
contamxpy Documentation

Release 0.0.8

W. Stuart Dols, Brian J. Polidoro

Sep 21, 2023

Contents

1	Introduction	2
2	cxLib	2
3	Zone	11
4	Path	12
5	Simple Air-handling Systems (AHS)	13
6	DuctTerminal	14
7	DuctJunction	15
8	DuctLeak	16
9	Control	16
10	Callback Function	17
11	Example Driver Programs	17
12	NIST Developer Notes	42
	Index	46

1 Introduction

This is the documentation for the `contamxpy` module which provides Python bindings to the *ContamX* API `contamx-lib`.

The `contamxpy` module consists mainly of the `contamxpy.cxLib` (page 2) class that provides the wrapper to `contamx-lib`. This documentation provides the details of the Python API which can be used to run CONTAM simulations on existing CONTAM projects, i.e., *.prj* files. `contamx-lib` provides a thread-safe API to the CONTAM simulation engine, ContamX. A simulation *state* is associated with each instance of `contamxpy.cxLib` (page 2) upon instantiation for a particular PRJ file.

1.1 To utilize `contamxpy` :

Install `contamxpy` from PyPI

```
pip install contamxpy
```

Import `cxLib` wrapper class into user-defined driver module

```
from contamxpy import cxLib
```

Note: `contamx-lib` will allow for co-simulation to be performed according to the following:

- steady airflow with no contaminants
- steady airflow with steady or transient contaminants
- transient airflow with transient contaminants
- Only the Default Solver (Implicit Euler) is allowed for contaminant calculations

Co-simulation is not allowed for the following:

- Airflow simulations: Duct Balance, Building Airflow Test, or Building Pressurization Test
 - Contaminant simulations: Cyclic, Short Time Step, or CVODE
 - CFD analysis
-

See also:

Example Driver Programs (page 17)

2 `cxLib`

class `contamxpy.cxLib` (*prj_file_path*, *wp_mode*=0, *cb_option*=False, *wth_init_function*=None)

`contamxpy.cxLib` (page 2) provides a wrapper around `contamx-lib` that provides a thread-safe API to the CONTAM simulation engine, ContamX. A simulation *state* is associated with each instance of `contamxpy.cxLib` (page 2) upon instantiation for a particular CONTAM PRJ file.

prj_file_path

Provide path to associated PRJ file. It will be converted to a *prjlib* path to set the `contamxpy.cxLib.prjPath` instance attribute.

Type

string

wp_mode

Set wind pressure calculation method `cxLib.wp_mode` (page 3)

- **0 = CONTAM computes wind pressures using WTH-like messages and ambient Mass Fractions using CTM-like messages, i.e., setAmbtXXX() messages.**
- **1 = Use envelope-related functions of the contam-x-cosim API to set wind pressure of individual envelope flow paths (default), i.e., setEnvelopeXXX().**

Type

int {0, 1}, optional

cb_option

Set callback option `cxLib.cb_option` (page 3) for `contamxpy.prjDataReadyFcnP()` (page 17)

- *False* = `contamx-lib` will not execute callback function.
- *True* = `contamx-lib` will execute callback function.

Type

bool, optional

wth_init_function

Set user-defined function `cxLib.wth_init_function()` (page 3) via this parameter. It will be called by `contamxpy.prjDataReadyFcnP()` (page 17), for example, to set initial ambient conditions prior to running steady-state initialization calculations.

Type

(function name), optional

prj_file_path

pathlib form of CONTAM PRJ file with which this instance of `contamxpy.cxLib` (page 2) is associated.

This attribute is set via the *prj_file_path* parameter of the constructor.

verbose: int

Logging level {0 = none (default), 1 = medium, 2 = high}.

Set via the `cxLib.setVerbosity()` (page 5).

nContaminants: int

(Read-only) Number of *Contaminants* in the PRJ. Defaults to *-1*. See `cxLib.contaminants` (page 3).

Set via the `contamxpy.prjDataReadyFcnP()` (page 17) callback function if `cxLib.cb_option` (page 3) is *True*.

contaminants: list

(Read-only) List of contaminant names (*list* of *str*).

Set via the `contamxpy.prjDataReadyFcnP()` (page 17) callback function if `cxLib.cb_option` (page 3) is *True*.

nZones: int

(Read-only) Number of *Zones* in the PRJ. Defaults to *-1*. See `cxLib.zones` (page 4).

Set via the `contamxpy.prjDataReadyFcnP()` (page 17) callback function if `cxLib.cb_option` (page 3) is *True*.

zones: list

(Read-only) List of `contamxpy.Zone` (page 11) objects.

Set via the `contamxpy.prjDataReadyFcnP()` (page 17) callback function if `cxLib.cb_option` (page 3) is *True*.

nPaths: int

(Read-only) Number of *Paths* in the PRJ. Defaults to *-1*. See `cxLib.paths` (page 4).

Set via the `contamxpy.prjDataReadyFcnP()` (page 17) callback function if `cxLib.cb_option` (page 3) is *True*.

paths: list

(Read-only) List of `contamxpy.Path` (page 12) objects.

