MDWF Database Management Tool
Complete Tutorial & Command Reference

LQCD Workflow Management

August 20, 2025

Contents

1 Installation & Setup

1.1 Installing the CLI Tool

Install the MDWF package to use the `mdwf_db` command directly:

```
$ pip install -e /path/to/mdwf_db
Successfully installed MDWFutils-0.1

# Now you can use the mdwf_db command directly
$ mdwf_db --help
```

Listing 1: Installing MDWF CLI

1.2 Perlmutter Environment Setup

On Perlmutter, you can load the MDWF environment using:

```
module load conda
conda activate /global/cfs/cdirs/m2986/cosmon/mdwf/scripts/cosmon_mdwf
```

This will make the `mdwf_db` command available in your shell.

2 Complete Command Reference

2.1 Database Management Commands

Command	**Alias**	**Purpose**	**Key Options**
`init-db`	`init`	Initialize database and directory structure	`--base-dir`, `--db-file`
`add-ensemble`	`add`	Add ensemble to database	`-p/--params`, `-s/--status`, `-d/--director`
`query`	`q`	List and inspect ensembles	`-e/--ensemble`, `--operations`, `--params`
`promote-ensemble`	`promote`	Move from TUNING to PRODUCTION	`-e/--ensemble`, `--force`
`remove-ensemble`	`remove`	Remove ensemble completely	`-e/--ensemble`, `--force`
`clear-history`	`clear`	Clear operation history	`-e/--ensemble`, `--force`

Table 1: Database Management Commands

2.2 Script Generation Commands

Command	**Alias**	**Purpose**	**Key Options**
`hmc-script`	`hmc`	Generate HMC XML and SLURM script	`-e`, `-a`, `-m`, `-x`, `-j`, `--use-default-params`
`hmc-xml`	`hmc-x`	Generate standalone HMC XML	`-e`, `-m`, `-x`, `-o`
`smear-script`	`smear`	Generate GLU smearing script	`-e`, `-j`, `-g`, `--use-default-params`
`wflow-script`	`wflow`	Generate gradient flow script	`-e`, `-j`, `-g`, `--use-default-params`
`meson2pt-script`	`meson`	Generate WIT meson script	`-e`, `-j`, `-w`, `--use-default-params`
`mres-script`	`mres`	Generate WIT MRES script	`-e`, `-j`, `-g`, `--use-default-params`
`glu-input`	`glu`	Generate GLU input file	`-e`, `-o`, `-g`, `-t`
`wit-input`	`wit`	Generate WIT input file	`-e`, `-o`, `-w`

Table 2: Script Generation Commands

Command	Alias	Purpose	Key Options
`default_params`	`defaults`	Manage default parameter files	`generate`, `show`, `edit`, `validate`

Table 3: Parameter Management Commands

Command	Alias	Purpose	Key Options
`update`	`u`	Record/update operation status	`-e`, `-o`, `-s`, `-p`, `-u`

Table 4: Operation Tracking Commands

2.3 Common Options Across Commands

The following options are used consistently across most MDWF commands:

- `-e, --ensemble`: **Ensemble identifier** - Can be an ensemble ID (integer), directory path, or "." for current directory. This is the most commonly used option.
- `-j, --job-params`: **SLURM/Job parameters** - Space-separated key=value pairs for job submission parameters like `time_limit`, `nodes`, `mail_user`, etc.
- `-x, --xml-params`: **HMC XML parameters** - Used in HMC commands for physics parameters like `Trajectories`, `MDsteps`, `trajL`, etc.
- `-w, --wit-params`: **WIT input parameters** - Used in WIT-based commands for measurement parameters like `Configurations.first`, `Propagator 0.Source`, etc.
- `-g, --glu-params`: **GLU input parameters** - Used in GLU-based commands for smearing and utility parameters like `SMITERS`, `ALPHA1`, etc.
- `--db-file`: Database file path (auto-discovered by default)
- `--use-default-params`: Load from ensemble default parameter file
- `--save-default-params`: Save current parameters to default file
- `--params-variant`: Use specific parameter variant
- `--save-params-as`: Save under custom variant name

2.4 init-db: Initialize Database

Purpose: Create a new MDWF database and directory structure.

This command initializes a SQLite database with the required schema and creates the TUNING/ and ENSEMBLES/ directory structure.

Options:

- `--db-file DB_FILE`: Path to SQLite database (optional, auto-discovered by default)
- `--base-dir BASE_DIR`: Root directory for TUNING/ and ENSEMBLES/ (default: current directory)

Examples:

```
# Initialize database in current directory
$ mkdir -p /scratch/lattice/my_project && cd /scratch/lattice/my_project
$ mdwf_db init-db
Ensured directory: /scratch/lattice/my_project
Ensured directory: /scratch/lattice/my_project/TUNING
Ensured directory: /scratch/lattice/my_project/ENSEMBLES
init_database returned: True

$ ls -la
-rw-r--r-- 1 user group 40960 mdwf_ensembles.db
```

```
drwxr-xr-x 2 user group    64 ENSEMBLES/
drwxr-xr-x 2 user group    64 TUNING/

# Initialize with custom base directory
$ mdwf_db init-db --base-dir /tmp/mdwf_expanded_test
Ensured directory: /private/tmp/mdwf_expanded_test
Ensured directory: /private/tmp/mdwf_expanded_test/TUNING
Ensured directory: /private/tmp/mdwf_expanded_test/ENSEMBLES
init_database returned: True
```

2.5 add-ensemble: Add New Ensemble

Purpose: Add a new ensemble to the database with physics parameters.

Creates the ensemble directory structure and adds the record to the database with all physics parameters.

Options:

- `-p, --params PARAMS`: **Required.** Space-separated key=val pairs for physics parameters
- `-s, --status {TUNING,PRODUCTION}`: **Required.** Ensemble status
- `-d, --directory DIRECTORY`: Explicit directory path (overrides auto-generated path)
- `-b, --base-dir BASE_DIR`: Root directory for TUNING/ENSEMBLES (default: current)
- `--description DESCRIPTION`: Optional description text

Required Physics Parameters: beta, b, Ls, mc, ms, ml, L, T

Directory Structure Created:

`<STATUS>/b<beta>/b<b>Ls<Ls>/mc<mc>/ms<ms>/ml<ml>/L<L>/T<T>/`

Each ensemble directory contains:

- `cnfg/`: Gauge configuration files
- `slurm/`: Generated SLURM scripts
- `jlog/`: Job logs and output
- `log_hmc/`: HMC-specific logs

