data_manager.WESTDataManager method), 195

process_config() (westpa.core.sim_manager.WESimManager method), 228

process_config() (westpa.core.we_driver.WEDriver method), 237

process_iter_chunk() (in module westpa.core.kinetics.rate_averaging), 178

process_iter_result() (westpa.cli.tools.w_crawl.WESTPACrawler method), 65

process_wm_args() (in module westpa.work_managers.environment), 245

process_wm_args() (westpa.work_managers.environment.WMEnvironment method), 245

ProcessWorkManager (class in westpa.work_managers), 241

ProcessWorkManager (class in westpa.work_managers.processes), 251

prog (westpa.cli.tools.ploterr.PloterrsTool attribute), 120

prog (westpa.cli.tools.plothist.PlotHistTool attribute), 115

prog (westpa.cli.tools.w_assign.WAssign attribute), 33

prog (westpa.cli.tools.w_bins.WBinTool attribute), 21

prog (westpa.cli.tools.w_bins.WESTTool attribute), 19

prog (westpa.cli.tools.w_crawl.WCrawl attribute), 65

prog (westpa.cli.tools.w_direct.WDirect attribute), 74

prog (westpa.cli.tools.w_dumpsegs.WDumpSegs attribute), 142

prog (westpa.cli.tools.w_dumpsegs.WESTTool attribute), 140

prog (westpa.cli.tools.w_eddist.WEDDist attribute), 92

prog (westpa.cli.tools.w_fluxanl.WESTTool attribute), 159

prog (westpa.cli.tools.w_fluxanl.WFluxanlTool attribute), 161

prog (westpa.cli.tools.w_kinavg.WDirect attribute), 126

prog (westpa.cli.tools.w_kinetics.WDirect attribute), 131

prog (westpa.cli.tools.w_multi_west.WESTTool attribute), 102

prog (westpa.cli.tools.w_multi_west.WMultiWest attribute), 103

prog (westpa.cli.tools.w_ntop.WESTTool attribute), 96

prog (westpa.cli.tools.w_ntop.WNTopTool attribute), 98

prog (westpa.cli.tools.w_pdist.WPDist attribute), 54

prog (westpa.cli.tools.w_postanalysis_matrix.WRweight attribute), 144

prog (westpa.cli.tools.w_postanalysis_reweight.WRweight attribute), 147

prog (westpa.cli.tools.w_red.WRed attribute), 106

prog (westpa.cli.tools.w_reweight.WRweight attribute), 155

prog (westpa.cli.tools.w_select.WSelectTool attribute), 80

prog (westpa.cli.tools.w_stateprobs.WDirect attribute), 139

prog (westpa.cli.tools.w_trace.WESTTool attribute), 37

prog (*westpa.cli.tools.w_trace.WTraceTool* attribute), 41
 prog (*westpa.tools.core.WESTTool* attribute), 286
 prog (*westpa.tools.WESTTool* attribute), 276
 progress (*westpa.core.progress.ProgressIndicator* property), 224
 progress (*westpa.tools.progress.ProgressIndicator* property), 295
 ProgressIndicator (class in *westpa.core.progress*), 224
 ProgressIndicator (class in *westpa.tools.progress*), 294
 ProgressIndicatorComponent (class in *westpa.cli.tools.ploterr*), 117
 ProgressIndicatorComponent (class in *westpa.cli.tools.w_assign*), 31
 ProgressIndicatorComponent (class in *westpa.cli.tools.w_crawl*), 64
 ProgressIndicatorComponent (class in *westpa.cli.tools.w_eddist*), 91
 ProgressIndicatorComponent (class in *westpa.cli.tools.w_ipa*), 46
 ProgressIndicatorComponent (class in *westpa.cli.tools.w_multi_west*), 102
 ProgressIndicatorComponent (class in *westpa.cli.tools.w_ntop*), 98
 ProgressIndicatorComponent (class in *westpa.cli.tools.w_pdist*), 54
 ProgressIndicatorComponent (class in *westpa.cli.tools.w_select*), 80
 ProgressIndicatorComponent (class in *westpa.tools*), 281
 ProgressIndicatorComponent (class in *westpa.tools.kinetics_tool*), 293
 ProgressIndicatorComponent (class in *westpa.tools.progress*), 295
 propagate() (in module *westpa.core.wm_ops*), 238
 propagate() (*westpa.core.binning.BinlessSimManager* method), 164
 propagate() (*westpa.core.binning.mab_manager.MABSimManager* attribute), 273
 propagate() (*westpa.core.binning.mab_manager.MABSimManager* method), 174
 propagate() (*westpa.core.binning.mab_manager.WESimManager* attribute), 256
 propagate() (*westpa.core.binning.mab_manager.WESimManager* method), 171
 propagate() (*westpa.core.binning.MABSimManager* method), 164
 propagate() (*westpa.core.data_manager.ExecutablePropagator* method), 194
 propagate() (*westpa.core.propagators.executable.ExecutablePropagator* method), 187
 propagate() (*westpa.core.propagators.executable.WESTPropagator* method), 181
 propagate() (*westpa.core.propagators.WESTPropagator* method), 179
 propagate() (*westpa.core.sim_manager.WESimManager* method), 229
 PropagationError, 228
 PROTOCOL_MAJOR (*westpa.work_managers.zeromq.core.ZMQCore* attribute), 261
 PROTOCOL_MAJOR (*westpa.work_managers.zeromq.node.ZMQCore* attribute), 263
 PROTOCOL_MAJOR (*westpa.work_managers.zeromq.work_manager.ZMQCore* attribute), 265
 PROTOCOL_MAJOR (*westpa.work_managers.zeromq.worker.ZMQCore* attribute), 272
 PROTOCOL_MAJOR (*westpa.work_managers.zeromq.ZMQCore* attribute), 256
 PROTOCOL_MINOR (*westpa.work_managers.zeromq.core.ZMQCore* attribute), 261
 PROTOCOL_MINOR (*westpa.work_managers.zeromq.node.ZMQCore* attribute), 263
 PROTOCOL_MINOR (*westpa.work_managers.zeromq.work_manager.ZMQCore* attribute), 265
 PROTOCOL_MINOR (*westpa.work_managers.zeromq.worker.ZMQCore* attribute), 272
 PROTOCOL_MINOR (*westpa.work_managers.zeromq.ZMQCore* attribute), 256
 PROTOCOL_UPDATE (*westpa.work_managers.zeromq.core.ZMQCore* attribute), 261
 PROTOCOL_UPDATE (*westpa.work_managers.zeromq.node.ZMQCore* attribute), 263
 PROTOCOL_UPDATE (*westpa.work_managers.zeromq.work_manager.ZMQCore* attribute), 265
 PROTOCOL_UPDATE (*westpa.work_managers.zeromq.worker.ZMQCore* attribute), 273
 PROTOCOL_UPDATE (*westpa.work_managers.zeromq.ZMQCore* attribute), 256
 PROTOCOL_VERSION (*westpa.work_managers.zeromq.core.ZMQCore* attribute), 261
 PROTOCOL_VERSION (*westpa.work_managers.zeromq.node.ZMQCore* attribute), 263
 PROTOCOL_VERSION (*westpa.work_managers.zeromq.work_manager.ZMQCore* attribute), 265
 PROTOCOL_VERSION (*westpa.work_managers.zeromq.worker.ZMQCore* attribute), 273
 PROTOCOL_VERSION (*westpa.work_managers.zeromq.ZMQCore* attribute), 256

Q

quantile() (*westpa.oldtools.stats.edfs.EDF* method), 318
 quantiles() (*westpa.oldtools.stats.edfs.EDF* method), 318

R

random_val_env_vars() (*westpa.core.data_manager.ExecutablePropagator* method), 193
 random_val_env_vars() (*westpa.core.propagators.executable.ExecutablePropagator* method), 187

- method), 186
- randomState (westpa.core.h5io.HDF5TrajectoryFile attribute), 211
- randomState (westpa.core.h5io.HDF5TrajectoryFile property), 212
- randport() (in module westpa.work_managers.zeromq.core), 259
- randport() (in module westpa.work_managers.zeromq.work_manager), 268
- rate_output_filename (westpa.cli.tools.ploterr.DirectKinetics attribute), 118
- rate_output_filename (westpa.cli.tools.ploterr.ReweightKinetics attribute), 119
- RateAverager (class in westpa.core.kinetics), 174
- RateAverager (class in westpa.core.kinetics.rate_averaging), 179
- RateAverager (class in westpa.westext.weed.weed_driver), 327
- RateAverager (class in westpa.westext.wess.wess_driver), 328
- RateCalculator (class in westpa.cli.tools.w_red), 106
- read() (westpa.core.h5io.HDF5TrajectoryFile method), 213
- read() (westpa.core.h5io.WESTIterationFile method), 219
- read() (westpa.core.propagators.executable.BytesIO method), 180
- read1() (westpa.core.propagators.executable.BytesIO method), 180
- read_as_traj() (westpa.core.h5io.HDF5TrajectoryFile method), 212
- read_as_traj() (westpa.core.h5io.WESTIterationFile method), 220
- read_data() (westpa.core.h5io.WESTIterationFile method), 220
- read_host_info() (westpa.work_managers.zeromq.work_manager.ZMQWorkManager class method), 271
- read_host_info() (westpa.work_managers.zeromq.ZMQWorkManager class method), 258
- read_restart() (westpa.core.h5io.WESTIterationFile method), 220
- readable() (westpa.core.propagators.executable.BytesIO method), 180
- readinto() (westpa.core.propagators.executable.BytesIO method), 180
- readline() (westpa.core.propagators.executable.BytesIO method), 180
- readlines() (westpa.core.propagators.executable.BytesIO method), 180
- rebin_current() (westpa.core.binning.mab_driver.WEDriver method), 169
- rebin_current() (westpa.core.we_driver.WEDriver method), 238
- recip() (westpa.westext.weed.UncertMath.UncertContainer method), 326
- RECONFIGURE_TIMEOUT (westpa.work_managers.zeromq.core.Message attribute), 260
- RECONFIGURE_TIMEOUT (westpa.work_managers.zeromq.node.Message attribute), 264
- RECONFIGURE_TIMEOUT (westpa.work_managers.zeromq.work_manager.Message attribute), 267
- RECONFIGURE_TIMEOUT (westpa.work_managers.zeromq.worker.Message attribute), 274
- record_data_binhash() (westpa.oldtools.iframe.binning.BinningMixin method), 308
- record_data_binhash() (westpa.oldtools.iframe.BinningMixin method), 304
- record_data_iter_range() (westpa.cli.tools.w_crawl.IterRangeSelection method), 64
- record_data_iter_range() (westpa.cli.tools.w_fluxanl.IterRangeSelection method), 161
- record_data_iter_range() (westpa.cli.tools.w_ntop.IterRangeSelection method), 97
- record_data_iter_range() (westpa.cli.tools.w_pdist.IterRangeSelection method), 53
- record_data_iter_range() (westpa.cli.tools.w_select.IterRangeSelection method), 79
- record_data_iter_range() (westpa.oldtools.iframe.iter_range.IterRangeMixin method), 312
- record_data_iter_range() (westpa.oldtools.iframe.IterRangeMixin method), 301
- record_data_iter_range() (westpa.tools.iter_range.IterRangeSelection method), 291
- record_data_iter_range() (westpa.tools.IterRangeSelection method), 280
- record_data_iter_range() (westpa.tools.kinetics_tool.IterRangeSelection method), 292
- record_data_iter_step() (westpa.cli.tools.w_crawl.IterRangeSelection method), 292

method), 64

`record_data_iter_step()`
(*westpa.cli.tools.w_fluxanl.IterRangeSelection method*), 161

`record_data_iter_step()`
(*westpa.cli.tools.w_ntop.IterRangeSelection method*), 97

`record_data_iter_step()`
(*westpa.cli.tools.w_pdist.IterRangeSelection method*), 53

`record_data_iter_step()`
(*westpa.cli.tools.w_select.IterRangeSelection method*), 79

`record_data_iter_step()`
(*westpa.oldtools.iframe.iter_range.IterRangeMixin method*), 312

`record_data_iter_step()`
(*westpa.oldtools.iframe.IterRangeMixin method*), 302

`record_data_iter_step()`
(*westpa.tools.iter_range.IterRangeSelection method*), 291

`record_data_iter_step()`
(*westpa.tools.IterRangeSelection method*), 280

`record_data_iter_step()`
(*westpa.tools.kinetics_tool.IterRangeSelection method*), 292

`record_transition_data()`
(*westpa.oldtools.iframe.TransitionEventAccumulator method*), 306

`record_transition_data()`
(*westpa.oldtools.iframe.transitions.TransitionEventAccumulator method*), 316

`rectilinear_assign()` (in module *westpa.core.binning.assign*), 166

`RectilinearBinMapper` (class in *westpa.core.binning*), 162

`RectilinearBinMapper` (class in *westpa.core.binning.assign*), 166

`RectilinearBinMapper` (class in *westpa.tools.binning*), 283

`RecursiveBinMapper` (class in *westpa.core.binning*), 163

`RecursiveBinMapper` (class in *westpa.core.binning.assign*), 167

`recv_ack()` (*westpa.work_managers.zeromq.core.ZMQCore method*), 262

`recv_ack()` (*westpa.work_managers.zeromq.node.ZMQCore method*), 263

`recv_ack()` (*westpa.work_managers.zeromq.work_manager.ZMQCore method*), 266

`recv_ack()` (*westpa.work_managers.zeromq.worker.ZMQCore method*), 273

`recv_ack()` (*westpa.work_managers.zeromq.ZMQCore method*), 257

`recv_all()` (*westpa.work_managers.zeromq.core.ZMQCore method*), 262

`recv_all()` (*westpa.work_managers.zeromq.node.ZMQCore method*), 263

`recv_all()` (*westpa.work_managers.zeromq.work_manager.ZMQCore method*), 266

`recv_all()` (*westpa.work_managers.zeromq.worker.ZMQCore method*), 273

`recv_all()` (*westpa.work_managers.zeromq.ZMQCore method*), 257

`recv_message()` (*westpa.work_managers.zeromq.core.ZMQCore method*), 261

`recv_message()` (*westpa.work_managers.zeromq.node.ZMQCore method*), 263

`recv_message()` (*westpa.work_managers.zeromq.work_manager.ZMQCore method*), 266

`recv_message()` (*westpa.work_managers.zeromq.worker.ZMQCore method*), 273

`recv_message()` (*westpa.work_managers.zeromq.ZMQCore method*), 257

`recycled` (*westpa.analysis.core.Walker property*), 337

`recycled_walkers` (*westpa.analysis.core.Iteration property*), 334

`recycled_walkers` (*westpa.analysis.core.Run property*), 332

`recycling_segments` (*westpa.core.binning.mab_driver.WEDriver property*), 169

`recycling_segments` (*westpa.core.we_driver.WEDriver property*), 237

`reduce_array()` (in module *westpa.westext.wess.wess_driver*), 328

`reference` (*westpa.core.h5io.HDF5TrajectoryFile attribute*), 212

`reference` (*westpa.core.h5io.HDF5TrajectoryFile property*), 212

`register_callback()`
(*westpa.core.binning.mab_manager.WESimManager method*), 170