Set via the `contamxpy.prjDataReadyFcnP()` (page 17) callback function if `cxLib.cb_option` (page 3) is *True*.

nEnvPaths: int

(Read-only) Number of *Paths* in the PRJ connected to ambient. Defaults to *-1*.

Set via the `contamxpy.prjDataReadyFcnP()` (page 17) callback function if `cxLib.cb_option` (page 3) is *True*.

envPaths: list

(Read-only) List of `contamxpy.Path` (page 12) objects with connections to Ambient.

Set via the `contamxpy.prjDataReadyFcnP()` (page 17) callback function if `cxLib.cb_option` (page 3) is *True*.

nAhs: int

(Read-only) Number of *Simple AHSs* in the PRJ. Defaults to *-1*.

Set via the `contamxpy.prjDataReadyFcnP()` (page 17) callback function if `cxLib.cb_option` (page 3) is *True*.

AHSs: list

(Read-only) List of `contamxpy.AHS` (page 13) objects.

Set via the `contamxpy.prjDataReadyFcnP()` (page 17) callback function if `cxLib.cb_option` (page 3) is *True*.

nInputControls: int

(Read-only) Number of *Input Controls*, i.e., *Constant* type controls in the PRJ. Defaults to *-1*.

Set via the `contamxpy.prjDataReadyFcnP()` (page 17) callback function if `cxLib.cb_option` (page 3) is *True*.

nOutputControls: int

(Read-only) Number of *Output Controls*, i.e., *Signal split* type controls in the PRJ. Defaults to *-1*.

Set via the `contamxpy.prjDataReadyFcnP()` (page 17) callback function if `cxLib.cb_option` (page 3) is *True*.

inputControls: list

(Read-only) List of *contamxpy.InputControl* (page 16) objects.

Set via the *contamxpy.prjDataReadyFcnP()* (page 17) callback function if *cxLib.cb_option* (page 3) is *True*.

outputControls: list

(Read-only) List of *contamxpy.OutputControl* (page 17) objects.

Set via the *contamxpy.prjDataReadyFcnP()* (page 17) callback function if *cxLib.cb_option* (page 3) is *True*.

nDuctJunctions: int

(Read-only) Number of *Duct Junctions* in the PRJ. Defaults to *-1*.

Set via the *contamxpy.prjDataReadyFcnP()* (page 17) callback function if *cxLib.cb_option* (page 3) is *True*.

ductJunctions: list

(Read-only) List of *contamxpy.DuctJunction* (page 15) objects.

Set via the *contamxpy.prjDataReadyFcnP()* (page 17) callback function if *cxLib.cb_option* (page 3) is *True*.

nDuctTerminals: int

(Read-only) Number of *Duct Terminals* in the PRJ. Defaults to *-1*.

Set via the *contamxpy.prjDataReadyFcnP()* (page 17) callback function if *cxLib.cb_option* (page 3) is *True*.

ductTerminals: list

(Read-only) List of *contamxpy.DuctTerminal* (page 14) objects.

Set via the *contamxpy.prjDataReadyFcnP()* (page 17) callback function if *cxLib.cb_option* (page 3) is *True*.

nEnvTerminals: int

(Read-only) Number of *Duct Terminals* in the PRJ connected to ambient. Defaults to *-1*.

Set via the *contamxpy.prjDataReadyFcnP()* (page 17) callback function if *cxLib.cb_option* (page 3) is *True*.

envTerminals: list

(Read-only) List of *contamxpy.DuctTerminal* (page 14) objects with connections to Ambient.

Set via the *contamxpy.prjDataReadyFcnP()* (page 17) callback function if *cxLib.cb_option* (page 3) is *True*.

nDuctLeaks: int

(Read-only) Number of *Duct Leaks* in the PRJ. Defaults to *-1*.

Set via the *contamxpy.prjDataReadyFcnP()* (page 17) callback function if *cxLib.cb_option* (page 3) is *True*.

ductLeaks: list

(Read-only) List of *contamxpy.DuctLeak* (page 16) objects.

Set via the *contamxpy.prjDataReadyFcnP()* (page 17) callback function if *cxLib.cb_option* (page 3) is *True*.

setVerbosity (*level=0*)

Set logging level for instance of *cxLib* (page 2). See *cxLib.verbose* (page 3).

Parameters

level – *int* Logging level:

- 0 No logging
- 1 Minimal logging
- 2 Maximum logging

setupSimulation (*use_cosim=1*)

Setup the simulation including the option to run ContamX in the co-simulation mode. Calling *cxSetupSimulation()* with *use_cosim* set to 1 will initiate the simulation by reading the PRJ file, allocating simulation data, calling of the user-defined callback function if set to do so via *cxRegisterCallback_PrjDataReady()*, and running the steady state initialization.

Parameters

use_cosim – *int* Select ContamX run mode:

- 0 = run a CONTAM-only simulation,
- 1 = run ContamX in co-simulation mode.