Examples:

```
# Add TUNING ensemble with standard parameters
$ mdwf_db add-ensemble \
    -p "beta=6.0 b=1.8 Ls=24 mc=0.8555 ms=0.0725 ml=0.0195 L=32 T=64" \
    -s TUNING \
    --description "First test ensemble - 32^3x64"
Ensemble added: ID=1

# Add second ensemble with different physics parameters
$ mdwf_db add-ensemble \
    -p "beta=5.8 b=2.0 Ls=16 mc=0.9 ms=0.08 ml=0.02 L=24 T=48" \
    -s TUNING \
    --description "Second test ensemble - 24^3x48 with different parameters"
Ensemble added: ID=2

# Add ensemble directly in PRODUCTION status
$ mdwf_db add-ensemble \
    -p "beta=6.2 b=1.5 Ls=32 mc=0.8 ms=0.06 ml=0.015 L=48 T=96" \
    -s PRODUCTION \
    --description "Large production ensemble - 48^3x96"
Ensemble added: ID=3
Marked PRODUCTION in DB: OK

$ find TUNING -name "*" -type d | head -5
```

```
TUNING
TUNING/b6.0
TUNING/b6.0/b1.8Ls24
TUNING/b6.0/b1.8Ls24/mc0.8555
TUNING/b6.0/b1.8Ls24/mc0.8555/ms0.0725
```

2.6 query: List and Inspect Ensembles

Purpose: Query ensemble information from the database.

The query command has two distinct modes: list mode (shows all ensembles) and detail mode (shows information for one specific ensemble).

Options:

- `-e, --ensemble ENSEMBLE`: Show details for specific ensemble (ID, path, or ".")
- `--detailed`: In list mode, show descriptions and operation counts
- `--sort-by-id`: In list mode, sort ensembles by EID instead of by parameters
- `--dir`: Show only the directory path (only works with `--ensemble`)

Two Modes:

1. List Mode (no ensemble specified): Shows a spreadsheet-like table of all ensembles with columns: `EID > beta > b > Ls > mc > ms > ml > L > T > LAST_OP > LAST_USER`

By default, ensembles are sorted numerically/alphabetically by parameters. Use `--sort-by-id` to sort by EID.

2. Detail Mode (with ensemble specified): Shows complete information for one ensemble including:

- All physics parameters (beta, masses, lattice dimensions)
- Full operation history with timestamps and parameters
- Job status and configuration ranges

Flexible Ensemble Identification: The `--ensemble` parameter accepts multiple formats:

- Ensemble ID: `-e 1`
- Relative path: `-e ./TUNING/b6.0/b1.8Ls24/mc0.85/ms0.07/ml0.02/L32/T64`
- Absolute path: `-e /full/path/to/ensemble`
- Current directory: `-e .` (when run from within ensemble directory)

Examples:

```
# List all ensembles in spreadsheet format (sorted by parameters)
$ mdwf_db query
EID > beta > b > Ls > mc > ms > ml > L > T > LAST_OP > LAST_USER
1 > 6.0 > 1.8 > 24 > 0.8555 > 0.0725 > 0.0195 > 32 > 64 > HMC_TUNE > wyatt
2 > 5.8 > 2.0 > 16 > 0.9 > 0.08 > 0.02 > 24 > 48 > GLU_SMEAR > alice
3 > 6.2 > 1.5 > 32 > 0.8 > 0.06 > 0.015 > 48 > 96 > WIT_MESON2PT > bob

# List all ensembles sorted by EID
$ mdwf_db query --sort-by-id
EID > beta > b > Ls > mc > ms > ml > L > T > LAST_OP > LAST_USER
1 > 6.0 > 1.8 > 24 > 0.8555 > 0.0725 > 0.0195 > 32 > 64 > HMC_TUNE > wyatt
2 > 5.8 > 2.0 > 16 > 0.9 > 0.08 > 0.02 > 24 > 48 > GLU_SMEAR > alice
3 > 6.2 > 1.5 > 32 > 0.8 > 0.06 > 0.015 > 48 > 96 > WIT_MESON2PT > bob

# List with descriptions and operation counts
$ mdwf_db query --detailed
EID > beta > b > Ls > mc > ms > ml > L > T > LAST_OP > LAST_USER > Description > Operations
```

```
1 > 6.0 > 1.8 > 24 > 0.8555 > 0.0725 > 0.0195 > 32 > 64 > HMC_TUNE > wyatt > First test
    ensemble - 32^3x64 > 3
2 > 5.8 > 2.0 > 16 > 0.9 > 0.08 > 0.02 > 24 > 48 > GLU_SMEAR > alice > Second test ensemble
    - 24^3x48 > 1
3 > 6.2 > 1.5 > 32 > 0.8 > 0.06 > 0.015 > 48 > 96 > WIT_MESON2PT > bob > Large production
    ensemble - 48^3x96 > 2

# Show detailed information for specific ensemble by ID
$ mdwf_db query -e 2
ID          = 2
Directory   = /path/to/ENSEMBLES/b5.8/b2.0Ls16/mc0.9/ms0.08/ml0.02/L24/T48
Status      = PRODUCTION
Created     = 2025-08-06T12:40:49.869456
Description = Second test ensemble - 24^3x48 with different parameters
Parameters:
    L = 24
    Ls = 16
    T = 48
    b = 2.0
    beta = 5.8
    mc = 0.9
    ml = 0.02
    ms = 0.08

=== Operation history ===
Op 1: GLU_SMEAR [COMPLETED]
  Created: 2025-08-06T12:41:44.054003 (by alice)
  Updated: 2025-08-06T12:42:15.123456
    config_start = 10
    config_end = 30
    exit_code = 0
    runtime = 1800

# Query using relative path instead of ID
$ mdwf_db query -e ./TUNING/b6.0/b1.8Ls24/mc0.8555/ms0.0725/ml0.0195/L32/T64
[Shows same detailed information as above]

# Query from within ensemble directory using "."
$ cd TUNING/b6.0/b1.8Ls24/mc0.8555/ms0.0725/ml0.0195/L32/T64
$ mdwf_db query -e .
[Shows same detailed information as above]

# Show only the directory path for an ensemble
$ mdwf_db query -e 1 --dir
/path/to/TUNING/b6.0/b1.8Ls24/mc0.8555/ms0.0725/ml0.0195/L32/T64
```

2.7 promote-ensemble: Move to Production

Purpose: Move ensemble from TUNING to PRODUCTION status and directory.

Physically moves the directory and updates the database record. Records a PROMOTE_ENSEMBLE operation in the history.