`register_callback()`
(*westpa.core.sim_manager.WESimManager method*), 228

`relpath()` (in module *westpa.core.data_manager*), 189

`remove()` (*westpa.work_managers.mpi.deque method*), 246

`remove()` (*westpa.work_managers.zeromq.work_manager.deque method*), 271

`remove_ipc_endpoints()`
(*westpa.work_managers.zeromq.core.ZMQCore class method*), 261

`remove_ipc_endpoints()`
(*westpa.work_managers.zeromq.node.ZMQCore class method*), 263

`remove_ipc_endpoints()` (*westpa.cli.core.w_succ.WESTAnalysisTool*
(westpa.work_managers.zeromq.work_manager.ZMQCore *method)*, 58
class method), 266 `require_analysis_group()`
`remove_ipc_endpoints()` (*westpa.oldtools.iframe.atool.WESTAnalysisTool*
(westpa.work_managers.zeromq.worker.ZMQCore *method)*, 307
class method), 273 `require_analysis_group()`
`remove_ipc_endpoints()` (*westpa.oldtools.iframe.WESTAnalysisTool*
(westpa.work_managers.zeromq.ZMQCore *method)*, 301
class method), 256 `require_bf_h5file()`
`remove_solvent()` (*westpa.core.h5io.Trajectory* *(westpa.oldtools.iframe.BFDataManager*
method), 207 *method)*, 304
`remove_timer()` (*westpa.work_managers.zeromq.core.PassiveMultiTime* *require_bf_h5file()*
method), 261 *(westpa.oldtools.iframe.data_reader.BFDataManager*
method), 312
`remove_timer()` (*westpa.work_managers.zeromq.node.PassiveMultiTime* *require_bin_assignments()*
method), 265 *method)*, 308
`remove_timer()` (*westpa.work_managers.zeromq.work_manager.PassiveMultiTime* *require_bin_assignments()*
method), 268 *method)*, 308
`remove_timer()` (*westpa.work_managers.zeromq.worker.PassiveMultiTime* *require_bin_assignments()*
method), 275 *method)*, 308
`remove_worker()` (*westpa.work_managers.zeromq.work_manager.ZMQWorkManager* *require_binning_group()*
method), 272 *method)*, 308
`remove_worker()` (*westpa.work_managers.zeromq.ZMQWorkManager* *require_binning_group()*
method), 259 *method)*, 308
`replace_dataset()` (*westpa.cli.tools.w_assign.WESTPAH5File* *require_binning_group()*
method), 33 *method)*, 304
`replace_dataset()` (*westpa.core.h5io.WESTPAH5File* *require_binning_group()*
method), 219 *method)*, 304
`report_basis_states()` *require_dataset_from_dsopts()* (*in module*
(westpa.core.binning.mab_manager.WESimManager *westpa.core.data_manager)*, 197
method), 170 *require_iter_group()*
`report_basis_states()` (*westpa.cli.tools.w_assign.WESTPAH5File*
(westpa.core.sim_manager.WESimManager *method)*, 33
method), 229 *require_iter_group()*
`report_bin_statistics()` *(westpa.core.data_manager.WESTDataManager*
(westpa.core.binning.mab_manager.WESimManager *method)*, 195
method), 170 *require_iter_group()*
`report_bin_statistics()` (*westpa.core.h5io.WESTPAH5File* *method)*,
(westpa.core.sim_manager.WESimManager *219*
method), 229 *require_matplotlib()*
`report_target_states()` *(westpa.oldtools.iframe.plotting.PlottingMixin*
(westpa.core.binning.mab_manager.WESimManager *method)*, 315
method), 170 *require_matplotlib()*
`report_target_states()` (*westpa.oldtools.iframe.PlottingMixin*
(westpa.core.sim_manager.WESimManager *method)*, 307
method), 229 *require_transitions()*
`request_task()` (*westpa.work_managers.zeromq.work_manager.ZMQWorkManager* *require_transitions()*
method), 268 *method)*, 306
`request_task()` (*westpa.work_managers.zeromq.worker.ZMQWorkManager* *require_transitions()*
method), 275 *method)*, 306
`request_task()` (*westpa.work_managers.zeromq.ZMQWorkManager* *require_transitions()*
method), 258 *method)*, 306
`require()` (*westpa.core.yamlcfg.YAMLConfig* *method)*,
239 *method)*, 317
`require_analysis_group()` *require_transitions()*
(westpa.oldtools.iframe.transitions.TransitionAnalysisMixin
method), 317 *method)*, 317

method), 317

require_transitions_group()
(westpa.oldtools.iframe.TransitionAnalysisMixin
method), 305

require_transitions_group()
(westpa.oldtools.iframe.transitions.TransitionAnalysisMixin
method), 316

require_type_if_present()
(westpa.core.yamlcfg.YAMLConfig method),
239

reset() (westpa.work_managers.core.FutureWatcher
method), 244

reset() (westpa.work_managers.zeromq.core.PassiveMultiTimer
method), 261

reset() (westpa.work_managers.zeromq.core.PassiveTimer
method), 260

reset() (westpa.work_managers.zeromq.node.PassiveMultiTimer
method), 265

reset() (westpa.work_managers.zeromq.work_manager.PassiveMultiTimer
method), 268

reset() (westpa.work_managers.zeromq.worker.PassiveMultiTimer
method), 275

resolution (westpa.core.sim_manager.timedelta
attribute), 226

resolve_filepath() (in module westpa.core.h5io),
216

restart_loader() (in module
westpa.core.propagators.executable), 185

restart_writer() (in module
westpa.core.propagators.executable), 185

restrict_atoms() (westpa.core.h5io.Trajectory
method), 207

Result (class in westpa.work_managers.zeromq.core),
260

Result (class in westpa.work_managers.zeromq.work_manager),
267

Result (class in westpa.work_managers.zeromq.worker),
275

result (westpa.work_managers.core.WMFuture prop-
erty), 244

result (westpa.work_managers.mpi.WMFuture prop-
erty), 248

result (westpa.work_managers.processes.WMFuture
property), 250

result (westpa.work_managers.serial.WMFuture prop-
erty), 253

result (westpa.work_managers.threads.WMFuture
property), 255

RESULT (westpa.work_managers.zeromq.core.Message
attribute), 260

RESULT (westpa.work_managers.zeromq.node.Message
attribute), 264

RESULT (westpa.work_managers.zeromq.work_manager.Message
attribute), 267

result (westpa.work_managers.zeromq.work_manager.WMFuture
property), 270

RESULT (westpa.work_managers.zeromq.worker.Message
attribute), 274

results_loop() (westpa.work_managers.processes.ProcessWorkManager
method), 251

results_loop() (westpa.work_managers.ProcessWorkManager
method), 242

retrieve_dataset_return()
(westpa.core.data_manager.ExecutablePropagator
method), 193

retrieve_dataset_return()
(westpa.core.propagators.executable.ExecutablePropagator
method), 186

return_state_type() (in module
westpa.core.propagators.executable), 183

turn_state_type() (in module westpa.core.states),
233

reverse() (westpa.work_managers.mpi.deque method),
246

use_force() (westpa.work_managers.zeromq.work_manager.deque
method), 271

reweight (westpa.cli.tools.w_ipa.WIPI property), 48

reweight (westpa.cli.tools.w_ipa.WIPIScheme prop-
erty), 47

reweight (westpa.tools.wipi.WIPIScheme property), 297

reweight (westpa.tools.WIPIScheme property), 283

reweight() (westpa.core.binning.assign.Bin method),
165

reweight() (westpa.core.binning.Bin method), 165

reweight() (westpa.core.binning.bins.Bin method), 167

reweight_for_c() (in module
westpa.cli.tools.w_reweight), 149

reweight_for_c() (in module westpa.core.reweight),
187

ReweightKinetics (class in westpa.cli.tools.ploterr),
119

ReweightStateprobs (class in westpa.cli.tools.ploterr),
119

root (westpa.core.h5io.HDF5TrajectoryFile attribute),
211

root (westpa.core.h5io.HDF5TrajectoryFile property),
212

rotate() (westpa.work_managers.mpi.deque method),
246

rotate() (westpa.work_managers.zeromq.work_manager.deque
method), 271

Run (class in westpa.analysis.core), 332

run (westpa.analysis.core.Walker property), 336

run() (westpa.core.binning.mab_manager.WESimManager
method), 171

run() (westpa.core.sim_manager.WESimManager
method), 229

run() (westpa.work_managers.core.WorkManager

- method), 243
- run() (westpa.work_managers.mpi.WorkManager method), 247
- run() (westpa.work_managers.processes.WorkManager method), 249
- run() (westpa.work_managers.serial.WorkManager method), 252
- run() (westpa.work_managers.threads.Task method), 255
- run() (westpa.work_managers.threads.WorkManager method), 254
- run() (westpa.work_managers.zeromq.node.ZMQNode method), 265
- run() (westpa.work_managers.zeromq.work_manager.WorkManager method), 269
- run() (westpa.work_managers.zeromq.work_manager.ZMQNode method), 269
- run() (westpa.work_managers.zeromq.ZMQNode method), 258
- run_calculation() (westpa.cli.tools.w_direct.AverageCommands method), 70
- run_calculation() (westpa.cli.tools.w_reweight.AverageCommands method), 148
- run_calculation() (westpa.tools.kinetics_tool.AverageCommands method), 294
- run_simulation() (in module westpa.cli.core.w_run), 22
- run_we() (westpa.core.binning.mab_manager.WESimManager method), 171
- run_we() (westpa.core.sim_manager.WESimManager method), 229
- RunningStatsAccumulator (class in westpa.oldtools.stats), 317
- RunningStatsAccumulator (class in westpa.oldtools.stats.accumulator), 317
- runtask() (westpa.work_managers.threads.ThreadsWorkManager method), 255
- runtask() (westpa.work_managers.ThreadsWorkManager method), 241
- RWall (class in westpa.cli.tools.w_reweight), 155
- RWAverage (class in westpa.cli.tools.w_postanalysis_reweight), 146
- RWAverage (class in westpa.cli.tools.w_reweight), 155
- RWMMatrix (class in westpa.cli.tools.w_postanalysis_matrix), 143
- RWMMatrix (class in westpa.cli.tools.w_reweight), 149
- RWRate (class in westpa.cli.tools.w_reweight), 151
- RWReweight (class in westpa.cli.tools.w_reweight), 150
- RWStateProbs (class in westpa.cli.tools.w_reweight), 153
- S**
- safe_extract() (in module westpa.core.h5io), 216
- safe_extract() (in module westpa.core.propagators.executable), 184
- save() (westpa.core.h5io.Trajectory method), 204
- save_amberrst7() (westpa.core.h5io.Trajectory method), 206
- save_bin_data() (westpa.core.binning.mab_manager.WESimManager method), 171
- save_bin_data() (westpa.core.sim_manager.WESimManager method), 229
- save_bin_mapper() (westpa.core.data_manager.WESTDataManager method), 197
- save_binpos() (westpa.core.h5io.Trajectory method), 205
- save_binpos() (westpa.core.h5io.Trajectory method), 205
- save_dtr() (westpa.core.h5io.Trajectory method), 205
- save_gro() (westpa.core.h5io.Trajectory method), 206
- save_gsd() (westpa.core.h5io.Trajectory method), 206
- save_hdf5() (westpa.core.h5io.Trajectory method), 204
- save_iter_binning() (westpa.core.data_manager.WESTDataManager method), 197
- save_iter_binning() (westpa.core.h5io.Trajectory method), 204
- save_new_weight_data() (westpa.core.data_manager.WESTDataManager method), 197
- save_pdb() (westpa.core.h5io.Trajectory method), 204
- save_target_states() (westpa.core.data_manager.WESTDataManager method), 195
- save_tng() (westpa.core.h5io.Trajectory method), 206
- save_trr() (westpa.core.h5io.Trajectory method), 205
- save_xtc() (westpa.core.h5io.Trajectory method), 204
- save_xyz() (westpa.core.h5io.Trajectory method), 204
- scan_data_range() (westpa.cli.tools.w_eddist.WEDDist method), 94
- scan_data_range() (westpa.cli.tools.w_pdist.WPDist method), 56
- scan_data_shape() (westpa.cli.tools.w_eddist.WEDDist method), 94
- scan_data_shape() (westpa.cli.tools.w_pdist.WPDist method), 56
- scheme (westpa.cli.tools.w_ipa.WIPI property), 48
- scheme (westpa.cli.tools.w_ipa.WIPIScheme property), 47
- scheme (westpa.tools.wipi.WIPIScheme property), 297
- scheme (westpa.tools.WIPIScheme property), 282