Returns

0 indicating success, 1 indicating invalid settings for co-simulation.

Return type

int

getVersion ()**Returns**

The version of *contamx-lib*, i.e., the *ContamX* version, e.g., *3.4.1.4-64bit*.

Return type

str

getSimTimeStep ()**Returns**

Calculation time step in seconds (1 - 60)

Return type

int

getSimStartDate ()**Returns**

Start day of year of the simulation [1 - 365]

Return type

int

getSimEndDate ()**Returns**

End day of year of the simulation [1 - 365]

Return type

int

getSimStartTime ()

Returns

Start time of day the simulation in seconds [0 - 86400)

Return type

int

getSimEndTime ()

Returns

End time of day of the simulation in seconds [0 - 86400)

Return type

int

getCurrentDayOfYear ()

Returns

Current day of year of the simulation [1 - 365]

Return type

int

getCurrentTimeInSec ()

Returns

Current time of day of the simulation in seconds [0 - 86400)

Return type

int

doSimStep (step_forward=1)

Run next simulation time step.

Parameters

stepForward – *int* Currently only a value of 1 is allowed which will run the next time step.

endSimulation ()

This function must be called at the end of a co-simulation. This should only be called after all time steps of the co-simulation have been completed, i.e., after *cxLib.doSimStep ()* (page 7) has been called for the values obtained for the ending date and time of the simulation.

setAmbtTemperature (temperature)

Set outdoor temperature [K] (temperature >= 0).

setAmbtPressure (pressure)

Set outdoor pressure [Pa] (pressure >= 0).

setAmbtWindSpeed (wind_speed)

Set wind speed [m/s] (WS >= 0).

setAmbtWindDirection (wind_direction)

Set wind direction [deg] (0 <= wind_direction <= 360).

setAmbtMassFraction (contaminant_nr, mass_fraction)

Set outdoor mass fraction [kg_cont/kg_air] (0.0 <= mass_fraction <= 1.0).

setEnvelopeWP (env_index, wind_pressure)

Set the wind pressure of an envelope flow path. This is akin to using a WPC file.

Parameters

- **env_index** – *int* Index set by ContamX. See [contamxpy.Path.envIndex](#) (page 13).
- **wind_pressure** – *float* Wind pressure value [Pa].

setEnvelopeMF (*env_index, contaminant_num, mass_fraction*)

Set the mass fraction at an envelope flow path. This is akin to using a WPC file.

Parameters

- **env_index** – *int* Index set by ContamX. See [contamxpy.Path.envIndex](#) (page 13).
- **contaminant_num** – *int* Contaminant number, e.g., assigned by ContamW
- **mass_fraction** – *float* Mass fraction [kg_cont/kg_air].

setZoneAddMass (*zone_num, contaminant_num, mass*)

Add mass of contaminant to zone.

Parameters

- **zone_num** – *int* Zone number to which mass should be added. Zone numbers range from 1 to :py:attr:contamxpy.cxLib.nZones and are assigned by ContamW.
- **contaminant_num** – *int* Number of contaminant for which mass is to be added to zone. Contaminant numbers range from 0 to [contamxpy.cxLib.nContaminants](#) (page 3) -1 and are assigned by ContamW.
- **mass** – *float* Amount of mass [kg] to add to the zone (≥ 0.0).

Returns

0 indicating success, > 0 indicating error occurred.

Return type

int

setZoneTemperature (*zone_num, temperature*)

Set zone temperature.

Parameters

- **zone_num** – *int* Zone number for which temperature should be set. Zone numbers range from 1 to [contamxpy.cxLib.nZones](#) (page 3) and are assigned by ContamW.
- **temperature** – *float* Temperature [K] to set (≥ 0.0).

Return type

int

setJunctionTemperature (*junction_num, temperature*)

Set Duct Junction temperature.

Parameters

- **junction_num** – *int* Junction number for which temperature should be set. Zone numbers range from 1 to [contamxpy.cxLib.nDuctJunctions](#) (page 5) and are assigned by ContamW.
- **temperature** – *float* Temperature [K] to set (≥ 0.0).

Return type

int

setAhsSupplyReturnFlow (*path_num*, *flow*)

Set airflow rate of Simple AHS Supply or Return flow path.

Parameters

- **path_num** – *int* Airflow path number for which flow should be set. Path numbers range from 1 to `contamxpy.cxLib.nPaths` (page 4) and are assigned by *ContamW*.
- **flow** – *float* Mass flow rate [kg/s] to set (≥ 0.0).

setAhsPercentOa (*ahs_num*, *oa_fraction*)

Set Outdoor Air fraction for Simple AHS.

Parameters

- **ahs_num** – *int* Air-handling System number for which OA fraction is to be set. AHS numbers range from 1 to `contamxpy.cxLib.nAhs` (page 4) and are typically assigned by *ContamW*.
- **oa_fraction** – *float* Fraction of outdoor air ($0 \leq \text{oa_fraction} \leq 1.0$)

getZoneMassFraction (*zone_num*, *contaminant_num*)

Get the mass fraction of a zone for selected contaminant.