Options:

- `-e, --ensemble ENSEMBLE`: **Required.** Ensemble to promote (ID, path, or ".")
- `--base-dir BASE_DIR`: Root directory containing TUNING/ and ENSEMBLES/
- `--force`: Skip confirmation prompt

Requirements:

- Ensemble must have TUNING status
- Target ENSEMBLES/ directory must not exist
- Source must be under TUNING/

Examples:

```
# Promote with confirmation prompt
$ mdwf_db promote-ensemble -e 1
Promote ensemble 1:
  from /scratch/lattice/TUNING/b6.0/b1.8Ls24/mc0.8555/ms0.0725/ml0.0195/L32/T64
    to /scratch/lattice/ENSEMBLES/b6.0/b1.8Ls24/mc0.8555/ms0.0725/ml0.0195/L32/T64
Proceed? (y/N) y
Created operation 2: Created
Promotion OK

# Promote with --force flag (skips confirmation)
$ mdwf_db promote-ensemble -e 2 --force
Promote ensemble 2:
  from /private/tmp/mdwf_expanded_test/TUNING/b5.8/b2.0Ls16/mc0.9/ms0.08/ml0.02/L24/T48
    to /private/tmp/mdwf_expanded_test/ENSEMBLES/b5.8/b2.0Ls16/mc0.9/ms0.08/ml0.02/L24/T48
Created operation 4: Created
Promotion OK

# Verify the move
$ ls ENSEMBLES/b6.0/b1.8Ls24/mc0.8555/ms0.0725/ml0.0195/L32/T64/
cnfg/  jlog/  log_hmc/  slurm/
```

2.8 hmc-script: Generate HMC Scripts

Purpose: Generate HMC XML parameters and SLURM batch script for gauge generation.
Creates both XML parameter files and complete SLURM scripts for GPU HMC execution.
Options:

- `-e, --ensemble-id ENSEMBLE_ID`: **Required.** Ensemble ID
- `-a, --account ACCOUNT`: **Required.** SLURM account name
- `-m, --mode {tepid,continue,reseed}`: **Required.** HMC run mode
- `-x, --xml-params XML_PARAMS`: Space-separated XML parameters
- `-j, --job-params JOB_PARAMS`: Space-separated SLURM job parameters
- `-o, --output-file OUTPUT_FILE`: Custom output script path
- `--use-default-params`: Load from ensemble default parameter file
- `--params-variant VARIANT`: Use specific parameter variant
- `--save-default-params`: Save current parameters to default file
- `--save-params-as VARIANT`: Save under custom variant name

HMC Modes:

- **tepid**: Initial thermalization run (TepidStart)
- **continue**: Continue from existing checkpoint (CheckpointStart)
- **reseed**: Start new run with different seed (CheckpointStartReseed)

Required Job Parameters: `cfg_max`: Maximum configuration number to generate
Common XML Parameters: StartTrajectory, Trajectories, MetropolisTest, MDsteps, trajL, Seed
Examples:

```
# Basic tepid HMC script with minimal parameters
$ printf "/usr/bin/hmc_exec\n/usr/bin/core_bind.sh\n" | \
  mdwf_db hmc-script -e 1 -a m2986 -m tepid -j "cfg_max=50 time_limit=6:00:00"
Generated HMC script: /private/tmp/.../TUNING/.../slurm/hmc_1_tepid.sbatch

# Continue mode with custom XML and job parameters
$ printf "/usr/bin/hmc_exec\n/usr/bin/core_bind.sh\n" | \
  mdwf_db hmc-script -e 2 -a nersc -m continue \
    -j "cfg_max=200 time_limit=12:00:00 nodes=2" \
    -x "StartTrajectory=50 Trajectories=100 MDsteps=4"
Generated HMC script: /private/tmp/.../TUNING/.../slurm/hmc_2_continue.sbatch

# Use stored default parameters from file
$ printf "/usr/bin/hmc_exec\n/usr/bin/core_bind.sh\n" | \
  mdwf_db hmc-script -e 1 -a m2986 -m continue --use-default-params -j "nodes=2"
Loaded HMC continue default parameters from .../mdwf_default_params.yaml
Generated HMC script: /private/tmp/.../TUNING/.../slurm/hmc_1_continue.sbatch

# Save parameters for future reuse
$ printf "/usr/bin/hmc_exec\n/usr/bin/core_bind.sh\n" | \
  mdwf_db hmc-script -e 1 -a m2986 -m tepid \
    -j "cfg_max=25 time_limit=3:00:00" \
    -x "MDsteps=6 trajL=0.5" --save-default-params
Generated HMC script: /private/tmp/.../slurm/hmc_1_tepid.sbatch
Saved parameters to default params: hmc.tepid
```

Complete Generated SLURM Scripts:

Here are the complete SLURM batch scripts generated by different option combinations, showing all the logic, environment setup, database integration, and execution flow:

Example 1: Complete Tepid Mode Script ($24^3 \times 48$ lattice)

```
# Generated by: mdwf_db hmc-script -e 1 -a physics123 -m tepid \
#   -j "time_limit=1:00:00 nodes=2 ntasks_per_node=4 cfg_max=100"

#!/bin/bash
#SBATCH -A physics123
#SBATCH -C gpu
#SBATCH -q regular
#SBATCH -t 1:00:00
#SBATCH --cpus-per-task=32
#SBATCH -N 2
#SBATCH --ntasks-per-node=4
#SBATCH --gres=gpu:1
#SBATCH --gpu-bind=none
#SBATCH --mail-type=BEGIN,END
#SBATCH --mail-user=wyatt
#SBATCH --signal=B:TERM@60

batch="$0"
DB="/path/to/mdwf_ensembles.db"
EID=1
mode="tepid"
ens="b2.10_b1.0Ls32_mc0.04_ms0.04_ml0.005_L24_T48"
ens_rel="24^3x48 test ensemble"
VOL="24.24.24.48"
EXEC="/opt/exec_file"
BIND="/opt/exec_file"
n_trajec=100
cfg_max=100
mpi="2.1.1.2"

cd /path/to/24^3x48\ test\ ensemble

echo "ens = $ens"
echo "ens_dir = /path/to/24^3x48 test ensemble"
echo "EXEC = $EXEC"
echo "BIND = $BIND"
echo "n_trajec = $n_trajec"
echo "cfg_max = $cfg_max"

mkdir -p cnfg
mkdir -p log_hmc

start=`ls -v cnfg/| grep lat | tail -1 | sed 's/[^0-9]*//g'`
if [[ -z $start ]]; then
    echo "no configs - start is empty - doing TepidStart"
    start=0
fi

# check if start <= cfg_max
if [[ $start -ge $cfg_max ]]; then
    echo "your latest config is greater than the target:"
    echo "  $start >= $cfg_max"
    exit
fi

echo "cfg_current = $start"

# Update database to show running job
out=$(
  mdwf_db update \
    --db-file="$DB" \
    --ensemble-id=$EID \
    --operation-type="$mode" \
    --status=RUNNING \
    --params="config_start=$start config_end=$(( start + n_trajec )) config_increment=$n_
    trajec slurm_job=$SLURM_JOB_ID exec_path=$EXEC bind_script=$BIND"
)
echo "$out"
op_id=${out#*operation }
op_id=${op_id%%:*}
export op_id

# Generate HMC parameters XML
mdwf_db hmc-xml -e $EID -m $mode --params "StartTrajectory=$start Trajectories=$n_trajec"

cp HMCparameters.xml cnfg/
cd cnfg

export CRAY_ACCEL_TARGET=nvidia80
export MPICH_OFI_NIC_POLICY=GPU
export SLURM_CPU_BIND="cores"
export MPICH_GPU_SUPPORT_ENABLED=1
export MPICH_RDMA_ENABLED_CUDA=1
export MPICH_GPU_IPC_ENABLED=1
export MPICH_GPU_EAGER_REGISTER_HOST_MEM=0
export MPICH_GPU_NO_ASYNC_MEMCPY=0
export OMP_NUM_THREADS=8

echo "Nthreads $OMP_NUM_THREADS"

echo "START `date`"
srun $BIND $EXEC --mpi $mpi --grid $VOL --accelerator-threads 32 --dslash-unroll --shm 2048
    --comms-overlap -shm-mpi 0 > ../log_hmc/log_b2.10_b1.0Ls32_mc0.04_ms0.04_ml0.005_L24_T48
    .$start
EXIT_CODE=$?
echo "STOP `date`"

# Update database with job status
STATUS=COMPLETED
[[ $EXIT_CODE -ne 0 ]] && STATUS=FAILED

mdwf_db update \
  --db-file="$DB" \
  --ensemble-id=$EID \
  --operation-id=$op_id \
  --operation-type="$mode" \
  --status=$STATUS \
  --params="exit_code=$EXIT_CODE runtime=$SECONDS slurm_job=$SLURM_JOB_ID host=$(hostname)"

echo "DB updated: operation $op_id to $STATUS (exit=$EXIT_CODE) [SLURM_JOB_ID=$SLURM_JOB_ID]
    "

# Check if we should resubmit
if [[ $EXIT_CODE -eq 0 && "true" == "true" && $mode != "reseed" ]]; then
    next_start=$((start + n_trajec))
    if [[ $next_start -lt $cfg_max ]]; then
        echo "Resubmitting with start=$next_start in continue mode"
        # Generate new XML for continue mode
        mdwf_db hmc-xml -e $EID -m continue --params "StartTrajectory=$next_start
    Trajectories=$n_trajec"
        # Resubmit the job
        sbatch --dependency=afterok:$SLURM_JOBID $batch
    else
        echo "Reached target config_max=$cfg_max"
    fi
fi

exit $EXIT_CODE
```