seconds (<i>westpa.core.sim_manager.timedelta</i> attribute), 226	SEG_ENDPOINT_MERGED (<i>westpa.cli.core.w_succ.Segment</i> attribute), 57
seek() (<i>westpa.core.h5io.HDF5TrajectoryFile</i> method), 214	SEG_ENDPOINT_MERGED (<i>westpa.cli.tools.w_dumpsegs.Segment</i> attribute), 141
seek() (<i>westpa.core.propagators.executable.BytesIO</i> method), 180	SEG_ENDPOINT_MERGED (<i>westpa.cli.tools.w_trace.Segment</i> attribute), 39
seekable() (<i>westpa.core.propagators.executable.BytesIO</i> method), 181	SEG_ENDPOINT_MERGED (<i>westpa.core.binning.mab_manager.Segment</i> attribute), 173
SEG_ENDPOINT_CONTINUES (<i>westpa.cli.core.w_fork.Segment</i> attribute), 24	SEG_ENDPOINT_MERGED (<i>westpa.core.data_manager.Segment</i> attribute), 189
SEG_ENDPOINT_CONTINUES (<i>westpa.cli.core.w_states.Segment</i> attribute), 85	SEG_ENDPOINT_MERGED (<i>westpa.core.propagators.executable.Segment</i> attribute), 184
SEG_ENDPOINT_CONTINUES (<i>westpa.cli.core.w_succ.Segment</i> attribute), 57	SEG_ENDPOINT_MERGED (<i>westpa.core.segment.Segment</i> attribute), 225
SEG_ENDPOINT_CONTINUES (<i>westpa.cli.tools.w_dumpsegs.Segment</i> attribute), 141	SEG_ENDPOINT_MERGED (<i>westpa.core.sim_manager.Segment</i> attribute), 227
SEG_ENDPOINT_CONTINUES (<i>westpa.cli.tools.w_trace.Segment</i> attribute), 39	SEG_ENDPOINT_MERGED (<i>westpa.core.states.Segment</i> attribute), 230
SEG_ENDPOINT_CONTINUES (<i>westpa.core.binning.mab_manager.Segment</i> attribute), 173	SEG_ENDPOINT_MERGED (<i>westpa.core.we_driver.Segment</i> attribute), 235
SEG_ENDPOINT_CONTINUES (<i>westpa.core.data_manager.Segment</i> attribute), 189	SEG_ENDPOINT_MERGED (<i>westpa.oldtools.iframe.data_reader.Segment</i> attribute), 309
SEG_ENDPOINT_CONTINUES (<i>westpa.core.propagators.executable.Segment</i> attribute), 184	SEG_ENDPOINT_RECYCLED (<i>westpa.cli.core.w_fork.Segment</i> attribute), 24
SEG_ENDPOINT_CONTINUES (<i>westpa.core.segment.Segment</i> attribute), 225	SEG_ENDPOINT_RECYCLED (<i>westpa.cli.core.w_states.Segment</i> attribute), 85
SEG_ENDPOINT_CONTINUES (<i>westpa.core.sim_manager.Segment</i> attribute), 226	SEG_ENDPOINT_RECYCLED (<i>westpa.cli.core.w_succ.Segment</i> attribute), 57
SEG_ENDPOINT_CONTINUES (<i>westpa.core.states.Segment</i> attribute), 230	SEG_ENDPOINT_RECYCLED (<i>westpa.cli.tools.w_dumpsegs.Segment</i> attribute), 141
SEG_ENDPOINT_CONTINUES (<i>westpa.core.we_driver.Segment</i> attribute), 235	SEG_ENDPOINT_RECYCLED (<i>westpa.cli.tools.w_trace.Segment</i> attribute), 39
SEG_ENDPOINT_CONTINUES (<i>westpa.oldtools.iframe.data_reader.Segment</i> attribute), 309	SEG_ENDPOINT_RECYCLED (<i>westpa.core.binning.mab_manager.Segment</i> attribute), 173
seg_endpoint_dtype (in <i>westpa.core.data_manager</i>), 194	SEG_ENDPOINT_RECYCLED (<i>westpa.core.data_manager.Segment</i> attribute), 189
SEG_ENDPOINT_MERGED (<i>westpa.cli.core.w_fork.Segment</i> attribute), 24	SEG_ENDPOINT_RECYCLED (<i>westpa.core.propagators.executable.Segment</i>
SEG_ENDPOINT_MERGED (<i>westpa.cli.core.w_states.Segment</i> attribute), 85	

attribute), 184
 SEG_ENDPOINT_RECYCLED (westpa.core.segment.Segment attribute), 225
 SEG_ENDPOINT_RECYCLED (westpa.core.sim_manager.Segment attribute), 227
 SEG_ENDPOINT_RECYCLED (westpa.core.states.Segment attribute), 230
 SEG_ENDPOINT_RECYCLED (westpa.core.we_driver.Segment attribute), 235
 SEG_ENDPOINT_RECYCLED (westpa.oldtools.iframe.data_reader.Segment attribute), 309
 SEG_ENDPOINT_UNSET (westpa.cli.core.w_fork.Segment attribute), 24
 SEG_ENDPOINT_UNSET (westpa.cli.core.w_states.Segment attribute), 85
 SEG_ENDPOINT_UNSET (westpa.cli.core.w_succ.Segment attribute), 57
 SEG_ENDPOINT_UNSET (westpa.cli.tools.w_dumpsegs.Segment attribute), 141
 SEG_ENDPOINT_UNSET (westpa.cli.tools.w_trace.Segment attribute), 39
 SEG_ENDPOINT_UNSET (westpa.core.binning.mab_manager.Segment attribute), 173
 SEG_ENDPOINT_UNSET (westpa.core.data_manager.Segment attribute), 189
 SEG_ENDPOINT_UNSET (westpa.core.propagators.executable.Segment attribute), 184
 SEG_ENDPOINT_UNSET (westpa.core.segment.Segment attribute), 225
 SEG_ENDPOINT_UNSET (westpa.core.sim_manager.Segment attribute), 226
 SEG_ENDPOINT_UNSET (westpa.core.states.Segment attribute), 230
 SEG_ENDPOINT_UNSET (westpa.core.we_driver.Segment attribute), 234
 SEG_ENDPOINT_UNSET (westpa.core.sim_manager.Segment attribute), 226
 SEG_ENDPOINT_UNSET (westpa.core.states.Segment attribute), 230
 SEG_ENDPOINT_UNSET (westpa.core.we_driver.Segment attribute), 235
 SEG_ENDPOINT_UNSET (westpa.oldtools.iframe.data_reader.Segment attribute), 309
 seg_id (westpa.trajtree.trajtree.trajnode attribute), 300
 seg_id_dtype (in module westpa.cli.core.w_fork), 26
 seg_id_dtype (in module westpa.cli.tools.w_assign), 28
 seg_id_dtype (in module westpa.cli.tools.w_ntop), 98
 seg_id_dtype (in module westpa.cli.tools.w_select), 80
 seg_id_dtype (in module westpa.cli.tools.w_trace), 40
 seg_id_dtype (in module westpa.core.data_manager), 194
 seg_id_dtype (in module westpa.tools.dtypes), 290
 seg_id_dtype (in module westpa.tools.selected_segs), 296
 SEG_INITPOINT_CONTINUES (westpa.cli.core.w_fork.Segment attribute), 24
 SEG_INITPOINT_CONTINUES (westpa.cli.core.w_states.Segment attribute), 85
 SEG_INITPOINT_CONTINUES (westpa.cli.core.w_succ.Segment attribute), 57
 SEG_INITPOINT_CONTINUES (westpa.cli.tools.w_dumpsegs.Segment attribute), 141
 SEG_INITPOINT_CONTINUES (westpa.cli.tools.w_trace.Segment attribute), 38
 SEG_INITPOINT_CONTINUES (westpa.core.binning.mab_manager.Segment attribute), 173
 SEG_INITPOINT_CONTINUES (westpa.core.data_manager.Segment attribute), 189
 SEG_INITPOINT_CONTINUES (westpa.core.propagators.executable.Segment attribute), 184
 SEG_INITPOINT_CONTINUES (westpa.core.segment.Segment attribute), 225
 SEG_INITPOINT_CONTINUES (westpa.core.sim_manager.Segment attribute), 226
 SEG_INITPOINT_CONTINUES (westpa.core.states.Segment attribute), 230
 SEG_INITPOINT_CONTINUES (westpa.core.we_driver.Segment attribute), 234
 SEG_INITPOINT_CONTINUES (westpa.oldtools.iframe.data_reader.Segment attribute), 309
 seg_initpoint_dtype (in module westpa.core.data_manager), 194
 SEG_INITPOINT_NEWTRAJ (westpa.cli.core.w_fork.Segment attribute), 24
 SEG_INITPOINT_NEWTRAJ (westpa.cli.core.w_states.Segment attribute), 85
 SEG_INITPOINT_NEWTRAJ (westpa.cli.core.w_succ.Segment attribute), 57
 SEG_INITPOINT_NEWTRAJ (westpa.cli.tools.w_dumpsegs.Segment attribute), 141
 SEG_INITPOINT_NEWTRAJ (westpa.cli.tools.w_trace.Segment attribute), 38
 SEG_INITPOINT_NEWTRAJ

<i>(westpa.core.binning.mab_manager.Segment attribute), 173</i>	<i>(westpa.core.we_driver.Segment attribute), 234</i>
SEG_INITPOINT_NEWTRAJ <i>(westpa.core.data_manager.Segment attribute), 189</i>	SEG_INITPOINT_UNSET <i>(westpa.oldtools.iframe.data_reader.Segment attribute), 309</i>
SEG_INITPOINT_NEWTRAJ <i>(westpa.core.propagators.executable.Segment attribute), 184</i>	seg_label_values() <i>(westpa.core.h5io.WESTTrajectory method), 215</i>
SEG_INITPOINT_NEWTRAJ <i>(westpa.core.segment.Segment attribute), 225</i>	seg_labels <i>(westpa.core.h5io.WESTTrajectory property), 215</i>
SEG_INITPOINT_NEWTRAJ <i>(westpa.core.sim_manager.Segment attribute), 226</i>	SEG_STATUS_COMPLETE <i>(westpa.cli.core.w_fork.Segment attribute), 24</i>
SEG_INITPOINT_NEWTRAJ <i>(westpa.core.states.Segment attribute), 230</i>	SEG_STATUS_COMPLETE <i>(westpa.cli.core.w_states.Segment attribute), 85</i>
SEG_INITPOINT_NEWTRAJ <i>(westpa.core.we_driver.Segment attribute), 235</i>	SEG_STATUS_COMPLETE <i>(westpa.cli.core.w_succ.Segment attribute), 57</i>
SEG_INITPOINT_NEWTRAJ <i>(westpa.oldtools.iframe.data_reader.Segment attribute), 309</i>	SEG_STATUS_COMPLETE <i>(westpa.cli.tools.w_dumpsegs.Segment attribute), 141</i>
SEG_INITPOINT_UNSET <i>(westpa.cli.core.w_fork.Segment attribute), 24</i>	SEG_STATUS_COMPLETE <i>(westpa.cli.tools.w_trace.Segment attribute), 38</i>
SEG_INITPOINT_UNSET <i>(westpa.cli.core.w_states.Segment attribute), 85</i>	SEG_STATUS_COMPLETE <i>(westpa.core.binning.mab_manager.Segment attribute), 173</i>
SEG_INITPOINT_UNSET <i>(westpa.cli.core.w_succ.Segment attribute), 57</i>	SEG_STATUS_COMPLETE <i>(westpa.core.data_manager.Segment attribute), 189</i>
SEG_INITPOINT_UNSET <i>(westpa.cli.tools.w_dumpsegs.Segment attribute), 141</i>	SEG_STATUS_COMPLETE <i>(westpa.core.propagators.executable.Segment attribute), 183</i>
SEG_INITPOINT_UNSET <i>(westpa.cli.tools.w_trace.Segment attribute), 38</i>	SEG_STATUS_COMPLETE <i>(westpa.core.segment.Segment attribute), 224</i>
SEG_INITPOINT_UNSET <i>(westpa.core.binning.mab_manager.Segment attribute), 173</i>	SEG_STATUS_COMPLETE <i>(westpa.core.sim_manager.Segment attribute), 226</i>
SEG_INITPOINT_UNSET <i>(westpa.core.data_manager.Segment attribute), 189</i>	SEG_STATUS_COMPLETE <i>(westpa.core.states.Segment attribute), 230</i>
SEG_INITPOINT_UNSET <i>(westpa.core.propagators.executable.Segment attribute), 184</i>	SEG_STATUS_COMPLETE <i>(westpa.core.we_driver.Segment attribute), 234</i>
SEG_INITPOINT_UNSET <i>(westpa.core.segment.Segment attribute), 224</i>	SEG_STATUS_COMPLETE <i>(westpa.oldtools.iframe.data_reader.Segment attribute), 309</i>
SEG_INITPOINT_UNSET <i>(westpa.core.sim_manager.Segment attribute), 226</i>	seg_status_dtype <i>(in module westpa.core.data_manager), 194</i>
SEG_INITPOINT_UNSET <i>(westpa.core.states.Segment attribute), 230</i>	SEG_STATUS_FAILED <i>(westpa.cli.core.w_fork.Segment attribute), 24</i>
SEG_INITPOINT_UNSET	SEG_STATUS_FAILED <i>(westpa.cli.core.w_states.Segment attribute), 85</i>
	SEG_STATUS_FAILED <i>(westpa.cli.core.w_succ.Segment attribute), 57</i>

SEG_STATUS_FAILED (*westpa.cli.tools.w_dumpsegs.Segment* attribute), 141
 SEG_STATUS_FAILED (*westpa.cli.tools.w_trace.Segment* attribute), 38
 SEG_STATUS_FAILED (*westpa.core.binning.mab_manager.Segment* attribute), 173
 SEG_STATUS_FAILED (*westpa.core.data_manager.Segment* attribute), 189
 SEG_STATUS_FAILED (*westpa.core.propagators.executable.Segment* attribute), 183
 SEG_STATUS_FAILED (*westpa.core.segment.Segment* attribute), 224
 SEG_STATUS_FAILED (*westpa.core.sim_manager.Segment* attribute), 226
 SEG_STATUS_FAILED (*westpa.core.states.Segment* attribute), 230
 SEG_STATUS_FAILED (*westpa.core.we_driver.Segment* attribute), 234
 SEG_STATUS_FAILED (*westpa.oldtools.iframe.data_reader.Segment* attribute), 309
 SEG_STATUS_PREPARED (*westpa.cli.core.w_fork.Segment* attribute), 24
 SEG_STATUS_PREPARED (*westpa.cli.core.w_states.Segment* attribute), 85
 SEG_STATUS_PREPARED (*westpa.cli.core.w_succ.Segment* attribute), 57
 SEG_STATUS_PREPARED (*westpa.cli.tools.w_dumpsegs.Segment* attribute), 141
 SEG_STATUS_PREPARED (*westpa.cli.tools.w_trace.Segment* attribute), 38
 SEG_STATUS_PREPARED (*westpa.core.binning.mab_manager.Segment* attribute), 173
 SEG_STATUS_PREPARED (*westpa.core.data_manager.Segment* attribute), 189
 SEG_STATUS_PREPARED (*westpa.core.propagators.executable.Segment* attribute), 183
 SEG_STATUS_PREPARED (*westpa.core.segment.Segment* attribute), 224
 SEG_STATUS_PREPARED (*westpa.core.sim_manager.Segment* attribute), 226
 SEG_STATUS_PREPARED (*westpa.core.states.Segment* attribute), 230
 SEG_STATUS_PREPARED (*westpa.core.we_driver.Segment* attribute), 234
 SEG_STATUS_PREPARED (*westpa.oldtools.iframe.data_reader.Segment* attribute), 308
 seglog_loader() (in module *westpa.core.propagators.executable*), 185
 Segment (class in *westpa.cli.core.w_fork*), 24
 Segment (class in *westpa.cli.core.w_states*), 84
 Segment (class in *westpa.cli.core.w_succ*), 57
 Segment (class in *westpa.cli.tools.w_dumpsegs*), 141
 Segment (class in *westpa.cli.tools.w_trace*), 38
 Segment (class in *westpa.core.binning.mab_manager*), 172
 Segment (class in *westpa.core.data_manager*), 189
 Segment (class in *westpa.core.propagators.executable*), 183
 Segment (class in *westpa.core.segment*), 224
 Segment (class in *westpa.core.sim_manager*), 226
 Segment (class in *westpa.core.states*), 230
 Segment (class in *westpa.core.we_driver*), 234
 Segment (class in *westpa.oldtools.iframe.data_reader*), 308
 segment_collector (*westpa.analysis.trajectories.Trajectory* property), 338
 segment_summaries (*westpa.analysis.core.Iteration* property), 333
 segment_summary (*westpa.analysis.core.Walker* property), 336
 SegmentCollector (class in *westpa.analysis.core*), 336