Parameters

- **zone_num** – *int* Zone number for which mass fraction should be obtained. Zone numbers range from 1 to `contamxpy.cxLib.nZones` (page 3) and are assigned by *ContamW*.
- **contaminant_num** – *int* Contaminant number for which mass fraction should be obtained. Contaminant numbers range from 0 to `contamxpy.cxLib.nContaminants` (page 3) -1 and are assigned by *ContamW*.

Returns

Mass fraction [kg_cont/kg_air].

Return type

float

getEnvelopeExfil (*envIndex*, *contaminant_num*)

Get the mass of contaminant exfiltrating from an envelope airflow path.

Parameters

- **envIndex** – *int* Envelope index of airflow path for which to obtain exfiltration.
- **contaminant_num** – *int* Contaminant number for which exfiltrating mass should be obtained. Contaminant numbers range from 0 to `contamxpy.cxLib.nContaminants` (page 3) -1 and are assigned by *ContamW*.

Returns

Mass of contaminant [kg_cont].

Return type

float

See also:

`contamxpy.cxLib.paths` (page 4) `contamxpy.Path.envIndex` (page 13)

getPathFlow (*path_num*)

Get airflow rate through path.

Parameters

path_num – *int* Number of flow path for which to obtain airflow rate.

Returns

Array of two-way flows through the path: Flow0 and Flow1 [kg/s]

Return type

float[]

getDuctTerminalFlow (*terminal_num*)

Get airflow rate through duct terminal.

Parameters

terminal_num – *int* Number of duct terminal (1 - *cxLib.ductTerminals* (page 5)) in the *contamxpy.cxLib.ductTerminals* (page 5) list for which to obtain the airflow rate.

Returns

Positive airflow out of the terminal into the zone or Negative airflow into the terminal from the zone [kg/s]

Return type

float

getDuctLeakFlow (*leak_num*)

Get airflow rate through duct leak.

Parameters

leak_num – *int* Number of duct leak (1 -> *cxLib.nDuctLeaks* (page 5)) in the *cxLib.ductLeaks* (page 5) list for which to obtain the airflow rate.

Returns

Positive airflow out of the junction into the zone or Negative airflow into the junction from the zone [kg/s]

Return type

float

setInputControlValue (*i*, *val*)

Set value of Input control type (CT_SET).

Parameters

- **i** – *int* Index of Input control (1 - *cxLib.nInputControls* (page 4)) in the *cxLib.inputControls* (page 4) list for which to set the value.
- **val** – *float* Value to set.

getOutputControlValue (*i*)

Get Output control value (CT_PASS).

Parameters

i – *int* Index of Output control (1 - *cxLib.nOutputControls* (page 4)) in the *contamxpy.cxLib.outputControls* (page 5) list for which to get the value.

Returns

Value of control output signal.

Return type

float

Todo: Initial values all seem to be 0.0 even though LOG file results appear to reflect steady-state simulation results.

3 Zone

class `contamxpy.Zone` (*nr, name, flags, volume, level_nr, level_name*)

Instances of *Zone* (page 11) are created via the *prjDataReadyFcnP()* (page 17) callback function. If the callback function is called as indicated via the *cxLib* (page 2) constructor, then *Zones* will be accessible via the *cxLib.zones* (page 4) list.

nr

Zone number, typically assigned by *ContamW*.

Type

int

name

Zone name, typically assigned by *ContamW*.

Implicit supply and return zones of simple air-handling systems (SAHS) will have either “(Sup)” or “(Ret)” appended to the name of the SAHS with which they are associated.

Type

str

flags

- 0x01 variable Pressure node
- 0x02 variable Mass fraction node
- 0x04 variable Temperature node
- 0x08 implicit Simple AHS node, i.e., supply or return
- 0x10 unconditioned node
- 0x20 node volume > 0, i.e., not massless

Type

int

volume

Zone volume [m³]

Type

real

level_nr

Level number on which this Zone is located.

Type

int

level_name

Name of level on which this Zone is located.

Type

str

4 Path

Todo: (BJP) *contamxpy.Path* (page 12) Are wind and WPC flags relevant to co-simulation, or can they be ignored?

class *contamxpy.Path* (*nr, flags, from_zone, to_zone, ahs_nr, X, Y, Z, envIndex*)

Instances of *Path* are created via the *contamxpy.prjDataReadyFcnP()* (page 17) callback function. If the callback function is called as indicated via the *cxLib* (page 2) constructor, then *Paths* will be accessible via the *cxLib.paths* (page 4) list.

nr

Path number, typically assigned by *ContamW*.

Type

int

flags

Flags used to indicate airflow path properties. Not all of these flags are relevant to co-simulation, e.g., the WPC-related flags can be ignored.