Example 2: Complete Continue Mode Script ($32^3\times64$ lattice)

```
# Generated by: mdwf_db hmc-script -e 2 -a physics456 -m continue \
#   -j "time_limit=4:00:00 nodes=4 ntasks_per_node=8 cfg_max=500"

#!/bin/bash
#SBATCH -A physics456
#SBATCH -C gpu
#SBATCH -q regular
#SBATCH -t 4:00:00
#SBATCH --cpus-per-task=32
#SBATCH -N 4
#SBATCH --ntasks-per-node=8
#SBATCH --gres=gpu:1
#SBATCH --gpu-bind=none
#SBATCH --mail-type=BEGIN,END
#SBATCH --mail-user=wyatt
#SBATCH --signal=B:TERM@60

batch="$0"
DB="/path/to/mdwf_ensembles.db"
EID=2
mode="continue"
ens="b2.13_b1.0Ls16_mc0.04_ms0.04_ml0.005_L32_T64"
ens_rel="32^3x64 production ensemble"
VOL="32.32.32.64"
EXEC="/usr/exec_file_2"
BIND="/usr/local/bin/bind.sh"
n_trajec=500
cfg_max=500
mpi="2.1.1.2"

cd /path/to/32^3x64\ production\ ensemble

echo "ens = $ens"
echo "ens_dir = /path/to/32^3x64 production ensemble"
echo "EXEC = $EXEC"
echo "BIND = $BIND"
echo "n_trajec = $n_trajec"
echo "cfg_max = $cfg_max"

mkdir -p cnfg
mkdir -p log_hmc

start=`ls -v cnfg/| grep lat | tail -1 | sed 's/[^0-9]*//g'`
```

```
if [[ -z $start ]]; then
    echo "no configs - start is empty - doing TepidStart"
    start=0
fi

# check if start <= cfg_max
if [[ $start -ge $cfg_max ]]; then
    echo "your latest config is greater than the target:"
    echo "  $start >= $cfg_max"
    exit
fi

echo "cfg_current = $start"

# Update database to show running job
out=$(
  mdwf_db update \
    --db-file="$DB" \
    --ensemble-id=$EID \
    --operation-type="$mode" \
    --status=RUNNING \
    --params="config_start=$start config_end=$(( start + n_trajec )) config_increment=$n_
    trajec slurm_job=$SLURM_JOB_ID exec_path=$EXEC bind_script=$BIND"
)
echo "$out"
op_id=${out#*operation }
op_id=${op_id%%:*}
export op_id

# Generate HMC parameters XML
mdwf_db hmc-xml -e $EID -m $mode --params "StartTrajectory=$start Trajectories=$n_trajec"

cp HMCparameters.xml cnfg/
cd cnfg

export CRAY_ACCEL_TARGET=nvidia80
export MPICH_OFI_NIC_POLICY=GPU
export SLURM_CPU_BIND="cores"
export MPICH_GPU_SUPPORT_ENABLED=1
export MPICH_RDMA_ENABLED_CUDA=1
export MPICH_GPU_IPC_ENABLED=1
export MPICH_GPU_EAGER_REGISTER_HOST_MEM=0
export MPICH_GPU_NO_ASYNC_MEMCPY=0
export OMP_NUM_THREADS=8

echo "Nthreads $OMP_NUM_THREADS"

echo "START `date`"
srun $BIND $EXEC --mpi $mpi --grid $VOL --accelerator-threads 32 --dslash-unroll --shm 2048
    --comms-overlap -shm-mpi 0 > ../log_hmc/log_b2.13_b1.0Ls16_mc0.04_ms0.04_ml0.005_L32_T64
    .$start
EXIT_CODE=$?
echo "STOP `date`"

# Update database with job status
STATUS=COMPLETED
[[ $EXIT_CODE -ne 0 ]] && STATUS=FAILED

mdwf_db update \
  --db-file="$DB" \
  --ensemble-id=$EID \
  --operation-id=$op_id \
  --operation-type="$mode" \
  --status=$STATUS \
  --params="exit_code=$EXIT_CODE runtime=$SECONDS slurm_job=$SLURM_JOB_ID host=$(hostname)"

echo "DB updated: operation $op_id to $STATUS (exit=$EXIT_CODE) [SLURM_JOB_ID=$SLURM_JOB_ID]
    "
```

```

# Check if we should resubmit
if [[ $EXIT_CODE -eq 0 && "true" == "true" && $mode != "reseed" ]]; then
    next_start=$((start + n_trajec))
    if [[ $next_start -lt $cfg_max ]]; then
        echo "Resubmitting with start=$next_start in continue mode"
        # Generate new XML for continue mode
        mdwf_db hmc-xml -e $EID -m continue --params "StartTrajectory=$next_start
    Trajectories=$n_trajec"
        # Resubmit the job
        sbatch --dependency=afterok:$SLURM_JOBID $batch
    else
        echo "Reached target config_max=$cfg_max"
    fi
fi

exit $EXIT_CODE
```