`westpa.analysis.trajectories`), 338
`SegmentSelection` (class in `westpa.tools.selected_segs`), 296
`SegSelector` (class in `westpa.tools`), 280
`SegSelector` (class in `westpa.tools.selected_segs`), 296
`selected_bin_pair_iter` (`westpa.oldtools.iframe.kinetics.KineticsAnalysisMixin` property), 313
`selected_bin_pair_iter` (`westpa.oldtools.iframe.KineticsAnalysisMixin` property), 307
`send_ack()` (`westpa.work_managers.zeromq.core.ZMQCore` method), 262
`send_ack()` (`westpa.work_managers.zeromq.node.ZMQCore` method), 264
`send_ack()` (`westpa.work_managers.zeromq.work_manager.ZMQCore` method), 266
`send_ack()` (`westpa.work_managers.zeromq.worker.ZMQCore` method), 274
`send_ack()` (`westpa.work_managers.zeromq.ZMQCore` method), 257
`send_inproc_message()` (`westpa.work_managers.zeromq.core.ZMQCore` method), 262
`send_inproc_message()` (`westpa.work_managers.zeromq.node.ZMQCore` method), 264
`send_inproc_message()` (`westpa.work_managers.zeromq.work_manager.ZMQCore` method), 267
`send_inproc_message()` (`westpa.work_managers.zeromq.worker.ZMQCore` method), 274
`send_inproc_message()` (`westpa.work_managers.zeromq.ZMQCore` method), 257
`send_message()` (`westpa.work_managers.zeromq.core.ZMQCore` method), 262
`send_message()` (`westpa.work_managers.zeromq.node.ZMQCore` method), 264
`send_message()` (`westpa.work_managers.zeromq.work_manager.ZMQCore` method), 266
`send_message()` (`westpa.work_managers.zeromq.work_manager.ZMQWorkManager` method), 272
`send_message()` (`westpa.work_managers.zeromq.worker.ZMQCore` method), 273
`send_message()` (`westpa.work_managers.zeromq.ZMQCore` method), 257
`send_message()` (`westpa.work_managers.zeromq.ZMQWorkManager` method), 259
`send_nak()` (`westpa.work_managers.zeromq.core.ZMQCore` method), 262
`send_nak()` (`westpa.work_managers.zeromq.node.ZMQCore` method), 264
`send_nak()` (`westpa.work_managers.zeromq.work_manager.ZMQCore` method), 267
`send_nak()` (`westpa.work_managers.zeromq.worker.ZMQCore` method), 274
`send_nak()` (`westpa.work_managers.zeromq.ZMQCore` method), 257
`send_reply()` (`westpa.work_managers.zeromq.core.ZMQCore` method), 262
`send_reply()` (`westpa.work_managers.zeromq.node.ZMQCore` method), 264
`send_reply()` (`westpa.work_managers.zeromq.work_manager.ZMQCore` method), 266
`send_reply()` (`westpa.work_managers.zeromq.worker.ZMQCore` method), 273
`send_reply()` (`westpa.work_managers.zeromq.ZMQCore` method), 257
`sequence_macro_flux_to_rate()` (in module `westpa.cli.tools.w_direct`), 69
`sequence_macro_flux_to_rate()` (in module `westpa.core.kinetics`), 175
`Serial` (class in `westpa.work_managers.mpi`), 248
`SerialWorkManager` (class in `westpa.work_managers`), 241
`SerialWorkManager` (class in `westpa.work_managers.serial`), 253
`set()` (`westpa.cli.tools.plothist.NonUniformImage` method), 112
`set_arg_default()` (`westpa.tools.binning.WESTToolComponent` method), 283
`set_arg_default()` (`westpa.tools.core.WESTToolComponent` method), 285
`set_arg_default()` (`westpa.tools.data_reader.WESTToolComponent` method), 288
`set_arg_default()` (`westpa.tools.iter_range.WESTToolComponent` method), 290
`set_arg_default()` (`westpa.tools.progress.WESTToolComponent` method), 295
`set_arg_default()` (`westpa.tools.selected_segs.WESTToolComponent` method), 295
`set_arg_default()` (`westpa.tools.WESTToolComponent` method), 277
`set_array()` (`westpa.cli.tools.plothist.NonUniformImage` method), 112
`set_cmap()` (`westpa.cli.tools.plothist.NonUniformImage` method), 111
`set_data()` (`westpa.cli.tools.plothist.NonUniformImage` method), 111
`set_filternorm()` (`westpa.cli.tools.plothist.NonUniformImage` method), 112
`set_filterrad()` (`westpa.cli.tools.plothist.NonUniformImage` method), 112
`set_interpolation()` (`westpa.cli.tools.plothist.NonUniformImage` method), 111

<code>set_norm()</code> (<i>westpa.cli.tools.plothist.NonUniformImage</i> method), 112	<code>shutdown()</code> (<i>westpa.work_managers.ProcessWorkManager</i> method), 242
<code>set_other_h5file_info()</code> (<i>westpa.cli.tools.w_assign.BinMappingComponent</i> method), 31	<code>shutdown()</code> (<i>westpa.work_managers.serial.WorkManager</i> method), 252
<code>set_other_h5file_info()</code> (<i>westpa.cli.tools.w_bins.BinMappingComponent</i> method), 20	<code>shutdown()</code> (<i>westpa.work_managers.threads.ThreadsWorkManager</i> method), 255
<code>set_other_h5file_info()</code> (<i>westpa.tools.BinMappingComponent</i> method), 281	<code>shutdown()</code> (<i>westpa.work_managers.threads.WorkManager</i> method), 254
<code>set_other_h5file_info()</code> (<i>westpa.tools.binning.BinMappingComponent</i> method), 285	<code>shutdown()</code> (<i>westpa.work_managers.ThreadsWorkManager</i> method), 241
<code>set_state()</code> (<i>westpa.oldtools.aframe.TransitionEventAccumulator</i> method), 306	<code>shutdown()</code> (<i>westpa.work_managers.zeromq.core.IsNode</i> method), 262
<code>set_state()</code> (<i>westpa.oldtools.aframe.transitions.TransitionEventManager</i> method), 316	<code>shutdown()</code> (<i>westpa.work_managers.zeromq.core.ZMQCore</i> method), 262
<code>set_we_h5file_info()</code> (<i>westpa.cli.tools.w_assign.BinMappingComponent</i> method), 30	<code>shutdown()</code> (<i>westpa.work_managers.zeromq.node.IsNode</i> method), 265
<code>set_we_h5file_info()</code> (<i>westpa.cli.tools.w_bins.BinMappingComponent</i> method), 20	<code>shutdown()</code> (<i>westpa.work_managers.zeromq.node.ZMQCore</i> method), 264
<code>set_we_h5file_info()</code> (<i>westpa.tools.BinMappingComponent</i> method), 281	<code>shutdown()</code> (<i>westpa.work_managers.zeromq.work_manager.IsNode</i> method), 268
<code>set_we_h5file_info()</code> (<i>westpa.tools.binning.BinMappingComponent</i> method), 285	<code>shutdown()</code> (<i>westpa.work_managers.zeromq.work_manager.WorkManager</i> method), 269
<code>setup_dataset_return()</code> (<i>westpa.core.data_manager.ExecutablePropagator</i> method), 193	<code>shutdown()</code> (<i>westpa.work_managers.zeromq.work_manager.ZMQCore</i> method), 267
<code>setup_dataset_return()</code> (<i>westpa.core.propagators.executable.ExecutablePropagator</i> method), 186	<code>shutdown()</code> (<i>westpa.work_managers.zeromq.work_manager.ZMQWorkManager</i> method), 272
<code>SHUTDOWN</code> (<i>westpa.work_managers.zeromq.core.Message</i> attribute), 260	<code>shutdown()</code> (<i>westpa.work_managers.zeromq.worker.ZMQCore</i> method), 274
<code>SHUTDOWN</code> (<i>westpa.work_managers.zeromq.node.Message</i> attribute), 264	<code>shutdown()</code> (<i>westpa.work_managers.zeromq.ZMQCore</i> method), 257
<code>SHUTDOWN</code> (<i>westpa.work_managers.zeromq.work_manager.Message</i> attribute), 267	<code>shutdown()</code> (<i>westpa.work_managers.zeromq.ZMQWorkManager</i> method), 259
<code>SHUTDOWN</code> (<i>westpa.work_managers.zeromq.worker.Message</i> attribute), 274	<code>shutdown_clear_tasks()</code> (<i>westpa.work_managers.zeromq.work_manager.ZMQWorkManager</i> method), 272
<code>shutdown()</code> (<i>westpa.work_managers.core.WorkManager</i> method), 243	<code>shutdown_clear_tasks()</code> (<i>westpa.work_managers.zeromq.ZMQWorkManager</i> method), 259
<code>shutdown()</code> (<i>westpa.work_managers.mpi.Manager</i> method), 249	<code>shutdown_executor()</code> (<i>westpa.work_managers.zeromq.work_manager.ZMQWorker</i> method), 269
<code>shutdown()</code> (<i>westpa.work_managers.mpi.WorkManager</i> method), 247	<code>shutdown_executor()</code> (<i>westpa.work_managers.zeromq.worker.ZMQWorker</i> method), 275
<code>shutdown()</code> (<i>westpa.work_managers.processes.ProcessWorkManager</i> method), 251	<code>shutdown_executor()</code> (<i>westpa.work_managers.zeromq.ZMQWorker</i> method), 258
<code>shutdown()</code> (<i>westpa.work_managers.processes.WorkManager</i> method), 249	<code>shutdown_handler()</code> (<i>westpa.work_managers.zeromq.core.ZMQCore</i> method), 262
	<code>shutdown_handler()</code> (<i>westpa.work_managers.zeromq.node.ZMQCore</i> method), 264
	<code>shutdown_handler()</code> (<i>westpa.work_managers.zeromq.work_manager.ZMQCore</i> method), 267
	<code>shutdown_handler()</code> (<i>westpa.work_managers.zeromq.worker.ZMQCore</i> method), 274

- method), 274
- shutdown_handler() (westpa.work_managers.zeromq.ZMQCore method), 257
- shutdown_process() (in module westpa.work_managers.zeromq.core), 262
- sigint_handler() (westpa.work_managers.core.WorkManager method), 243
- sigint_handler() (westpa.work_managers.mpi.WorkManager method), 247
- sigint_handler() (westpa.work_managers.processes.WorkManager method), 249
- sigint_handler() (westpa.work_managers.serial.WorkManager method), 252
- sigint_handler() (westpa.work_managers.threads.WorkManager method), 254
- sigint_handler() (westpa.work_managers.zeromq.work_manager.WorkManager method), 269
- signal() (westpa.work_managers.core.FutureWatcher method), 244
- signal_shutdown() (westpa.work_managers.zeromq.core.ZMQCore method), 262
- signal_shutdown() (westpa.work_managers.zeromq.node.ZMQCore method), 264
- signal_shutdown() (westpa.work_managers.zeromq.work_manager.ZMQCore method), 267
- signal_shutdown() (westpa.work_managers.zeromq.work_manager.ZMQCore method), 274
- signal_shutdown() (westpa.work_managers.zeromq.ZMQCore method), 257
- SingleDSSpec (class in westpa.core.h5io), 221
- SingleIterDSSpec (class in westpa.core.h5io), 221
- SingleIterDSSpec (class in westpa.tools.data_reader), 289
- SingleSegmentDSSpec (class in westpa.core.h5io), 221
- SingleSegmentDSSpec (class in westpa.tools.data_reader), 289
- sink (westpa.analysis.core.Iteration property), 335
- slice() (westpa.core.h5io.Trajectory method), 202
- slice() (westpa.core.h5io.WESTTrajectory method), 216
- slice_per_iter_data() (westpa.cli.tools.w_crawl.IterRangeSelection method), 64
- slice_per_iter_data() (westpa.cli.tools.w_fluxanl.IterRangeSelection method), 161
- slice_per_iter_data() (westpa.cli.tools.w_ntop.IterRangeSelection method), 98
- slice_per_iter_data() (westpa.cli.tools.w_pdist.IterRangeSelection method), 54
- slice_per_iter_data() (westpa.cli.tools.w_select.IterRangeSelection method), 80
- slice_per_iter_data() (westpa.oldtools.iframe.iter_range.IterRangeMixin method), 313
- slice_per_iter_data() (westpa.oldtools.iframe.IterRangeMixin method), 302
- slice_per_iter_data() (westpa.tools.iter_range.IterRangeSelection method), 291
- slice_per_iter_data() (westpa.tools.IterRangeSelection method), 280
- slice_per_iter_data() (westpa.tools.kinetics_tool.IterRangeSelection method), 280
- smallest_allowed_weight (westpa.core.binning.mab_driver.WEDriver attribute), 168
- smallest_allowed_weight (westpa.core.we_driver.WEDriver attribute), 168
- smooth() (westpa.core.h5io.Trajectory method), 208
- solve() (westpa.core.h5io.Trajectory method), 201
- stamp_creator_data() (in module westpa.core.h5io), 216
- stamp_hash() (westpa.cli.tools.w_ipa.WIPI method), 48
- stamp_iter_range() (in module westpa.core.h5io), 217
- stamp_iter_step() (in module westpa.core.h5io), 217
- stamp_mcbs_info() (westpa.cli.tools.w_direct.AverageCommands method), 70
- stamp_mcbs_info() (westpa.cli.tools.w_reweight.AverageCommands method), 148
- stamp_mcbs_info() (westpa.tools.kinetics_tool.AverageCommands method), 294
- start() (westpa.core.progress.ProgressIndicator method), 224
- start() (westpa.tools.progress.ProgressIndicator method), 295
- start() (westpa.work_managers.zeromq.core.PassiveTimer method), 260
- start_accumulation() (westpa.oldtools.iframe.TransitionEventAccumulator method), 306
- start_accumulation() (westpa.oldtools.iframe.transitions.TransitionEventAccumulator method), 316
- start_index (westpa.core.binning.assign.RecursiveBinMapper property), 167
- start_index (westpa.core.binning.RecursiveBinMapper property), 163