Airflow path flag values:

- 0x0001 possible wind pressure
- 0x0002 path uses WPC file pressure
- 0x0004 path uses WPC file contaminants
- 0x0008 Simple air-handling system (SAHS) supply or return path
- 0x0010 SAHS recirculation flow path
- 0x0020 SAHS outside air flow path
- 0x0040 SAHS exhaust flow path
- 0x0080 path has associate pressure limits
- 0x0100 path has associate flow limits
- 0x0200 path has associated constant airflow element
- 0x0400 junction leak path

Type

int

from_zone

Number of *From* zone used to indicate positive flow direction: *from_zone* -> *to_zone*. Zone 0 indicates *Ambient*.

Type

int

to_zone

Number of *To* zone used to indicate positive flow direction: from_zone -> to_zone. Zone 0 indicates *Ambient*.

Type
int

ahs_nr

Number of the Simple AHS associated with this Path if represents either a ventilation system *supply* or *return*.

Type
int

x

X-coordinate

Type
real

y

Y-coordinate

Type
real

z

Z-coordinate

Type
real

envIndex

Index identifies order of specifying values in WPC file and used to reference specific airflow paths located in ambient when using the `contamx-lib` API. Zero for airflow paths not located in ambient.

Type
int

5 Simple Air-handling Systems (AHS)

class `contamxpy.AHS` (*nr, name, zone_ret, zone_sup, path_rec, path_oa, path_exh*)

Instances of AHS are created via the `contamxpy.prjDataReadyFcnP()` (page 17) callback function.

If the callback function is called as indicated via the `cxLib` (page 2) constructor, then *AHSs* will be accessible via the `cxLib.AHSs` (page 4) list.

nr

Simple AHS number, typically assigned by *ContamW*.

Type
int

name

Simple AHS name, typically assigned by *ContamW*.

Type
str

zone_ret

Number of the implicit Return zone associated with this AHS.

Type
int

zone_sup

Number of the implicit Supply zone associated with this AHS.

Type
int

path_rec

Number of the implicit Recirculation flow path associated with this AHS.

Type
int

path_oa

Number of the implicit Outdoor air intake flow path associated with this AHS.

Type
int

path_exh

Number of the implicit Exhaust flow path associated with this AHS.

Type
int

6 DuctTerminal

class `contamxpy.DuctTerminal` (*nr, flags, X, Y, Z, relHt, to_zone, envIndex*)

Instances of `DuctTerminal` are created via the `contamxpy.prjDataReadyFcnP()` (page 17) callback function.

If the callback function is called as indicated via the `cxLib` (page 2) constructor, then *DuctTerminals* will be accessible via the `cxLib.ductTerminals` (page 5) list.

nr

Duct Terminal number, typically assigned by *ContamW*.

Type
int

flags

Airflow path flag values:

- 0x0001 terminal has wind pressure associated with it
- 0x0002 terminal uses WPC file pressure
- 0x0400 duct leak

Type
int

X
X-coordinate [m]
Type
real

Y
Y-coordinate [m]
Type
real

Z
Z-coordinate [m]
Type
real

relHt
Height relative to level on which the terminal is located [m]
Type
real

to_zone
Zone number in which the terminal is located. Positive flow: terminal -> *to_zone*. Zone 0 indicates *Ambient*.
Type
int

envIndex
Index identifies order of specifying values in WPC file and used to reference specific terminals located in ambient when using the `contamx-lib` API. Zero for terminals not located in ambient.
Type
int

7 DuctJunction

Todo: Document `contamxpy.DuctJunction` (page 15) flags.

Todo: `contamxpy.DuctJunction` (page 15) Why is there an `envIndex` property? ContamW does not provide for wind pressure to be specified for Duct Junctions. Coordinates are provided for Junctions in ContamW for use in 1D Zones. Is wind pressure accounted for at Leaks in Duct Junctions???

Todo: `contamxpy.DuctJunction` (page 15) All terminals are junctions, but not all junctions are terminals. Separate them into different lists using flags?

class `contamxpy.DuctJunction` (*nr, flags, containing_zone, envIndex*)

Instances of *DuctJunction* are created via the `contamxpy.prjDataReadyFcnP()` (page 17) callback function.

If the callback function is called as indicated via the `cxLib` (page 2) constructor, then *DuctJunctions* will be accessible via the `cxLib.ductJunctions` (page 5) list.

nr

Duct Junction number, typically assigned by *ContamW*.

Type

int

flags

Type

int

containing_zone

Zone number in which the junction is located. Zone 0 indicates *Ambient*.

Type

int

envIndex

Index identifies order of specifying values in WPC file and used to reference specific duct junctions located in ambient when using the `contamx-lib` API. Zero for duct junctions not located in ambient.

Type

int

8 DuctLeak

class `contamxpy.DuctLeak` (*nr, flags, X, Y, Z, relHt, to_zone, envIndex*)

Instances of *DuctLeak* are created via the `contamxpy.prjDataReadyFcnP()` (page 17) callback function.

If the callback function is called as indicated via the `cxLib` (page 2) constructor, then *DuctLeaks* will be accessible via the `cxLib.ductLeaks` (page 5) list.

`contamxpy.DuctLeak` (page 16) inherits `contamxpy.DuctTerminal` (page 14).