Key Differences Between Tepid and Continue Scripts:

- **SLURM Resources:** Continue mode uses more nodes (4 vs 2) and tasks (8 vs 4) for production runs
- **Grid Size:** Different lattice volumes reflected in VOL variable ($32^3 \times 64$ vs $24^3 \times 48$)
- **Configuration Targets:** Higher cfg_max for production (500 vs 100)
- **Executable Paths:** Different EXEC and BIND paths based on user input
- **Environment:** Both scripts set identical GPU/MPI environment variables for HPC execution
- **Database Integration:** Both track operations with status updates and parameter logging
- **Auto-resubmission:** Both include logic to chain jobs until cfg_max is reached
- **Directory Structure:** Ensemble-specific paths derived from physics parameters

2.9 hmc-xml: Generate HMC XML Files

Purpose: Generate standalone HMC XML parameter files.
Creates XML files with HMC parameters without generating SLURM scripts.
Options:

- `-e, --ensemble-id ENSEMBLE_ID`: **Required.** Ensemble ID
- `-m, --mode {tepid,continue,reseed}`: **Required.** HMC run mode
- `-b, --base-dir BASE_DIR`: Root directory for TUNING/ENSEMBLES
- `-x, --xml-params XML_PARAMS`: Space-separated XML parameters to override

Examples:

```
$ mdwf_db hmc-xml -e 1 -m tepid -x "Trajectories=50 MDsteps=4 trajL=0.75"
Generated XML file: /scratch/lattice/ENSEMBLES/.../HMCparameters.tepid.xml

$ mdwf_db hmc-xml -e 2 -m continue -x "Trajectories=100"
Generated XML file: /scratch/lattice/ENSEMBLES/.../HMCparameters.continue.xml
```

Generated XML Examples:
The XML files generated show how different modes affect the HMC parameters:
Tepid mode XML (ensemble 1):

```xml
<?xml version="1.0" ?>
<grid>
  <HMCparameters>
    <StartTrajectory>0</StartTrajectory>
    <Trajectories>100</Trajectories>
    <MetropolisTest>false</MetropolisTest>
    <StartingType>TepidStart</StartingType>      <!-- Tepid mode -->
    <Seed>776304</Seed>
    <MD>
      <name>
        <elem>OMF2_5StepV</elem>
        <elem>OMF2_5StepV</elem>
        <elem>OMF4_11StepV</elem>
      </name>
      <lvl_sizes>
        <elem>9</elem>
        <elem>1</elem>
        <elem>1</elem>
      </lvl_sizes>
    </MD>
    <MDsteps>1</MDsteps>
    <trajL>0.75</trajL>
  </HMCparameters>
</grid>
```

Continue mode XML (ensemble 2):

```xml
<?xml version="1.0" ?>
<grid>
  <HMCparameters>
    <StartTrajectory>12</StartTrajectory>        <!-- Auto-detected start -->
    <Trajectories>50</Trajectories>
    <MetropolisTest>true</MetropolisTest>        <!-- Different from tepid -->
    <StartingType>CheckpointStart</StartingType> <!-- Continue mode -->
    <Seed>368640</Seed>                         <!-- Different seed -->
    <MD>
      <name>
        <elem>OMF2_5StepV</elem>
        <elem>OMF2_5StepV</elem>
        <elem>OMF4_11StepV</elem>
      </name>
      <lvl_sizes>
        <elem>9</elem>
        <elem>1</elem>
        <elem>1</elem>
      </lvl_sizes>
    </MD>
    <MDsteps>1</MDsteps>
    <trajL>0.75</trajL>
  </HMCparameters>
</grid>
```

2.10 smear-script: Generate Smearing Scripts

Purpose: Generate complete SLURM script for configuration smearing using GLU.
Creates GLU input files and SLURM batch scripts for GPU smearing execution.
Options:

- `-e, --ensemble-id ENSEMBLE_ID`: **Required.** Ensemble ID
- `-j, --job-params JOB_PARAMS`: Space-separated SLURM job parameters
- `-g, --glu-params GLU_PARAMS`: Space-separated GLU parameters
- `-o, --output-file OUTPUT_FILE`: Custom output script path
- `--use-default-params`: Load from ensemble default parameter file

- `--params-variant VARIANT`: Use specific parameter variant
- `--save-default-params`: Save current parameters to default file
- `--save-params-as VARIANT`: Save under custom variant name

Required Job Parameters: mail_user, config_start, config_end
Common GLU Parameters: SMEARTYPE, SMITERS, ALPHA1, ALPHA2, ALPHA3
Examples:

```
# Basic smearing job with custom GLU parameters
$ mdwf_db smear-script -e 1 \
    -j "mail_user=user@nersc.gov config_start=10 config_end=30 time_limit=3:00:00" \
    -g "SMITERS=8 ALPHA1=0.1"
Generated GLU input file: /private/tmp/.../cnfg_STOUT8/glu_smear.in
Wrote smearing SBATCH script to /private/tmp/.../slurm/glu_smear_STOUT8_10_30.sh

# Large-scale smearing with APE algorithm and multiple nodes
$ mdwf_db smear-script -e 3 \
    -j "mail_user=admin@lab.edu config_start=100 config_end=200 nodes=2 time_limit=8:00:00"
    \
    -g "SMEARTYPE=APE SMITERS=12 ALPHA1=0.05"
Generated GLU input file: /private/tmp/.../cnfg_STOUT8/glu_smear.in
Wrote smearing SBATCH script to /private/tmp/.../slurm/glu_smear_STOUT8_100_200.sh

# Use default parameters with selective overrides
$ mdwf_db smear-script -e 1 --use-default-params \
    --params-variant stout8 -j "time_limit=4:00:00"
Loaded smearing.stout8 default parameters from .../mdwf_default_params.yaml
Generated GLU input: /scratch/lattice/.../cnfg_STOUT8/glu_smear.in
Generated script: /scratch/lattice/.../slurm/glu_smear_STOUT8_100_200.sh
```

2.11 glu-input: Generate GLU Input Files

Purpose: Generate GLU input files for gauge field utility operations.
Creates properly formatted GLU input files with ensemble parameters and custom settings.
Options:

- `-e, --ensemble-id ENSEMBLE_ID`: **Required.** Ensemble ID
- `-o, --output-file OUTPUT_FILE`: **Required.** Output file path
- `-g, --glu-params GLU_PARAMS`: Space-separated GLU parameters
- `-t, --type {smearing,gluon_props,other}`: Calculation type (default: smearing)

Common Parameters: CONFNO, SMEARTYPE, SMITERS, ALPHA1, GFTYPE, ACCURACY
Examples:

```
# Basic GLU input for smearing (default type)
$ mdwf_db glu-input -e 1 -o smear_config.in -g "CONFNO=168 SMITERS=50 ALPHA1=0.1"
Generated GLU input file: smear_config.in

# GLU input for gluon propagator calculations
$ mdwf_db glu-input -e 1 -o /tmp/custom_glu.in \
    -g "CONFNO=25 SMITERS=15 ALPHA1=0.05" -t gluon_props
Generated GLU input file: /tmp/custom_glu.in

# GLU input with gauge fixing parameters
$ mdwf_db glu-input -e 2 -o gauge_fix.in -t other \
    -g "CONFNO=100 GFTYPE=LANDAU ACCURACY=16"
Generated GLU input file: gauge_fix.in
```

Generated GLU Input File Example:
Here's an example of the GLU input file content generated for a $24^3 \times 48$ ensemble:

```
# Generated by: mdwf_db glu-input -e 1 -o test_glu.in -g "APE_alpha=0.6 APE_iter=50"

MODE = SMEARING
HEADER = NERSC
    DIM_0 = 24          # Automatically set from ensemble L parameter
    DIM_1 = 24
    DIM_2 = 24
    DIM_3 = 48          # Automatically set from ensemble T parameter
CONFNO = 24
RANDOM_TRANSFORM = NO
SEED = 0
GFTYPE = COULOMB        # Default gauge fixing
    GF_TUNE = 0.09
    ACCURACY = 14
    MAX_ITERS = 650
CUTTYPE = GLUON_PROPS
FIELD_DEFINITION = LINEAR
    MOM_CUT = CYLINDER_CUT
    MAX_T = 7
    MAXMOM = 4
    CYL_WIDTH = 2.0
    ANGLE = 60
    OUTPUT = ./
SMEARTYPE = STOUT       # Default smearing type
    DIRECTION = ALL
    SMITERS = 8         # Default iterations
    ALPHA1 = 0.75       # Default alpha values
    ALPHA2 = 0.4
    ALPHA3 = 0.2
U1_MEAS = U1_RECTANGLE
    U1_ALPHA = 0.0796
    U1_CHARGE = -1.0
CONFIG_INFO = 2+1DWF_b2.25_TEST
    STORAGE = CERN
BETA = 6.0              # Derived from ensemble physics parameters
    ITERS = 1500
    MEASURE = 1
    OVER_ITERS = 4
    SAVE = 25
    THERM = 100
```

2.12 meson2pt-script: Generate Meson Correlator Scripts

Purpose: Generate WIT meson correlator measurement SLURM scripts.
Creates WIT input files and SLURM scripts for meson correlator calculations.
Options:

- `-e, --ensemble ENSEMBLE`: **Required.** Ensemble ID, directory path, or "." for current directory
- `-j, --job-params JOB_PARAMS`: **Required.** Space-separated key=val for SLURM job parameters
- `-w, --wit-params WIT_PARAMS`: Space-separated key=val for WIT parameters (dot notation)
- `-o, --output-file OUTPUT_FILE`: Output SBATCH script path (auto-generated if not specified)
- `--use-default-params`: Load parameters from ensemble default parameter file
- `--params-variant PARAMS_VARIANT`: Specify which parameter variant to use
- `--save-default-params`: Save current command parameters to default parameter file
- `--save-params-as SAVE_PARAMS_AS`: Save current parameters under specific variant name

Required Job Parameters:

- `mail_user`: Email address for job notifications

- `config_start`: First configuration number to measure
- `config_end`: Last configuration number to measure
- `config_inc`: Step/increment between configurations

Required WIT Parameters:

- `Configurations.first`: First configuration number
- `Configurations.last`: Last configuration number
- `Configurations.step`: Step between configurations

Common WIT Parameters:

- `Witness.no_prop`: Number of witness propagators
- `Solver 0.nmx`: Maximum solver iterations
- `Propagator 0.Source`: Source type for propagators

Examples:

```
# Basic meson measurement with debug queue
$ mdwf_db meson2pt-script -e 1 \
    -j "mail_user=test@example.com config_start=10 config_end=20 config_inc=2" \
    -w "Configurations.first=10 Configurations.last=20 Configurations.step=2"

# Use stored default parameters
$ mdwf_db meson2pt-script -e 1 --use-default-params

# Save current parameters for later reuse
$ mdwf_db meson2pt-script -e 1 \
    -j "mail_user=user@nersc.gov config_start=100 config_end=200 config_inc=4" \
    --save-default-params

Generated WIT SBATCH script: \texttt{/private/tmp/.../meson2pt/meson2pt\_10\_20.sh}
Wrote WIT SBATCH script to \texttt{/private/tmp/.../meson2pt/meson2pt\_10\_20.sh}

# Large-scale measurement with wall sources
$ mdwf_db meson-2pt -e 3 \
    -j "queue=regular time_limit=10:00:00 nodes=4 cpus_per_task=16 mail_user=hpc@university.
    edu" \
    -w "Configurations.first=100 Configurations.last=300 Propagator 0.Source=Wall"
WARNING: WIT parameter '0.Source' was provided but is not used in DWF.in
Generated WIT input file: /private/tmp/.../meson2pt/DWF.in
Generated WIT SBATCH script: /private/tmp/.../meson2pt/meson2pt_100_300.sh
Wrote WIT SBATCH script to /private/tmp/.../meson2pt/meson2pt_100_300.sh

# Use default parameters with custom configuration range
$ mdwf_db meson-2pt -e 1 --use-default-params \
    -w "Configurations.first=200 Configurations.last=250" -j "nodes=2"
Loaded meson_2pt.default default parameters from .../mdwf_default_params.yaml
Generated WIT input: /scratch/lattice/.../meson2pt/DWF.in
Generated script: /scratch/lattice/.../meson2pt/meson2pt_200_250.sh
```

2.13 wit-input: Generate WIT Input Files

Purpose: Generate WIT input files for meson correlator measurements.
Creates properly formatted WIT input files with ensemble parameters.
Options:

- `-e, --ensemble-id ENSEMBLE_ID`: **Required.** Ensemble ID
- `-o, --output-file OUTPUT_FILE`: **Required.** Output file path

- `-w, --wit-params WIT_PARAMS`: Space-separated WIT parameters (dot notation)