`start_iter(westpa.tools.selected_segs.SegmentSelection property), 296`
`started(westpa.work_managers.zeromq.core.PassiveTime attribute), 260`
`startup() (westpa.work_managers.core.WorkManager method), 243`
`startup() (westpa.work_managers.mpi.Manager method), 249`
`startup() (westpa.work_managers.mpi.Worker method), 249`
`startup() (westpa.work_managers.mpi.WorkManager method), 247`
`startup() (westpa.work_managers.processes.ProcessWorkManager method), 251`
`startup() (westpa.work_managers.processes.WorkManager method), 249`
`startup() (westpa.work_managers.ProcessWorkManager method), 242`
`startup() (westpa.work_managers.serial.WorkManager method), 252`
`startup() (westpa.work_managers.threads.ThreadsWorkManager method), 255`
`startup() (westpa.work_managers.threads.WorkManager method), 254`
`startup() (westpa.work_managers.ThreadsWorkManager method), 241`
`startup() (westpa.work_managers.zeromq.core.IsNode method), 262`
`startup() (westpa.work_managers.zeromq.core.ZMQCore method), 262`
`startup() (westpa.work_managers.zeromq.node.IsNode method), 265`
`startup() (westpa.work_managers.zeromq.node.ZMQCore method), 264`
`startup() (westpa.work_managers.zeromq.node.ZMQNode method), 265`
`startup() (westpa.work_managers.zeromq.work_manager.IsNode method), 268`
`startup() (westpa.work_managers.zeromq.work_manager.WorkManager method), 269`
`startup() (westpa.work_managers.zeromq.work_manager.ZMQCore class method), 267`
`startup() (westpa.work_managers.zeromq.work_manager.ZMQNode class method), 269`
`startup() (westpa.work_managers.zeromq.work_manager.ZMQWorker class method), 269`
`startup() (westpa.work_managers.zeromq.work_manager.ZMQWorkManager class method), 272`
`startup() (westpa.work_managers.zeromq.worker.ZMQCore method), 274`
`startup() (westpa.work_managers.zeromq.worker.ZMQExecutor method), 276`
`startup() (westpa.work_managers.zeromq.worker.ZMQWorker method), 275`
`startup() (westpa.work_managers.zeromq.ZMQCore method), 257`
`startup() (westpa.work_managers.zeromq.ZMQNode method), 258`
`startup() (westpa.work_managers.zeromq.ZMQWorker method), 258`
`startup() (westpa.work_managers.zeromq.ZMQWorkManager method), 259`
`state_labels (westpa.cli.tools.w_ipa.WIPI property), 48`
`state_labels (westpa.cli.tools.w_ipa.WIPIScheme property), 47`
`state_labels (westpa.tools.wipi.WIPIScheme property), 297`
`state_labels (westpa.tools.WIPIScheme property), 282`
`states_from_dict() (westpa.cli.tools.w_assign.WAssign method), 35`
`states_from_file() (westpa.cli.core.w_init.BasisState class method), 15`
`states_from_file() (westpa.cli.core.w_init.TargetState class method), 15`
`states_from_file() (westpa.cli.core.w_states.BasisState class method), 86`
`states_from_file() (westpa.cli.core.w_states.TargetState class method), 86`
`states_from_file() (westpa.core.data_manager.BasisState class method), 190`
`states_from_file() (westpa.core.data_manager.TargetState class method), 191`
`states_from_file() (westpa.core.propagators.executable.BasisState class method), 182`
`states_from_file() (westpa.core.states.BasisState class method), 231`
`states_from_file() (westpa.core.states.TargetState class method), 232`
`states_to_file() (westpa.cli.core.w_init.BasisState class method), 15`
`states_to_file() (westpa.cli.core.w_init.TargetState class method), 15`
`states_to_file() (westpa.cli.core.w_states.BasisState class method), 86`
`states_to_file() (westpa.cli.core.w_states.TargetState class method), 86`
`states_to_file() (westpa.core.data_manager.BasisState class method), 190`
`states_to_file() (westpa.core.data_manager.TargetState class method), 191`
`states_to_file() (westpa.core.propagators.executable.BasisState class method), 182`
`states_to_file() (westpa.core.states.BasisState class method), 231`
`states_to_file() (westpa.core.states.TargetState class method), 232`
`stats_process() (in module westpa.core.reweight),`

187

`stats_process()` (in `westpa.core.reweight.matrix`), 187

`status_names` (`westpa.cli.core.w_fork.Segment` attribute), 24

`status_names` (`westpa.cli.core.w_states.Segment` attribute), 85

`status_names` (`westpa.cli.core.w_succ.Segment` attribute), 58

`status_names` (`westpa.cli.tools.w_dumpsegs.Segment` attribute), 141

`status_names` (`westpa.cli.tools.w_trace.Segment` attribute), 39

`status_names` (`westpa.core.binning.mab_manager.Segment` attribute), 173

`status_names` (`westpa.core.data_manager.Segment` attribute), 189

`status_names` (`westpa.core.propagators.executable.Segment` attribute), 184

`status_names` (`westpa.core.segment.Segment` attribute), 225

`status_names` (`westpa.core.sim_manager.Segment` attribute), 227

`status_names` (`westpa.core.states.Segment` attribute), 230

`status_names` (`westpa.core.we_driver.Segment` attribute), 235

`status_names` (`westpa.oldtools.iframe.data_reader.Segment` attribute), 309

`status_text` (`westpa.cli.core.w_fork.Segment` property), 25

`status_text` (`westpa.cli.core.w_states.Segment` property), 85

`status_text` (`westpa.cli.core.w_succ.Segment` property), 58

`status_text` (`westpa.cli.tools.w_dumpsegs.Segment` property), 142

`status_text` (`westpa.cli.tools.w_trace.Segment` property), 39

`status_text` (`westpa.core.binning.mab_manager.Segment` property), 174

`status_text` (`westpa.core.data_manager.Segment` property), 190

`status_text` (`westpa.core.propagators.executable.Segment` property), 184

`status_text` (`westpa.core.segment.Segment` property), 225

`status_text` (`westpa.core.sim_manager.Segment` property), 227

`status_text` (`westpa.core.states.Segment` property), 231

`status_text` (`westpa.core.we_driver.Segment` property), 235

`status_text` (`westpa.oldtools.iframe.data_reader.Segment` property), 309

`statuses` (`westpa.cli.core.w_fork.Segment` attribute), 24

`statuses` (`westpa.cli.core.w_states.Segment` attribute), 85

`statuses` (`westpa.cli.core.w_succ.Segment` attribute), 57

`statuses` (`westpa.cli.tools.w_dumpsegs.Segment` attribute), 141

`statuses` (`westpa.cli.tools.w_trace.Segment` attribute), 39

`statuses` (`westpa.core.binning.mab_manager.Segment` attribute), 173

`statuses` (`westpa.core.data_manager.Segment` attribute), 189

`statuses` (`westpa.core.propagators.executable.Segment` attribute), 184

`statuses` (`westpa.core.segment.Segment` attribute), 225

`statuses` (`westpa.core.sim_manager.Segment` attribute), 227

`statuses` (`westpa.core.states.Segment` attribute), 230

`statuses` (`westpa.core.we_driver.Segment` attribute), 235

`statuses` (`westpa.oldtools.iframe.data_reader.Segment` attribute), 309

`std()` (`westpa.oldtools.stats.accumulator.RunningStatsAccumulator` method), 318

`std()` (`westpa.oldtools.stats.edfs.EDF` method), 318

`std()` (`westpa.oldtools.stats.RunningStatsAccumulator` method), 317

`stop()` (`westpa.core.progress.ProgressIndicator` method), 224

`stop()` (`westpa.tools.progress.ProgressIndicator` method), 295

`stop_iter` (`westpa.tools.selected_segs.SegmentSelection` property), 296

`StreamingStats1D` (class in `westpa.core.kinetics.rate_averaging`), 177

`StreamingStats2D` (class in `westpa.core.kinetics.rate_averaging`), 178

`StreamingStatsTuple` (class in `westpa.core.kinetics.rate_averaging`), 178

`subcommand` (`westpa.cli.tools.ploterr.DirectKinetics` attribute), 118

`subcommand` (`westpa.cli.tools.ploterr.DirectStateprobs` attribute), 119

`subcommand` (`westpa.cli.tools.ploterr.GenericIntervalSubcommand` attribute), 118

`subcommand` (`westpa.cli.tools.ploterr.ReweightKinetics` attribute), 119

`subcommand` (`westpa.cli.tools.ploterr.ReweightStateprobs` attribute), 119

`subcommand` (`westpa.cli.tools.ploterr.WESTSubcommand` attribute), 117

`subcommand` (`westpa.cli.tools.plothist.AveragePlotHist` attribute), 114

- subcommand (*westpa.cli.tools.plothist.EvolutionPlotHist* attribute), 115
- subcommand (*westpa.cli.tools.plothist.InstantPlotHist* attribute), 114
- subcommand (*westpa.cli.tools.plothist.WESTSubcommand* attribute), 113
- subcommand (*westpa.cli.tools.w_direct.DAll* attribute), 74
- subcommand (*westpa.cli.tools.w_direct.DAverage* attribute), 74
- subcommand (*westpa.cli.tools.w_direct.DKinAvg* attribute), 71
- subcommand (*westpa.cli.tools.w_direct.DKinetics* attribute), 70
- subcommand (*westpa.cli.tools.w_direct.DStateProbs* attribute), 73
- subcommand (*westpa.cli.tools.w_kinavg.DKinAvg* attribute), 124
- subcommand (*westpa.cli.tools.w_kinavg.WKinAvg* attribute), 126
- subcommand (*westpa.cli.tools.w_kinetics.DKinetics* attribute), 130
- subcommand (*westpa.cli.tools.w_kinetics.WKinetics* attribute), 131
- subcommand (*westpa.cli.tools.w_postanalysis_matrix.PAMass* attribute), 144
- subcommand (*westpa.cli.tools.w_postanalysis_matrix.RWMass* attribute), 143
- subcommand (*westpa.cli.tools.w_postanalysis_reweight.PAAverage* attribute), 147
- subcommand (*westpa.cli.tools.w_postanalysis_reweight.RWAAverage* attribute), 146
- subcommand (*westpa.cli.tools.w_reweight.RWAll* attribute), 155
- subcommand (*westpa.cli.tools.w_reweight.RWAverage* attribute), 155
- subcommand (*westpa.cli.tools.w_reweight.RWMatrix* attribute), 149
- subcommand (*westpa.cli.tools.w_reweight.RWRate* attribute), 151
- subcommand (*westpa.cli.tools.w_reweight.RWStateProbs* attribute), 153
- subcommand (*westpa.cli.tools.w_stateprobs.DStateProbs* attribute), 137
- subcommand (*westpa.cli.tools.w_stateprobs.WStateProbs* attribute), 138
- subcommand (*westpa.tools.core.WESTSubcommand* attribute), 287
- subcommand (*westpa.tools.kinetics_tool.WESTSubcommand* attribute), 293
- subcommand (*westpa.tools.WESTSubcommand* attribute), 277
- subcommands (*westpa.cli.tools.ploterr.PloterrsTool* attribute), 120
- subcommands (*westpa.cli.tools.ploterr.WESTMasterCommand* attribute), 116
- subcommands (*westpa.cli.tools.plothist.PlotHistTool* attribute), 115
- subcommands (*westpa.cli.tools.plothist.WESTMasterCommand* attribute), 112
- subcommands (*westpa.cli.tools.w_direct.WDirect* attribute), 74
- subcommands (*westpa.cli.tools.w_direct.WESTMasterCommand* attribute), 68
- subcommands (*westpa.cli.tools.w_kinavg.WDirect* attribute), 126
- subcommands (*westpa.cli.tools.w_kinavg.WESTMasterCommand* attribute), 123
- subcommands (*westpa.cli.tools.w_kinetics.WDirect* attribute), 131
- subcommands (*westpa.cli.tools.w_kinetics.WESTMasterCommand* attribute), 129
- subcommands (*westpa.cli.tools.w_postanalysis_matrix.WESTMasterCommand* attribute), 142
- subcommands (*westpa.cli.tools.w_postanalysis_matrix.WRweight* attribute), 144
- subcommands (*westpa.cli.tools.w_postanalysis_reweight.WESTMasterCommand* attribute), 145
- subcommands (*westpa.cli.tools.w_postanalysis_reweight.WRweight* attribute), 147
- subcommands (*westpa.cli.tools.w_reweight.WESTMasterCommand* attribute), 147
- subcommands (*westpa.cli.tools.w_reweight.WRweight* attribute), 155
- subcommands (*westpa.cli.tools.w_stateprobs.WDirect* attribute), 139
- subcommands (*westpa.cli.tools.w_stateprobs.WESTMasterCommand* attribute), 136
- subcommands (*westpa.tools.core.WESTMasterCommand* attribute), 288
- subcommands (*westpa.tools.WESTMasterCommand* attribute), 278
- submit() (*westpa.work_managers.core.WorkManager* method), 243
- submit() (*westpa.work_managers.mpi.Manager* method), 249
- submit() (*westpa.work_managers.mpi.MPIWorkManager* method), 248
- submit() (*westpa.work_managers.mpi.Serial* method), 248
- submit() (*westpa.work_managers.mpi.WorkManager* method), 247
- submit() (*westpa.work_managers.processes.ProcessWorkManager* method), 251
- submit() (*westpa.work_managers.processes.WorkManager* method), 250
- submit() (*westpa.work_managers.ProcessWorkManager* method), 242

`submit()` (`westpa.work_managers.serial.SerialWorkManager` method), 253

`submit()` (`westpa.work_managers.serial.WorkManager` method), 252

`submit()` (`westpa.work_managers.SerialWorkManager` method), 241

`submit()` (`westpa.work_managers.threads.ThreadsWorkManager` method), 255

`submit()` (`westpa.work_managers.threads.WorkManager` method), 254

`submit()` (`westpa.work_managers.ThreadsWorkManager` method), 241

`submit()` (`westpa.work_managers.zeromq.work_manager.WorkManager` method), 269

`submit()` (`westpa.work_managers.zeromq.work_manager.ZMQWorkManager` method), 272

`submit()` (`westpa.work_managers.zeromq.ZMQWorkManager` method), 258

`submit_as_completed()` (`westpa.work_managers.core.WorkManager` method), 243

`submit_as_completed()` (`westpa.work_managers.mpi.WorkManager` method), 247

`submit_as_completed()` (`westpa.work_managers.processes.WorkManager` method), 250

`submit_as_completed()` (`westpa.work_managers.serial.WorkManager` method), 252

`submit_as_completed()` (`westpa.work_managers.threads.WorkManager` method), 254

`submit_as_completed()` (`westpa.work_managers.zeromq.work_manager.WorkManager` method), 270

`submit_many()` (`westpa.work_managers.core.WorkManager` method), 243

`submit_many()` (`westpa.work_managers.mpi.WorkManager` method), 247

`submit_many()` (`westpa.work_managers.processes.WorkManager` method), 250

`submit_many()` (`westpa.work_managers.serial.WorkManager` method), 252

`submit_many()` (`westpa.work_managers.threads.WorkManager` method), 254

`submit_many()` (`westpa.work_managers.zeromq.work_manager.WorkManager` method), 269

`submit_many()` (`westpa.work_managers.zeromq.work_manager.ZMQWorkManager` method), 272