9 Control

Instances of *Controls* are created via the `contamxpy.prjDataReadyFcnP()` (page 17) callback function. If the callback function is called as indicated via the `cxLib` constructor, then *Controls* will be accessible via the `contamxpy.cxLib.inputControls` (page 4) and `contamxpy.cxLib.outputControls` (page 5) lists.

class `contamxpy.Control` (*nr, name*)

Abstract Base Class of `contamxpy.InputControl` (page 16) and `contamxpy.OutputControl` (page 17).

nr

Control node number, typically assigned by *ContamW*.

name

Name of control node, typically user-defined via *ContamW*.

class `contamxpy.InputControl` (*nr, name, type_str='CT_SET'*)

This class extends base class `contamxpy.Control` (page 16).


```

    typeStr = CT_SET

class contamxpy.OutputControl(nr, name, type_str='CT_PASS')
    This class extends base class contamxpy.Control (page 16).

    typeStr = CT_PASS

```

10 Callback Function

`contamxpy.prjDataReadyFcnP` (*state*, *handle*)

This function populates the list attributes associated with *cxLib* (page 2), e.g., *contamxpy.cxLib.contaminants* (page 3), *contamxpy.cxLib.zones* (page 4), and *contamxpy.cxLib.paths* (page 4).

NOTE: The function name must match that in the associated *contamx-build.py* module that defines the CFFI interface of *contamxpy*.

Parameters

- **state** – The *ContamXState* obtained from *contamx-lib* upon instantiation of a *cxLib* (page 2) object.
- **handle** – The CFFI handle to the current *cxLib* (page 2) instance.

11 Example Driver Programs

Several driver programs provide examples of the main functionality of the *contamxpy.py* module. Some are tailored to specific PRJ files in order to apply boundary conditions associated with those PRJs. The cases are described below, and the code is provided in the *DriverPrograms* (page 19) section below. These drivers also import the *cxRunSim.py* (page 39) and *cxResults.py* (page 40) modules to obtain and output results to text files.

11.1 WTH and CTM-like API Cases

These cases can be run via the *test_OneZoneWthCtm.py* driver module and will apply corresponding boundary conditions based on the PRJ file name provided on the command line. Each of these “-UseApi” PRJs have a corresponding non-API version to which they can be compared below.

Note: The `-verbose=2` command line option will provide detailed information related to the contents of the PRJ file as obtained via *contamxpy.cxLib* (page 2), and `> out.txt` redirects the output to a text file that can be viewed via a text editor.

- *test_OneZoneWthCtm-UseApi.prj*
- *test_OneZoneWthCtmStack-UseApi.prj*
- *valThreeZonesWthCtm-UseApi.prj*
- *testGetPrjInfo.prj*

This is a generic test case to demonstrate all API functions that are available for obtaining information about the contents of the PRJ that are relevant to utilizing the API via driver modules. Run this case with `-verbose=2` to show all possible PRJ info available via *contamxpy.cxLib* (page 2). Further, this case also includes Simple AHSs and Controls for which the *test_OneZoneWthCtm.py* module includes inputs to demonstrate setting AHS flows and outdoor air fraction and setting Input Control values.

Command line:

```
test_OneZoneWthCtm.py <PRJ File Name> --verbose=2 > out.txt
```

11.2 WTH and CTM-like non-API Cases

These cases are fully contained PRJ files that include references to WTH and CTM files within them. They can be run via *ContamW*, *ContamX*, or the generic driver program *test_cxcffi.py* (page 26). The driver program will simply run the PRJ through all of its time steps and output zone mass fractions to a text file along with other CONTAM-generated results files.

Command line:

```
test_cxcffi.py <PRJ File Path>
```

- test_OneZoneWthCtm.prj
 - test_OneZoneWthCtm.wth
 - test_OneZoneWthCtm.ctm
- test_OneZoneWthCtmStack.prj
 - test_OneZoneWthCtmStack.wth
 - test_OneZoneWthCtm.ctm
- valThreeZonesWthCtm.prj
 - valThreeZonesWthCtm.wth
 - valThreeZonesWthCtm.ctm

11.3 WPC-like API Cases

This case can be run via the *test_OneFloorWpcAddMf.py* (page 29) driver module and will apply boundary conditions similar to those provided via a WPC file. The corresponding non-API version should provide the same results as the API version.

Command line:

```
test_OneFloorWpcAddMf.py test_OneFloorWpcAddMf-UseApi.prj
```

11.4 Thread-safe Driver

contamxpy is thread safe, and the *test_Multirun.py* (page 33) driver demonstrates this capability by running multiple PRJ files provided in a list file using the Python *threading* module. **NOTE:** At the time of this release, *contamxpy* cannot be used with Python's multiprocessing capabilities due to pickling limitations of CFFI-based modules.

Command line:

```
test_MultiRun.py prjFile.lst -m1
```

11.5 Steady-State Case

While `contamx-lib` was originally developed to perform co-simulation for transient simulations, it can also be used to run PRJ files configured for steady-state airflow and/or contaminant analysis. This case can be run with the `test_OneZoneSS.py` (page 37) example driver program.