Common Parameters: Configurations.first, Configurations.last, Configurations.step, Propagator 0.Source
Example:

```
$ mdwf_db wit-input -e 1 -o DWF.in \
    -w "Configurations.first=100 Configurations.last=200 Configurations.step=2"
Generated WIT input file: DWF.in
```

2.14 update: Track Operation Status

Purpose: Create or update operation records in the database.
Records operation status, parameters, and execution details for tracking job progress.
Options:

- `-e, --ensemble-id ENSEMBLE_ID`: **Required.** Ensemble ID
- `-o, --operation-type OPERATION_TYPE`: **Required.** Operation type
- `-s, --status {RUNNING,COMPLETED,FAILED}`: **Required.** Operation status
- `-i, --operation-id OPERATION_ID`: Existing operation ID to update
- `-p, --params PARAMS`: Space-separated key=val operation details

Common Operation Types: HMC_TUNE, HMC_PRODUCTION, GLU_SMEAR, WIT_MESON2PT, PROMOTE_ENSEMBLE
Common Parameters: config_start, config_end, exit_code, runtime, slurm_job, host
Examples:

```
# Record new running HMC operation
$ mdwf_db update -e 1 -o HMC_TUNE -s RUNNING \
    -p "config_start=0 config_end=50 slurm_job=123456"
Created operation 1: Created

# Record completed smearing operation with timing info
$ mdwf_db update -e 2 -o GLU_SMEAR -s COMPLETED \
    -p "config_start=10 config_end=30 exit_code=0 runtime=1800"
Created operation 2: Created

# Record failed meson measurement with error details
$ mdwf_db update -e 3 -o WIT_MESON2PT -s FAILED \
    -p "config_start=100 config_end=150 exit_code=1 error_msg=Out_of_memory"
Created operation 3: Created

# Update existing operation status to completed
$ mdwf_db update -e 1 -o HMC_TUNE -s COMPLETED -i 1 \
    -p "exit_code=0 runtime=14400 final_config=50"
Updated operation 1: Updated

# Record operation with hostname and user info
$ mdwf_db update -e 2 -o PROMOTE_ENSEMBLE -s COMPLETED \
    -p "host=perlmutter-node01 runtime=5"
Created operation 4: Created
```

2.15 clear-history: Clear Operation History

Purpose: Clear all operation history for an ensemble while preserving the ensemble record.
Removes all operation records but keeps ensemble metadata and physics parameters.
Options:

- `-e, --ensemble ENSEMBLE`: **Required.** Ensemble to clear (ID, path, or ".")

- `--force`: Skip confirmation prompt

What is Removed: All operation records, parameters, timestamps, and status information
What is Preserved: Ensemble record, physics parameters, description, creation time
Examples:

```
# Clear history with confirmation prompt
$ mdwf_db clear-history -e 1
Clear all operation history for ensemble 1? This cannot be undone. (y/N) y
Cleared 2 operations for ensemble 1

# Clear history with --force flag (no prompt)
$ mdwf_db clear-history -e 3 --force
Ensemble 3: /private/tmp/.../ENSEMBLES/b6.2/b1.5Ls32/mc0.8/ms0.06/ml0.015/L48/T96
Found 1 operation(s) to clear
Successfully cleared 1 operation(s) from ensemble 3

# Verify history is cleared (query shows no operations)
$ mdwf_db query -e 3
ID          = 3
Directory   = /private/tmp/.../ENSEMBLES/b6.2/b1.5Ls32/mc0.8/ms0.06/ml0.015/L48/T96
Status      = PRODUCTION
Created     = 2025-08-06T12:40:50.104567
Description = Large production ensemble - 48^3x96
Parameters:
    L = 48
    ...

=== Operation history ===
No operations recorded
```

2.16 remove-ensemble: Remove Ensemble

Purpose: Remove ensemble and all its operations from the database.
Completely removes ensemble record and all associated operations. Directory structure is not deleted.
Options:

- `-e, --ensemble ENSEMBLE`: **Required.** Ensemble to remove (ID, path, or ".")
- `--force`: Skip confirmation prompt

Example:

```
$ mdwf_db remove-ensemble -e 1
Remove ensemble 1 and all its operations? This cannot be undone. (y/N) y
Removed ensemble 1 and 3 operations
```

2.17 mres-script: Generate MRES Measurement Scripts

Purpose: Generate WIT MRES measurement SLURM scripts for mass renormalization.
Options:

- `-e, --ensemble ENSEMBLE`: **Required.** Ensemble ID, directory path, or "." for current directory
- `-j, --job-params JOB_PARAMS`: **Required.** Space-separated key=val for SLURM job parameters
- `-g, --glu-params GLU_PARAMS`: Space-separated key=val for GLU parameters
- `-o, --output-file OUTPUT_FILE`: Output SBATCH script path (auto-generated if not specified)
- `--use-default-params`: Load parameters from ensemble default parameter file
- `--params-variant PARAMS_VARIANT`: Specify which parameter variant to use

- `--save-default-params`: Save current command parameters to default parameter file
- `--save-params-as SAVE_PARAMS_AS`: Save current parameters under specific variant name

Required Job Parameters:

- `mail_user`: Email address for job notifications
- `config_start`: First configuration number to measure
- `config_end`: Last configuration number to measure
- `config_inc`: Step/increment between configurations

Examples:

```
# Basic MRES measurement job
mdwf_db mres-script -e 1 \
  -j "mail_user=user@example.com config_start=100 config_end=200 config_inc=4"

# Use stored default parameters
mdwf_db mres-script -e 1 --use-default-params

# Save current parameters for later reuse
mdwf_db mres-script -e 1 \
  -j "mail_user=user@nersc.gov config_start=100 config_end=200 config_inc=4" \
  --save-default-params
```

2.18 wflow-script: Generate Gradient Flow Scripts

Purpose: Generate gradient flow SLURM scripts for Wilson flow measurements.
Options:

- `-e, --ensemble ENSEMBLE`: **Required.** Ensemble ID, directory path, or "." for current directory
- `-j, --job-params JOB_PARAMS`: **Required.** Space-separated key=val for SLURM job parameters
- `-g, --glu-params GLU_PARAMS`: Space-separated key=val for GLU parameters
- `-o, --output-file OUTPUT_FILE`: Output SBATCH script path (auto-generated if not specified)
- `--use-default-params`: Load parameters from ensemble default parameter file
- `--params-variant PARAMS_VARIANT`: Specify which parameter variant to use
- `--save-default-params`: Save current command parameters to default parameter file
- `--save-params-as SAVE_PARAMS_AS`: Save current parameters under specific variant name

Required Job Parameters:

- `mail_user`: Email address for job notifications
- `config_start`: First configuration number to measure
- `config_end`: Last configuration number to measure
- `config_inc`: Step/increment between configurations

Examples:

```
# Basic gradient flow job
mdwf_db wflow-script -e 1 \
  -j "mail_user=user@example.com config_start=100 config_end=200 config_inc=4"

# Use stored default parameters
mdwf_db wflow-script -e 1 --use-default-params

# Save current parameters for later reuse
mdwf_db wflow-script -e 1 \
  -j "mail_user=user@nersc.gov config_start=100 config_end=200 config_inc=4" \
  --save-default-params
```

2.19 default_params: Parameter Management

Purpose: Manage default parameter files for storing operation parameters.