`submit_many()` (`westpa.work_managers.zeromq.ZMQWorkManager` method), 258

`subparsers_title` (`westpa.cli.tools.ploterr.PloterrsTool` attribute), 120

`subparsers_title` (`westpa.cli.tools.ploterr.WESTMasterCommand` attribute), 116

`subparsers_title` (`westpa.cli.tools.plothist.PlotHistTool` attribute), 115

`subparsers_title` (`westpa.cli.tools.plothist.WESTMasterCommand` attribute), 112

`subparsers_title` (`westpa.cli.tools.w_direct.WDirect` attribute), 74

`subparsers_title` (`westpa.cli.tools.w_direct.WESTMasterCommand` attribute), 68

`subparsers_title` (`westpa.cli.tools.w_kinavg.WDirect` attribute), 126

`subparsers_title` (`westpa.cli.tools.w_kinavg.WESTMasterCommand` attribute), 123

`subparsers_title` (`westpa.cli.tools.w_kinetics.WDirect` attribute), 132

`subparsers_title` (`westpa.cli.tools.w_kinetics.WESTMasterCommand` attribute), 129

`subparsers_title` (`westpa.cli.tools.w_postanalysis_matrix.WESTMasterCommand` attribute), 142

`subparsers_title` (`westpa.cli.tools.w_postanalysis_matrix.WRweight` attribute), 144

`subparsers_title` (`westpa.cli.tools.w_postanalysis_reweight.WESTMasterCommand` attribute), 145

`subparsers_title` (`westpa.cli.tools.w_postanalysis_reweight.WRweight` attribute), 147

`subparsers_title` (`westpa.cli.tools.w_reweight.WESTMasterCommand` attribute), 147

`subparsers_title` (`westpa.cli.tools.w_reweight.WRweight` attribute), 155

`subparsers_title` (`westpa.cli.tools.w_stateprobs.WDirect` attribute), 139

`subparsers_title` (`westpa.cli.tools.w_stateprobs.WESTMasterCommand` attribute), 136

`subparsers_title` (`westpa.tools.core.WESTMasterCommand` attribute), 288

`subparsers_title` (`westpa.tools.WESTMasterCommand` attribute), 277

`sum_except_along()` (in `westpa.cli.tools.plothist` module), 113

`summary` (`westpa.analysis.core.Iteration` property), 333

`summary` (`westpa.analysis.core.Run` property), 332

`superpose()` (`westpa.core.h5io.Trajectory` method), 200

`system` (`westpa.core.data_manager.WESTDataManager` property), 195

`target_state()` (`westpa.analysis.core.Iteration` method), 336

`target_state_pcoords` (`westpa.analysis.core.Iteration` property), 335

target_state_summaries
 (*westpa.analysis.core.Iteration* property), 335
 target_states (*westpa.analysis.core.Iteration* property), 335
 TargetState (class in *westpa.cli.core.w_init*), 15
 TargetState (class in *westpa.cli.core.w_states*), 86
 TargetState (class in *westpa.core.data_manager*), 190
 TargetState (class in *westpa.core.states*), 232
 Task (class in *westpa.work_managers.mpi*), 248
 Task (class in *westpa.work_managers.threads*), 255
 Task (class in *westpa.work_managers.zeromq.core*), 260
 Task (class in *westpa.work_managers.zeromq.work_manager*), 267
 Task (class in *westpa.work_managers.zeromq.worker*), 275
 TASK (*westpa.work_managers.zeromq.core.Message* attribute), 260
 TASK (*westpa.work_managers.zeromq.node.Message* attribute), 264
 TASK (*westpa.work_managers.zeromq.work_manager.Message* attribute), 267
 TASK (*westpa.work_managers.zeromq.worker.Message* attribute), 274
 task_generator() (*westpa.core.kinetics.rate_averaging.RateAverager* method), 179
 task_generator() (*westpa.core.kinetics.RateAverager* method), 174
 task_generator() (*westpa.westext.weed.weed_driver.RateAverager* method), 327
 task_generator() (*westpa.westext.wess.wess_driver.RateAverager* method), 328
 task_loop() (*westpa.work_managers.processes.ProcessWorkManager* method), 251
 task_loop() (*westpa.work_managers.ProcessWorkManager* method), 242
 TASK_REQUEST (*westpa.work_managers.zeromq.core.Message* attribute), 260
 TASK_REQUEST (*westpa.work_managers.zeromq.node.Message* attribute), 264
 TASK_REQUEST (*westpa.work_managers.zeromq.work_manager.Message* attribute), 267
 TASK_REQUEST (*westpa.work_managers.zeromq.worker.Message* attribute), 274
 TASKS_AVAILABLE (*westpa.work_managers.zeromq.core.Message* attribute), 260
 TASKS_AVAILABLE (*westpa.work_managers.zeromq.node.Message* attribute), 264
 TASKS_AVAILABLE (*westpa.work_managers.zeromq.work_manager.Message* attribute), 267
 TASKS_AVAILABLE (*westpa.work_managers.zeromq.worker.Message* attribute), 274
 tau (*westpa.cli.tools.w_red.RateCalculator* property), 106
 tdat_buffersize (*westpa.oldtools.iframe.TransitionEventAccumulator* attribute), 306
 tdat_buffersize (*westpa.oldtools.iframe.transitions.TransitionEventAccumulator* attribute), 316
 tell() (*westpa.core.h5io.HDF5TrajectoryFile* method), 214
 tell() (*westpa.core.propagators.executable.BytesIO* method), 181
 temperature (*westpa.core.h5io.Frames* attribute), 215
 template_args_for_segment() (*westpa.core.data_manager.ExecutablePropagator* method), 193
 template_args_for_segment() (*westpa.core.propagators.executable.ExecutablePropagator* method), 186
 text_to_h5dataset() (*westpa.oldtools.iframe.data_reader.ExtDataReaderMixin* method), 311
 text_to_h5dataset() (*westpa.oldtools.iframe.ExtDataReaderMixin* method), 303
 ThreadProxy (class in *westpa.work_managers.zeromq.node*), 265
 ThreadsWorkManager (class in *westpa.work_managers*), 241
 ThreadsWorkManager (class in *westpa.work_managers.threads*), 255
 time (*westpa.core.h5io.Frames* attribute), 215
 time (*westpa.core.h5io.Trajectory* attribute), 199
 time (*westpa.core.h5io.Trajectory* property), 203
 time_average() (in module *westpa.analysis.statistics*), 340
 time_delta (class in *westpa.core.sim_manager*), 225
 timestep (*westpa.core.h5io.Trajectory* attribute), 199
 timestep (*westpa.core.h5io.Trajectory* property), 200
 title (*westpa.core.h5io.HDF5TrajectoryFile* attribute), 211
 title (*westpa.core.h5io.HDF5TrajectoryFile* property), 212
 top (*westpa.core.h5io.Trajectory* attribute), 199
 top (*westpa.core.h5io.Trajectory* property), 200
 topology (*westpa.core.h5io.HDF5TrajectoryFile* attribute), 211
 topology (*westpa.core.h5io.HDF5TrajectoryFile* property), 212
 topology (*westpa.core.h5io.Trajectory* attribute), 199
 topology (*westpa.core.h5io.Trajectory* property), 202
 tostr() (in module *westpa.core.h5io*), 216
 total_number_of_walkers() (*westpa.cli.tools.w_multi_west.WMultiWest* method), 103
 total_seconds() (*westpa.core.sim_manager.timedelta* method), 226
 total_segs_in_range()

[westpa.cli.core.w_succ.WESTDataReaderMixin method](#)), 59
[total_segs_in_range\(\)](#) ([westpa.oldtools.afame.data_reader.WESTDataReaderMixin method](#)), 310
[total_segs_in_range\(\)](#) ([westpa.oldtools.afame.WESTDataReaderMixin method](#)), 303
[Trace](#) (class in [westpa.analysis.core](#)), 338
[Trace](#) (class in [westpa.cli.tools.w_trace](#)), 40
[trace\(\)](#) ([westpa.analysis.core.Walker method](#)), 337
[trace\(\)](#) ([westpa.cli.tools.w_ipa.WIPI method](#)), 49
[trace_timepoint_dataset\(\)](#) ([westpa.cli.tools.w_trace.Trace method](#)), 41
[trace_to_root\(\)](#) ([westpa.oldtools.afame.TrajWalker method](#)), 305
[trace_to_root\(\)](#) ([westpa.oldtools.afame.trajwalker.TrajWalker method](#)), 315
[trace_to_root\(\)](#) ([westpa.oldtools.afame.transitions.TrajWalker method](#)), 315
[trace_trajectories\(\)](#) ([westpa.oldtools.afame.TrajWalker method](#)), 305
[trace_trajectories\(\)](#) ([westpa.oldtools.afame.trajwalker.TrajWalker method](#)), 315
[trace_trajectories\(\)](#) ([westpa.oldtools.afame.transitions.TrajWalker method](#)), 316
[trace_trajectories\(\)](#) ([westpa.trajtrees.trajtrees.TrajTreeSet method](#)), 300
[trace_trajectories\(\)](#) ([westpa.trajtrees.trajtrees.TrajTreeSet method](#)), 299
[traceback](#) ([westpa.work_managers.core.WMFuture property](#)), 244
[traceback](#) ([westpa.work_managers.mpi.WMFuture property](#)), 248
[traceback](#) ([westpa.work_managers.processes.WMFuture property](#)), 251
[traceback](#) ([westpa.work_managers.serial.WMFuture property](#)), 253
[traceback](#) ([westpa.work_managers.threads.WMFuture property](#)), 255
[traceback](#) ([westpa.work_managers.zeromq.work_manager.WMFuture property](#)), 270
[traj_index_dtype](#) ([westpa.oldtools.afame.BFDDataManager attribute](#)), 304
[traj_index_dtype](#) ([westpa.oldtools.afame.data_reader.BFDDataManager attribute](#)), 311
[Trajectory](#) (class in [westpa.analysis.trajectories](#)), 338
[Trajectory](#) (class in [westpa.core.h5io](#)), 198
[trajectory_loader\(\)](#) (in module [westpa.core.propagators.executable](#)), 185
[trajnode](#) (class in [westpa.trajtrees.trajtrees](#)), 300
[TrajTreeSet](#) (class in [westpa.trajtrees.trajtrees](#)), 299
[TrajTreeSet](#) (class in [westpa.trajtrees.trajtrees](#)), 300
[TrajWalker](#) (class in [westpa.oldtools.afame](#)), 305
[TrajWalker](#) (class in [westpa.oldtools.afame.trajwalker](#)), 315
[TrajWalker](#) (class in [westpa.oldtools.afame.transitions](#)), 315
[TransitionAnalysisMixin](#) (class in [westpa.oldtools.afame](#)), 305
[TransitionAnalysisMixin](#) (class in [westpa.oldtools.afame.transitions](#)), 316
[TransitionEventAccumulator](#) (class in [westpa.oldtools.afame](#)), 306
[TransitionEventAccumulator](#) (class in [westpa.oldtools.afame.transitions](#)), 316
[UncertContainer](#) (class in [westpa.core.propagators.executable.BytesIO](#) method), 181
[tuple2stats\(\)](#) (in module [westpa.core.kinetics.rate_averaging](#)), 178

U

[UncertContainer](#) (class in [westpa.core.kinetics.rate_averaging](#)), 178
[UncertContainer](#) (class in [westpa.westext.weed.UncertMath.UncertContainer](#) method), 326
[union\(\)](#) ([westpa.analysis.core.BinUnion method](#)), 337
[unitcell_angles](#) ([westpa.core.h5io.Trajectory attribute](#)), 199
[unitcell_angles](#) ([westpa.core.h5io.Trajectory property](#)), 202
[unitcell_lengths](#) ([westpa.core.h5io.Trajectory attribute](#)), 199
[unitcell_lengths](#) ([westpa.core.h5io.Trajectory property](#)), 202
[unitcell_vectors](#) ([westpa.core.h5io.Trajectory attribute](#)), 199
[unitcell_vectors](#) ([westpa.core.h5io.Trajectory property](#)), 200
[unitcell_volumes](#) ([westpa.core.h5io.Trajectory property](#)), 200
[update\(\)](#) ([westpa.core.kinetics.rate_averaging.StreamingStats1D method](#)), 178
[update\(\)](#) ([westpa.core.kinetics.rate_averaging.StreamingStats2D method](#)), 178
[update_args_env_basis_state\(\)](#) ([westpa.core.data_manager.ExecutablePropagator method](#)), 193
[update_args_env_basis_state\(\)](#) ([westpa.core.propagators.executable.ExecutablePropagator method](#)), 186
[update_args_env_initial_state\(\)](#) ([westpa.core.data_manager.ExecutablePropagator method](#)), 193

`method`), 193
`update_args_env_initial_state()`
 (`westpa.core.propagators.executable.ExecutablePropagator`
 `method`), 186
`update_args_env_iter()`
 (`westpa.core.data_manager.ExecutablePropagator`
 `method`), 193
`update_args_env_iter()`
 (`westpa.core.propagators.executable.ExecutablePropagator`
 `method`), 186
`update_args_env_segment()`
 (`westpa.core.data_manager.ExecutablePropagator`
 `method`), 193
`update_args_env_segment()`
 (`westpa.core.propagators.executable.ExecutablePropagator`
 `method`), 186
`update_basis_initial_states()`
 (`westpa.core.propagators.executable.WESTPropagator`
 `method`), 181
`update_basis_initial_states()`
 (`westpa.core.propagators.WESTPropagator`
 `method`), 179
`update_bin_mapper()`
 (`westpa.westext.adaptvoronoi.AdaptiveVoronoiDriver`
 `method`), 321
`update_bin_mapper()`
 (`westpa.westext.adaptvoronoi.adaptVor_driver.AdaptiveVoronoiDriver`
 `method`), 320
`update_centers()` (`westpa.westext.adaptvoronoi.AdaptiveVoronoiDriver`
 `method`), 321
`update_centers()` (`westpa.westext.adaptvoronoi.adaptVor_driver.AdaptiveVoronoiDriver`
 `method`), 320
`update_from_file()` (`westpa.core.yamlcfg.YAMLConfig`
 `method`), 239
`update_initial_states()`
 (`westpa.core.data_manager.WESTDataManager`
 `method`), 196
`update_iter_group_links()`
 (`westpa.core.data_manager.WESTDataManager`
 `method`), 196
`update_iter_h5file()`
 (`westpa.core.data_manager.WESTDataManager`
 `method`), 196
`update_iter_summary()`
 (`westpa.core.data_manager.WESTDataManager`
 `method`), 196
`update_mask()` (`westpa.westext.weed.UncertMath.UncertMath`
 `method`), 326
`update_master_info()`
 (`westpa.work_managers.zeromq.worker.ZMQWorker`
 `method`), 268
`update_master_info()`
 (`westpa.work_managers.zeromq.worker.ZMQWorker`
 `method`), 275
`update_master_info()`
 (`westpa.work_managers.zeromq.ZMQWorker`
 `method`), 258
`update_segments()` (`westpa.core.data_manager.WESTDataManager`
 `method`), 196
`update_traj_index()`
 (`westpa.oldtools.iframe.BFDDataManager`
 `method`), 304
`update_traj_index()`
 (`westpa.oldtools.iframe.data_reader.BFDDataManager`
 `method`), 311
`update_worker_information()`
 (`westpa.work_managers.zeromq.work_manager.ZMQWorkManager`
 `method`), 272
`update_worker_information()`
 (`westpa.work_managers.zeromq.ZMQWorkManager`
 `method`), 259
`usage` (`westpa.cli.tools.w_bins.WESTTool` attribute), 19
`usage` (`westpa.cli.tools.w_dumpsegs.WESTTool` attribute), 140
`usage` (`westpa.cli.tools.w_fluxanl.WESTTool` attribute), 160
`usage` (`westpa.cli.tools.w_multi_west.WESTTool` attribute), 102
`usage` (`westpa.cli.tools.w_ntop.WESTTool` attribute), 96
`usage` (`westpa.cli.tools.w_trace.WESTTool` attribute), 37
`usage` (`westpa.core.WESTTool` attribute), 286
`usage` (`westpa.tools.WESTTool` attribute), 276
`utime_dtype` (in module `westpa.cli.tools.w_trace`), 40
`utime_dtype` (in module `westpa.core.data_manager`), 104