Command line:

```
test_OneZoneSS.py test_OneZoneSS-UseApi.prj
```

NOTE: `test_OneZoneSS-UseApi-BAD.prj` is provided to demonstrate a PRJ that has incorrect simulation parameters that `contamx-lib` should prevent from being executed. It is defined to perform a cyclic simulation.

There are several non-API versions with which this test can be compared. These are listed below with associated WTH and CTM files.

- `test_OneZoneSS.prj` - Steady-state airflow and contaminant transport simulation.
- `test_OneZoneSS-Transient.prj` - Equivalent transient simulation using WTH and CTM files.
 - `test_OneZoneSS-Transient.wth`
 - `test_OneZoneSS-Transient.ctm`

11.6 Driver Programs

`test_OneZoneWthCtm.py`

```
from contamxpy import cxLib
import cxResults as cxxr
import os, sys
from optparse import OptionParser
import typing
import numpy as np

## TEST CASES - use global variables
g_times = np.zeros(shape=1, dtype=int)      # sec
g_Tambt = np.zeros(shape=1, dtype=float)    # K
g_Pambt = np.zeros(shape=1, dtype=float)    # Pa
g_WSambt = np.zeros(shape=1, dtype=float)   # m/s
g_WDambt = np.zeros(shape=1, dtype=float)   # deg
g_MFambt = np.zeros(shape=1, dtype=float)   # kg/kg
g_Tzone = np.zeros(shape=1, dtype=float)    # K
g_PctOa = np.zeros(shape=1, dtype=float)    # 0.0 - 1.0
g_AhsFlow = np.zeros(shape=1, dtype=float)  # m3/h-m2
g_CtrlIn = np.zeros(shape=(1,2), dtype=float) # assumes only two sets of control data

def setWthCtmData( prjName ):
    dataSetNum = 1
    #
    #           0      6:00:00 14:00:00 15:00:00 23:00:00 24:00:00
    times_data = np.array([ 0,      21600,  50400,  57600,  82800,  86400])
    Tambt_data = np.array([ 273.15,  293.15,  293.15,  293.15,  293.15,  293.15])
    Pambt_data = np.array([ 101325,  101325,  101325,  101325,  101325,  101325])
    WSambt_data = np.array([ 0.0,    7.397,  0.0,    7.397,  0.0,    0.0])
    WDambt_data = np.array([ 0.0,    270.0,  0.0,    90.0,  0.0,    0.0])
    MFambt_data = np.array([ 0.0023254, 0.0023254, 0.0,    0.0,    0.0,    0.0])
    Tzone_data = np.array([ 283.15,  288.15,  293.15,  298.15,  290.15,  283.15])
    #
    #           C      10      15      20      25      17      10
```

(continues on next page)

(continued from previous page)

```
PctOa_data = np.array([ 0.25, 0.75, 0.0, 1.0, 1.0, 1.0])
AhsFlow_data= np.array([ 10.0, 5.0, 7.5, 10.0, 9.0, 8.0])
CtrlIn_data = np.array([
    [ 263.15, 268.15, 273.15, 278.15, 270.15, 263.15],
    # K for Tzone
    [ 100, 200, 300, 150, 0.0, 100 ]
    # Multiplier on Flow
])

dataSetNum = 1
if prjName == "test_OneZoneWthCtmStack-UseApi":
    dataSetNum = 2
    #
    times_data = np.array([ 0, 6:00:00 12:00:00 13:00:00 24:00:00
    21600, 43200, 46800, 86400])
    Tambt_data = np.array([ 273.15, 273.15, 293.15, 263.15, 263.15])
    Pambt_data = np.array([101325, 101325, 101325, 101325, 101325])
    WSambt_data = np.array([ 0.0, 0.0, 0.0, 0.0, 0.0])
    WDambt_data = np.array([ 0.0, 0.0, 0.0, 0.0, 0.0])
    MFambt_data = np.array([0.0023254, 0.0023254, 0.0, 0.0, 0.0])
    Tzone_data = np.array([ 293.15, 293.15, 293.15, 303.15, 303.15])
    # C 20 20 20 30 30
    PctOa_data = np.array([ 0.25, 0.75, 0.0, 1.0, 1.0])
    AhsFlow_data= np.array([ 10.0, 5.0, 7.5, 10.0, 8.0])
    CtrlIn_data = np.array([
        [ 293.15, 293.15, 293.15, 303.15, 303.15],
        [ 0.0, 0.0, 0.0, 0.0, 0.0]
    ])

elif prjName == "valThreeZonesWthCtm-UseApi":
    dataSetNum = 3
    times_data = np.array([ 0, 86400])
    Tambt_data = np.array([ 293.15, 293.15])
    Pambt_data = np.array([ 101325, 101325])
    WSambt_data = np.array([ 5.23, 5.23])
    WDambt_data = np.array([ 270, 270])
    MFambt_data = np.array([0.0023254, 0.0023254])
    Tzone_data = np.array([ 293.15, 293.15])
    PctOa_data = np.array([ 1.0, 1.0])
    AhsFlow_data= np.array([ 9.0, 9.0])
    CtrlIn_data = np.array([
        [ 293.15, 293.15],
        [ 0.0, 0.0 ]
    ])

n = len(times_data)
t = np.zeros(shape=n)
Ta = np.zeros(shape=n)
Pa = np.zeros(shape=n)
WS = np.zeros(shape=n)
WD = np.zeros(shape=n)
MF = np.zeros(shape=n)
Tz = np.zeros(shape=n)
OA = np.zeros(shape=n)
AF = np.zeros(shape=n)
CI = np.zeros(shape=(2,n))

for i in range(n):
```