Save "recipes" of parameters that work well for specific ensembles and reuse them in script generation commands.

Subcommands:

- `generate`: Generate a template default parameter file
- `show`: Display current default parameters
- `edit`: Edit default parameter file
- `validate`: Validate default parameter file

Options:

- `-e, --ensemble ENSEMBLE`: **Required.** Ensemble to manage (ID, path, or ".")
- `--format {yaml,json}`: File format for generation (default: yaml)

Parameter File Structure: Parameters are organized by operation type and mode/variant:

```
hmc:
  tepid:
    xml_params: "StartTrajectory=0 Trajectories=100 MDsteps=2"
    job_params: "cfg_max=100 time_limit=12:00:00 nodes=1"
  continue:
    xml_params: "Trajectories=50 MDsteps=2"
    job_params: "cfg_max=500 time_limit=6:00:00"

smearing:
  stout8:
    params: "nsteps=8 rho=0.1"
    job_params: "time_limit=2:00:00"

meson_2pt:
  default:
    params: "source_type=point sink_type=point"
    job_params: "time_limit=4:00:00"
```

Usage with Other Commands: Use `--use-default-params` flag in script commands to load parameters from the file. CLI parameters override default parameters.

Examples:

```
# Generate complete template file with all operation types
$ mdwf_db default_params generate -e 1
Generated configuration template: /private/tmp/.../mdwf_default_params.yaml
Edit this file to customize parameters for your ensemble

# View all available parameter configurations
$ mdwf_db default_params show -e 1
Configuration file: /private/tmp/.../mdwf_default_params.yaml
Available operation configurations:

  hmc:
    tepid:
      xml_params: StartTrajectory=0 Trajectories=100 MDsteps=2 trajL=0.75 MetropolisTest=false
      job_params: cfg_max=100 time_limit=12:00:00 nodes=1 constraint=gpu cpus_per_task=32
    continue:
      xml_params: Trajectories=50 MDsteps=2 trajL=0.75 MetropolisTest=true
      job_params: cfg_max=500 time_limit=6:00:00 nodes=1 constraint=gpu cpus_per_task=32
    reseed:
      xml_params: StartTrajectory=0 Trajectories=200 MDsteps=2 trajL=0.75 MetropolisTest=true
      job_params: cfg_max=200 time_limit=12:00:00 nodes=1 constraint=gpu cpus_per_task=32

  smearing:
    stout8:
      params: nsteps=8 rho=0.1
      job_params: time_limit=2:00:00 nodes=1
    stout4:
      params: nsteps=4 rho=0.15
      job_params: time_limit=1:30:00 nodes=1

  meson_2pt:
    default:
      params: source_type=point sink_type=point
      job_params: time_limit=4:00:00 nodes=1
    wall:
      params: source_type=wall sink_type=point
      job_params: time_limit=6:00:00 nodes=2

  wit:
    default:
      params: mass_preset=physical
      job_params: time_limit=8:00:00 nodes=2

  mres:
    default:
      params: mass_preset=physical
      job_params: time_limit=6:00:00 nodes=1

  wflow:
    default:
      params: flow_time=0.1
      job_params: time_limit=2:00:00 nodes=1

# Show updated parameters after saving new ones
$ mdwf_db default_params show -e 1
Configuration file: /private/tmp/.../mdwf_default_params.yaml
Available operation configurations:

  hmc:
    tepid:
      xml_params: MDsteps=6 trajL=0.5
      job_params: cfg_max=25 time_limit=3:00:00
    continue:
      xml_params: Trajectories=50 MDsteps=2 trajL=0.75 MetropolisTest=true
      job_params: cfg_max=500 time_limit=6:00:00 nodes=1 constraint=gpu cpus_per_task=32
    ...

# Use parameters with CLI overrides
$ mdwf_db hmc-script -e 1 -a m2986 -m continue --use-default-params -j "nodes=2"
Loaded HMC continue default parameters from .../mdwf_default_params.yaml
$ mdwf_db smear-script -e 1 --use-default-params --params-variant stout8
Loaded smearing.stout8 default parameters from .../mdwf_default_params.yaml
```

3 Default Parameter System

The MDWF system includes a comprehensive default parameter management system for reproducible workflows. This system allows you to save ”recipes” of parameters that work well for specific ensembles and reuse them across different operations.

3.1 Default Parameter Files

Default parameters are stored in `mdwf_default_params.yaml` files within each ensemble directory:

```yaml
---
hmc:
  tepid:
    xml_params: "StartTrajectory=0 Trajectories=100 MDsteps=2 trajL=0.75"
    job_params: "cfg_max=100 time_limit=12:00:00 nodes=1 constraint=gpu"
  continue:
    xml_params: "Trajectories=50 MDsteps=2 trajL=0.75"
    job_params: "cfg_max=500 time_limit=6:00:00"

smearing:
  stout8:
    params: "nsteps=8 rho=0.1"
    job_params: "time_limit=2:00:00"

meson_2pt:
  default:
    params: "source_type=point sink_type=point"
    job_params: "time_limit=4:00:00"
```

3.2 Parameter Precedence

When using default parameters, the system follows this precedence order:

1. **CLI parameters** (highest priority) - explicitly specified on command line
2. **Default parameters** - loaded from ensemble’s `mdwf_default_params.yaml`
3. **Command defaults** - built-in defaults for each command

This means you can:

- Use `--use-default-params` to load all parameters from the file
- Override specific parameters with CLI options
- Mix default and CLI parameters for flexible workflows

3.3 CLI Parameter Handling

The system provides several options for managing default parameters:

- `--use-default-params`: Load parameters from ensemble’s default parameter file
- `--save-default-params`: Save current command parameters to the default file
- `--params-variant PARAMS_VARIANT`: Use a specific parameter variant (e.g., ”tepid”, ”continue”)
- `--save-params-as SAVE_PARAMS_AS`: Save current parameters under a custom variant name

Examples:

```
# Load default parameters for HMC continue mode
$ mdwf_db hmc-script -e 1 -a m2986 -m continue --use-default-params

# Save current parameters as new variant
$ mdwf_db hmc-script -e 1 -a m2986 -m continue \
    -x "Trajectories=200 MDsteps=4" \
    -j "time_limit=24:00:00" \
    --save-params-as long_run

# Use new variant
mdwf_db smear-script -e 1 --use-default-params --params-variant stout12
```