V

`valid_work_managers`
 (`westpa.work_managers.environment.WMEnvironment`
 attribute), 245
`validate_message()` (`westpa.work_managers.zeromq.core.ZMQCore`
 `method`), 261
`validate_message()` (`westpa.work_managers.zeromq.node.ZMQCore`
 `method`), 263
`validate_message()` (`westpa.work_managers.zeromq.work_manager.ZMQCore`
 `method`), 266
`validate_message()` (`westpa.work_managers.zeromq.worker.ZMQCore`
 `method`), 273
`validate_message()` (`westpa.work_managers.zeromq.ZMQCore`
 `method`), 257
`variance` (`westpa.core.kinetics.rate_averaging.StreamingStats1D`
 attribute), 178
`var` (`westpa.core.kinetics.rate_averaging.StreamingStats2D`
 attribute), 178
`var()` (`westpa.oldtools.stats.edfs.EDF` method), 318
`VectorizingFuncBinMapper` (class in
 `westpa.core.binning`), 163

- VectorizingFuncBinMapper (class in *westpa.core.binning.assign*), 167
- velocities (*westpa.core.h5io.Frames* attribute), 215
- VoronoiBinMapper (class in *westpa.core.binning*), 163
- VoronoiBinMapper (class in *westpa.core.binning.assign*), 167
- VoronoiBinMapper (class in *westpa.westext.adaptvoronoi.adaptVor_driver*), 319
- ## W
- w_kinavg() (*westpa.cli.tools.w_direct.DKinAvg* method), 72
- w_kinavg() (*westpa.cli.tools.w_kinavg.DKinAvg* method), 125
- w_kinetics() (*westpa.cli.tools.w_direct.WKinetics* method), 69
- w_kinetics() (*westpa.core.kinetics.events.WKinetics* method), 175
- w_kinetics() (*westpa.core.kinetics.WKinetics* method), 175
- w_postanalysis_matrix() (*westpa.cli.tools.w_reweight.FluxMatrix* method), 149
- w_postanalysis_matrix() (*westpa.core.reweight.FluxMatrix* method), 187
- w_postanalysis_matrix() (*westpa.core.reweight.matrix.FluxMatrix* method), 187
- w_postanalysis_reweight() (*westpa.cli.tools.w_reweight.RWRate* method), 152
- w_postanalysis_stateprobs() (*westpa.cli.tools.w_reweight.RWStateProbs* method), 154
- w_stateprobs() (*westpa.cli.tools.w_direct.DStateProbs* method), 73
- w_stateprobs() (*westpa.cli.tools.w_stateprobs.DStateProbs* method), 138
- wait() (*westpa.work_managers.core.FutureWatcher* method), 244
- wait() (*westpa.work_managers.core.WMFuture* method), 244
- wait() (*westpa.work_managers.mpi.WMFuture* method), 248
- wait() (*westpa.work_managers.processes.WMFuture* method), 250
- wait() (*westpa.work_managers.serial.WMFuture* method), 253
- wait() (*westpa.work_managers.threads.WMFuture* method), 255
- wait() (*westpa.work_managers.zeromq.work_manager.WMFuture* method), 270
- wait_all() (*westpa.work_managers.core.WorkManager* method), 243
- wait_all() (*westpa.work_managers.mpi.WorkManager* method), 247
- wait_all() (*westpa.work_managers.processes.WorkManager* method), 250
- wait_all() (*westpa.work_managers.serial.WorkManager* method), 252
- wait_all() (*westpa.work_managers.threads.WorkManager* method), 254
- wait_all() (*westpa.work_managers.zeromq.work_manager.WorkManager* method), 270
- wait_any() (*westpa.work_managers.core.WorkManager* method), 243
- wait_any() (*westpa.work_managers.mpi.WorkManager* method), 247
- wait_any() (*westpa.work_managers.processes.WorkManager* method), 250
- wait_any() (*westpa.work_managers.serial.WorkManager* method), 252
- wait_any() (*westpa.work_managers.threads.WorkManager* method), 254
- wait_any() (*westpa.work_managers.zeromq.work_manager.WorkManager* method), 270
- Walker (class in *westpa.analysis.core*), 336
- walker() (*westpa.analysis.core.Iteration* method), 335
- walkers (*westpa.analysis.core.Iteration* property), 334
- walkers (*westpa.analysis.core.Run* property), 332
- warn() (in module *westpa.cli.tools.w_dumpsegs*), 140
- warn() (in module *westpa.cli.tools.w_fluxanl*), 159
- warn() (in module *westpa.cli.tools.w_kinavg*), 125
- warn() (in module *westpa.cli.tools.w_kinetics*), 130
- warn() (in module *westpa.cli.tools.w_postanalysis_matrix*), 143
- warn() (in module *westpa.cli.tools.w_postanalysis_reweight*), 146
- warn() (in module *westpa.cli.tools.w_stateprobs*), 136
- warn_dubious_config_entry() (in module *westpa.core.yamlcfg*), 239
- WAssign (class in *westpa.cli.tools.w_assign*), 33
- WBinTool (class in *westpa.cli.tools.w_bins*), 21
- WCrawl (class in *westpa.cli.tools.w_crawl*), 65
- WDirect (class in *westpa.cli.tools.w_direct*), 74
- WDirect (class in *westpa.cli.tools.w_kinavg*), 126
- WDirect (class in *westpa.cli.tools.w_kinetics*), 131
- WDirect (class in *westpa.cli.tools.w_stateprobs*), 139
- WDumpSegs (class in *westpa.cli.tools.w_dumpsegs*), 142
- WEDDist (class in *westpa.cli.tools.w_eddist*), 92
- WEDriver (class in *westpa.core.binning.mab_driver*), 168
- WEDriver (class in *westpa.core.we_driver*), 237
- WEEDDriver (class in *westpa.westext.weed*), 328
- WEEDDriver (class in *westpa.westext.weed.weed_driver*), 327

weight (*westpa.analysis.core.Walker* property), 336
 weight (*westpa.core.binning.assign.Bin* property), 165
 weight (*westpa.core.binning.Bin* property), 165
 weight (*westpa.core.binning.bins.Bin* property), 167
 weight_dsspec (*westpa.cli.tools.w_assign.WESTDataReader* property), 29
 weight_dsspec (*westpa.cli.tools.w_bins.WESTDataReader* property), 19
 weight_dsspec (*westpa.cli.tools.w_crawl.WESTDataReader* property), 63
 weight_dsspec (*westpa.cli.tools.w_dumpsegs.WESTDataReader* property), 141
 weight_dsspec (*westpa.cli.tools.w_fluxanl.WESTDataReader* property), 160
 weight_dsspec (*westpa.cli.tools.w_ipa.WESTDataReader* property), 46
 weight_dsspec (*westpa.cli.tools.w_ntop.WESTDataReader* property), 97
 weight_dsspec (*westpa.cli.tools.w_pdist.WESTDataReader* property), 52
 weight_dsspec (*westpa.cli.tools.w_select.WESTDataReader* property), 79
 weight_dsspec (*westpa.cli.tools.w_trace.WESTDataReader* property), 38
 weight_dsspec (*westpa.tools.data_reader.WESTDataReader* property), 289
 weight_dsspec (*westpa.tools.kinetics_tool.WESTDataReader* property), 292
 weight_dsspec (*westpa.tools.WESTDataReader* property), 279
 weight_dtype (in module *westpa.cli.tools.w_assign*), 28
 weight_dtype (in module *westpa.cli.tools.w_direct*), 68
 weight_dtype (in module *westpa.cli.tools.w_fluxanl*), 159
 weight_dtype (in module *westpa.cli.tools.w_ntop*), 98
 weight_dtype (in module *westpa.cli.tools.w_select*), 80
 weight_dtype (in module *westpa.cli.tools.w_trace*), 40
 weight_dtype (in module *westpa.core.data_manager*), 194
 weight_dtype (in module *westpa.core.kinetics.events*), 175
 weight_dtype (in module *westpa.core.kinetics.matrates*), 175
 weight_dtype (in module *westpa.core.reweight.matrix*), 187
 weight_dtype (in module *westpa.core.sim_manager*), 226
 weight_dtype (in module *westpa.tools.binning*), 283
 weight_dtype (in module *westpa.tools.dtypes*), 290
 weight_dtype (*westpa.oldtools.afame.TransitionEventAccumulator* attribute), 306
 weight_dtype (*westpa.oldtools.afame.transitions.TransitionEventAccumulator* attribute), 316
 weight_merge_cutoff (*westpa.core.binning.mab_driver.WEDriver* attribute), 168
 weight_merge_cutoff (*westpa.core.we_driver.WEDriver* attribute), 237
 weight_split_threshold (*westpa.core.binning.mab_driver.WEDriver* attribute), 168
 weight_split_threshold (*westpa.core.we_driver.WEDriver* attribute), 237
 weighted_average() (*westpa.westext.weed.UncertMath.UncertContainer* method), 326
 weights (*westpa.analysis.core.Iteration* property), 333
 WESimManager (class in *westpa.core.binning.mab_manager*), 170
 WESimManager (class in *westpa.core.sim_manager*), 228
 WESSDriver (class in *westpa.westext.wess*), 329
 WESSDriver (class in *westpa.westext.wess.wess_driver*), 328
 west (*westpa.cli.tools.w_ipa.WIPI* property), 48
 west (*westpa.cli.tools.w_ipa.WIPIScheme* property), 47
 west (*westpa.tools.wipi.WIPIScheme* property), 297
 west (*westpa.tools.WIPIScheme* property), 282
 WESTAnalysisTool (class in *westpa.cli.core.w_succ*), 58
 WESTAnalysisTool (class in *westpa.oldtools.afame*), 301
 WESTAnalysisTool (class in *westpa.oldtools.afame.atool*), 307
 WESTDataManager (class in *westpa.core.data_manager*), 194
 WESTDataReader (class in *westpa.cli.tools.w_assign*), 29
 WESTDataReader (class in *westpa.cli.tools.w_bins*), 19
 WESTDataReader (class in *westpa.cli.tools.w_crawl*), 63
 WESTDataReader (class in *westpa.cli.tools.w_dumpsegs*), 140
 WESTDataReader (class in *westpa.cli.tools.w_fluxanl*), 160
 WESTDataReader (class in *westpa.cli.tools.w_ipa*), 46
 WESTDataReader (class in *westpa.cli.tools.w_ntop*), 96
 WESTDataReader (class in *westpa.cli.tools.w_pdist*), 52
 WESTDataReader (class in *westpa.cli.tools.w_select*), 79
 WESTDataReader (class in *westpa.cli.tools.w_trace*), 38
 WESTDataReader (class in *westpa.tools*), 278
 WESTDataReader (class in *westpa.tools.data_reader*), 289
 WESTDataReader (class in *westpa.tools.kinetics_tool*), 292
 WESTDataReaderMixin (class in *westpa.cli.core.w_succ*), 58
 WESTDataReaderMixin (class in *westpa.oldtools.afame*), 302
 WESTDataReaderMixin (class in *westpa.oldtools.afame.data_reader*), 310

WESTDSSynthesizer	(class in westpa.cli.tools.w_assign), 29	in westpa.cli.core.w_states module, 84
WESTDSSynthesizer	(class in westpa.cli.tools.w_pdist), 52	westpa.cli.core.w_succ module, 57
WESTDSSynthesizer	(class in westpa.tools), 279	westpa.cli.core.w_truncate module, 23
WESTDSSynthesizer	(class in westpa.tools.data_reader), 289	westpa.cli.tools.ploterr module, 116
WESTIterationFile	(class in westpa.core.h5io), 219	westpa.cli.tools.plothist module, 111
WESTKineticsBase	(class in westpa.cli.tools.w_direct), 69	westpa.cli.tools.w_assign module, 28
WESTKineticsBase	(class in westpa.cli.tools.w_reweight), 148	westpa.cli.tools.w_bins module, 19
WESTKineticsBase	(class in westpa.tools.kinetics_tool), 293	westpa.cli.tools.w_crawl module, 63
WESTMasterCommand	(class in westpa.cli.tools.ploterr), 116	westpa.cli.tools.w_direct module, 68
WESTMasterCommand	(class in westpa.cli.tools.plothist), 112	westpa.cli.tools.w_dumpsegs module, 140
WESTMasterCommand	(class in westpa.cli.tools.w_direct), 68	westpa.cli.tools.w_eddist module, 91
WESTMasterCommand	(class in westpa.cli.tools.w_kinavg), 123	westpa.cli.tools.w_fluxanl module, 157
WESTMasterCommand	(class in westpa.cli.tools.w_kinetics), 129	westpa.cli.tools.w_ipa module, 45
WESTMasterCommand	(class in westpa.cli.tools.w_postanalysis_matrix), 142	westpa.cli.tools.w_kinavg module, 123
WESTMasterCommand	(class in westpa.cli.tools.w_postanalysis_reweight), 145	westpa.cli.tools.w_kinetics module, 129
WESTMasterCommand	(class in westpa.cli.tools.w_reweight), 147	westpa.cli.tools.w_multi_west module, 102
WESTMasterCommand	(class in westpa.cli.tools.w_stateprobs), 136	westpa.cli.tools.w_ntop module, 96
WESTMasterCommand	(class in westpa.tools), 277	westpa.cli.tools.w_pdist module, 52
WESTMasterCommand	(class in westpa.tools.core), 287	westpa.cli.tools.w_postanalysis_matrix module, 142
WESTMultiTool	(class in westpa.cli.tools.w_multi_west), 102	westpa.cli.tools.w_postanalysis_reweight module, 145
WESTMultiTool	(class in westpa.tools), 278	westpa.cli.tools.w_red module, 105
WESTMultiTool	(class in westpa.tools.core), 287	westpa.cli.tools.w_reweight module, 147
WESTMultiTool.NoSimulationsException,	103, 278, 287	westpa.cli.tools.w_select module, 78
westpa.analysis.core	module, 332	westpa.cli.tools.w_stateprobs module, 136
westpa.analysis.statistics	module, 340	westpa.cli.tools.w_trace module, 37
westpa.analysis.trajectories	module, 338	westpa.core module, 188
westpa.cli.core.w_fork	module, 24	westpa.core.binning module, 162
westpa.cli.core.w_init	module, 15	
westpa.cli.core.w_run	module, 22	

westpa.core.binning.assign	westpa.oldtools
module, 165, 349	module, 300
westpa.core.binning.bins	westpa.oldtools.iframe
module, 167, 349	module, 301
westpa.core.binning.mab	westpa.oldtools.iframe.atool
module, 168	module, 307
westpa.core.binning.mab_driver	westpa.oldtools.iframe.base_mixin
module, 168	module, 307
westpa.core.binning.mab_manager	westpa.oldtools.iframe.binning
module, 170	module, 308
westpa.core.data_manager	westpa.oldtools.iframe.data_reader
module, 188	module, 308
westpa.core.extloader	westpa.oldtools.iframe.iter_range
module, 198	module, 312
westpa.core.h5io	westpa.oldtools.iframe.kinetics
module, 198	module, 313
westpa.core.kinetics	westpa.oldtools.iframe.mcbs
module, 174	module, 313
westpa.core.kinetics.events	westpa.oldtools.iframe.output
module, 175	module, 314
westpa.core.kinetics.matrates	westpa.oldtools.iframe.plotting
module, 175	module, 315
westpa.core.kinetics.rate_averaging	westpa.oldtools.iframe.trajwalker
module, 177	module, 315
westpa.core.progress	westpa.oldtools.iframe.transitions
module, 222	module, 315
westpa.core.propagators	westpa.oldtools.cmds
module, 179	module, 317
westpa.core.propagators.executable	westpa.oldtools.files
module, 180	module, 300
westpa.core.reweight	westpa.oldtools.miscfn
module, 187	module, 301
westpa.core.reweight.matrix	westpa.oldtools.stats
module, 187	module, 317
westpa.core.segment	westpa.oldtools.stats.accumulator
module, 224	module, 317
westpa.core.sim_manager	westpa.oldtools.stats.edfs
module, 225	module, 318
westpa.core.states	westpa.oldtools.stats.mcbs
module, 230	module, 319
westpa.core.systems	westpa.tools
module, 233	module, 276
westpa.core.textio	westpa.tools.binning
module, 234	module, 283
westpa.core.we_driver	westpa.tools.core
module, 234	module, 285
westpa.core.wm_ops	westpa.tools.data_reader
module, 238	module, 288
westpa.core.yamlcfg	westpa.tools.dtypes
module, 239, 349	module, 290
westpa.fasthist	westpa.tools.iter_range
module, 298	module, 290
westpa.mclib	westpa.tools.kinetics_tool
module, 298	module, 292

`westpa.tools.plot`
 module, 294

`westpa.tools.progress`
 module, 294

`westpa.tools.selected_segs`
 module, 295

`westpa.tools.wipi`
 module, 297

`westpa.trajtree`
 module, 299

`westpa.trajtree.trajtree`
 module, 300

`westpa.westext`
 module, 329

`westpa.westext.adaptvoronoi`
 module, 320

`westpa.westext.adaptvoronoi.adaptVor_driver`
 module, 319

`westpa.westext.weed`
 module, 327

`westpa.westext.weed.BinCluster`
 module, 326

`westpa.westext.weed.ProbAdjustEquil`
 module, 326

`westpa.westext.weed.UncertMath`
 module, 326

`westpa.westext.weed.weed_driver`
 module, 327

`westpa.westext.wess`
 module, 329

`westpa.westext.wess.ProbAdjust`
 module, 328

`westpa.westext.wess.wess_driver`
 module, 328

`westpa.work_managers`
 module, 241

`westpa.work_managers.core`
 module, 242

`westpa.work_managers.environment`
 module, 245

`westpa.work_managers.mpi`
 module, 246

`westpa.work_managers.processes`
 module, 249

`westpa.work_managers.serial`
 module, 252

`westpa.work_managers.threads`
 module, 254

`westpa.work_managers.zeromq`
 module, 256

`westpa.work_managers.zeromq.core`
 module, 259

`westpa.work_managers.zeromq.node`
 module, 263

`westpa.work_managers.zeromq.work_manager`
 module, 265

`westpa.work_managers.zeromq.worker`
 module, 272

`WESTPACrawler` (class in `westpa.cli.tools.w_crawl`), 65

`WESTPAH5File` (class in `westpa.cli.tools.w_assign`), 31

`WESTPAH5File` (class in `westpa.core.h5io`), 217

`WESTParallelTool` (class in `westpa.cli.tools.w_assign`), 29

`WESTParallelTool` (class in `westpa.cli.tools.w_crawl`), 63

`WESTParallelTool` (class in `westpa.cli.tools.w_direct`), 69

`WESTParallelTool` (class in `westpa.cli.tools.w_eddist`), 91

`WESTParallelTool` (class in `westpa.cli.tools.w_ipa`), 45

`WESTParallelTool` (class in `westpa.cli.tools.w_kinavg`), 124

`WESTParallelTool` (class in `westpa.cli.tools.w_kinetics`), 129

`WESTParallelTool` (class in `westpa.cli.tools.w_pdist`), 52

`WESTParallelTool` (class in `westpa.cli.tools.w_postanalysis_matrix`), 143

`WESTParallelTool` (class in `westpa.cli.tools.w_postanalysis_reweight`), 145

`WESTParallelTool` (class in `westpa.cli.tools.w_red`), 105

`WESTParallelTool` (class in `westpa.cli.tools.w_reweight`), 148

`WESTParallelTool` (class in `westpa.cli.tools.w_select`), 78

`WESTParallelTool` (class in `westpa.cli.tools.w_stateprobs`), 136

`WESTParallelTool` (class in `westpa.tools`), 276

`WESTParallelTool` (class in `westpa.tools.core`), 286

`WESTPropagator` (class in `westpa.core.propagators`), 179

`WESTPropagator` (class in `westpa.core.propagators.executable`), 181

`WESTRC` (class in `westpa.core._rc`), 349

`WESTSubcommand` (class in `westpa.cli.tools.ploterr`), 117

`WESTSubcommand` (class in `westpa.cli.tools.plothist`), 113

`WESTSubcommand` (class in `westpa.tools`), 277

`WESTSubcommand` (class in `westpa.tools.core`), 287

`WESTSubcommand` (class in `westpa.tools.kinetics_tool`), 293

`WESTSystem` (class in `westpa.core.systems`), 233

`WESTTool` (class in `westpa.cli.tools.w_bins`), 19

`WESTTool` (class in `westpa.cli.tools.w_dumpsegs`), 140

`WESTTool` (class in `westpa.cli.tools.w_fluxanl`), 159

`WESTTool` (class in `westpa.cli.tools.w_multi_west`), 102

- WESTTool (class in westpa.cli.tools.w_ntop), 96
- WESTTool (class in westpa.cli.tools.w_trace), 37
- WESTTool (class in westpa.tools), 276
- WESTTool (class in westpa.tools.core), 286
- WESTToolComponent (class in westpa.tools), 277
- WESTToolComponent (class in westpa.tools.binning), 283
- WESTToolComponent (class in westpa.tools.core), 285
- WESTToolComponent (class in westpa.tools.data_reader), 288
- WESTToolComponent (class in westpa.tools.iter_range), 290
- WESTToolComponent (class in westpa.tools.progress), 295
- WESTToolComponent (class in westpa.tools.selected_segs), 295
- WESTTrajectory (class in westpa.core.h5io), 215
- WESTWSSynthesizer (class in westpa.cli.tools.w_pdist), 53
- WESTWSSynthesizer (class in westpa.tools), 279
- WESTWSSynthesizer (class in westpa.tools.data_reader), 289
- WFluxanlTool (class in westpa.cli.tools.w_fluxanl), 161
- which_expired() (westpa.work_managers.zeromq.core.PassiveMultiTimer method), 261
- which_expired() (westpa.work_managers.zeromq.node.PollMultiTimer method), 265
- which_expired() (westpa.work_managers.zeromq.work_manager.PollMultiTimer method), 268
- which_expired() (westpa.work_managers.zeromq.worker.PollMultiTimer method), 275
- WIPI (class in westpa.cli.tools.w_ipa), 47
- WIPI.Future (class in westpa.cli.tools.w_ipa), 49
- WIPIDataset (class in westpa.cli.tools.w_ipa), 46
- WIPIDataset (class in westpa.tools), 282
- WIPIDataset (class in westpa.tools.wipi), 297
- WIPIScheme (class in westpa.cli.tools.w_ipa), 46
- WIPIScheme (class in westpa.tools), 282
- WIPIScheme (class in westpa.tools.wipi), 297
- WKinAvg (class in westpa.cli.tools.w_kinavg), 126
- WKinetics (class in westpa.cli.tools.w_direct), 69
- WKinetics (class in westpa.cli.tools.w_kinetics), 131
- WKinetics (class in westpa.core.kinetics), 175
- WKinetics (class in westpa.core.kinetics.events), 175
- WMEEnvironment (class in westpa.work_managers.environment), 245
- WMFuture (class in westpa.work_managers.core), 244
- WMFuture (class in westpa.work_managers.mpi), 247
- WMFuture (class in westpa.work_managers.processes), 250
- WMFuture (class in westpa.work_managers.serial), 252
- WMFuture (class in westpa.work_managers.threads), 254
- WMFuture (class in westpa.work_managers.zeromq.work_manager), 270
- WMultiWest (class in westpa.cli.tools.w_multi_west), 103
- WNTopTool (class in westpa.cli.tools.w_ntop), 98
- work_manager (westpa.cli.tools.ploterr.WESTSubcommand property), 117
- work_manager (westpa.cli.tools.plotthist.WESTSubcommand property), 113
- work_manager (westpa.tools.core.WESTSubcommand property), 287
- work_manager (westpa.tools.kinetics_tool.WESTSubcommand property), 293
- work_manager (westpa.tools.WESTSubcommand property), 277
- Worker (class in westpa.work_managers.mpi), 249
- WorkManager (class in westpa.work_managers.core), 243
- WorkManager (class in westpa.work_managers.mpi), 246
- WorkManager (class in westpa.work_managers.processes), 249
- WorkManager (class in westpa.work_managers.serial), 252
- WorkManager (class in westpa.work_managers.threads), 254
- WorkManager (class in westpa.work_managers.zeromq.work_manager), 269
- WRD (class in westpa.cli.tools.w_pdist), 54
- WRed (class in westpa.cli.tools.w_red), 106
- WReweight (class in westpa.cli.tools.w_postanalysis_matrix), 144
- WReweight (class in westpa.cli.tools.w_postanalysis_reweight), 147
- WReweight (class in westpa.cli.tools.w_reweight), 155
- writable() (westpa.core.propagators.executable.BytesIO method), 181
- write() (westpa.core.h5io.HDF5TrajectoryFile method), 213
- write() (westpa.core.propagators.executable.BytesIO method), 181
- write() (westpa.core.textio.NumericTextOutputFormatter method), 234
- write_bin_info() (in module westpa.cli.tools.w_bins), 20
- write_bin_info() (in module westpa.tools.binning), 284
- write_bin_labels() (in module westpa.tools.binning), 284
- write_bin_labels() (westpa.oldtools.iframe.binning.BinningMixin method), 308
- write_bin_labels() (westpa.oldtools.iframe.BinningMixin method), 304
- write_comment() (westpa.core.textio.NumericTextOutputFormatter method), 234
- write_data() (westpa.core.h5io.WESTIterationFile method), 220

`write_header()` (*westpa.core.textio.NumericTextOutputFormatter* (class in *westpa.core.textio*), 234)
`write_header()` (*westpa.work_managers.zeromq.core.ZMQWorker* (class in *westpa.work_managers.zeromq*), 258)
`write_host_info()` (*westpa.work_managers.zeromq.core.ZMQWorker* (class in *westpa.work_managers.zeromq*), 262)
`write_host_info()` (*westpa.work_managers.zeromq.node.ZMQWorker* (class in *westpa.work_managers.zeromq*), 265)
`write_host_info()` (*westpa.work_managers.zeromq.worker.ZMQWorker* (class in *westpa.work_managers.zeromq*), 275)
`write_host_info()` (*westpa.work_managers.zeromq.work_manager.ZMQWorkerMissing*, 256, 259, 267)
`write_segment()` (*westpa.core.h5io.WESTIterationFile* (class in *westpa.core.h5io*), 220)
`writelines()` (*westpa.core.propagators.executable.BytesIO* (class in *westpa.core.propagators.executable*), 181)
`writelines()` (*westpa.core.textio.NumericTextOutputFormatter* (class in *westpa.core.textio*), 234)
`WSelectTool` (class in *westpa.cli.tools.w_select*), 80
`WStateProbs` (class in *westpa.cli.tools.w_stateprobs*), 138
`WSucc` (class in *westpa.cli.core.w_succ*), 59
`WTraceTool` (class in *westpa.cli.tools.w_trace*), 41

X

`xyz` (*westpa.core.h5io.Trajectory* attribute), 199
`xyz` (*westpa.core.h5io.Trajectory* property), 202

Y

`YAMLConfig` (class in *westpa.core.yamlcfg*), 239
`YAMLSystem` (class in *westpa.core.yamlcfg*), 240
`YLoader` (in module *westpa.core.yamlcfg*), 239

Z

`zip_longest` (class in *westpa.core.kinetics.rate_averaging*), 177
`zip_longest` (class in *westpa.core.sim_manager*), 226
`ZMQCore` (class in *westpa.work_managers.zeromq*), 256
`ZMQCore` (class in *westpa.work_managers.zeromq.core*), 261
`ZMQCore` (class in *westpa.work_managers.zeromq.node*), 263
`ZMQCore` (class in *westpa.work_managers.zeromq.work_manager*), 265
`ZMQCore` (class in *westpa.work_managers.zeromq.worker*), 272
`ZMQExecutor` (class in *westpa.work_managers.zeromq.worker*), 275
`ZMQNode` (class in *westpa.work_managers.zeromq*), 257
`ZMQNode` (class in *westpa.work_managers.zeromq.node*), 265
`ZMQNode` (class in *westpa.work_managers.zeromq.work_manager*), 269
`ZMQWEnvironmentError`, 256, 259, 268
`ZMQWError`, 256, 259
`ZMQWTimeout`, 256, 259, 274