(continues on next page)

```

        t[i] = times_data[i]
        Ta[i] = Tambt_data[i]
        Pa[i] = Pambt_data[i]
        WS[i] = WSambt_data[i]
        WD[i] = WDambt_data[i]
        MF[i] = MFambt_data[i]
        Tz[i] = Tzone_data[i]
        OA[i] = PctOa_data[i]
        AF[i] = AhsFlow_data[i]
    for ir in range(2):
        for ic in range(n):
            CI[ir,ic] = CtrlIn_data[ir,ic]

    print(f"\nUSING DATA SET {prjName} NUMBER {dataSetNum} !!!\n")
    print(f"time:\t{t}\nTambt:\t{Ta}\nPambt:\t{Pa}\nWSpeed:\t{WS}\nWDir:\t{WD}\n"
    ↪nMFambt:\t{MF}\nTzone:\t{Tz}\nPctOA:\t{OA}\nFAhs:\t{AF}\nCtrlIn:\t{CI}\n")
    '''
    for i in range(2):
        for j in range(n):
            print(f"{i},{j}={CI[i][j]}\t")
        print("\n")
    '''
    return t, Ta, Pa, WS, WD, MF, Tz, OA, AF, CI

#=====↪
↪setWthCtmInit() =====
# Set the initial conditions for the SetAmbt API test.
# This function is set as a parameter to instantiation of cxLib
# to be called by the contamxpy.prjDataReadyFcnP() in order to set
# ambient boundary conditions for steady-state initialization.
# NOTE: The parameter {cxl as cxLib} is passed through contamx-lib to
# contamxpy.prjDataReadyFcnP() which in turn passes it to this function
# to distinguish the instance of cxLib to be used.
#
def setWthCtmInit( cxl ):
    global g_times, g_Tambt, g_Pambt, g_WSambt, g_WDambt, g_MFambt, g_Tzone, g_PctOa, ↪
    ↪g_AhsFlow, g_CtrlIn
    ### DEBUG
    print(f"setWthCtmInit({cxl})")
    cxl.setAmbtPressure(g_Pambt[0])
    cxl.setAmbtWindSpeed(g_WSambt[0])
    cxl.setAmbtWindDirection(g_WDambt[0])
    cxl.setAmbtMassFraction(0, g_MFambt[0])
    cxl.setAmbtTemperature(g_Tambt[0])
    # Vary Tzone by iz index to show clear differences when plotted.
    for iz in range(cxl.nZones):
        cxl.setZoneTemperature(iz+1, g_Tzone[0]+iz)
    for ij in range(cxl.nDuctJunctions):
        cxl.setJunctionTemperature(ij+1, g_Tzone[0]+ij)
    fOA = g_PctOa[0]
    for ia in range(cxl.nAhs):
        cxl.setAhsPercentOa(ia+1, fOA)
        fOA *= 0.95
    fFlow = g_AhsFlow[0]
    for ih in range(cxl.nAhs):
        setAhsFlows(cxl, ih, fFlow)
        #fFlow *= 0.95

```

(continues on next page)

```

for ic in range(cxl.nInputControls):
    j = ic % 2
    cxl.setInputControlValue(ic+1, g_CtrlIn[j][0])

#####
↪ setWthCtm() =====
# Update conditions when appropriate.
def setWthCtm( cxl, date, time, step, index ) -> int:
    # Check if it's time to change to the next set of Ambient data.
    global g_times, g_Tambt, g_Pambt, g_WSambt, g_WDambt, g_MFambt, g_Tzone, g_PctOa, ↪
    ↪ g_AhsFlow, g_CtrlIn
    if index >= len(g_times):
        return index
    if (time == g_times[index]):
        ###print(f"setWthCtm({date} {time})")
        cxl.setAmbtTemperature(g_Tambt[index])
        cxl.setAmbtPressure(g_Pambt[index])
        cxl.setAmbtWindSpeed(g_WSambt[index])
        cxl.setAmbtWindDirection(g_WDambt[index])
        cxl.setAmbtMassFraction(0, g_MFambt[index])
        # Vary Tzone by iz index to show clear differences when plotted.
        for iz in range(cxl.nZones):
            cxl.setZoneTemperature(iz+1, g_Tzone[index]+iz)
        for ij in range(cxl.nDuctJunctions):
            cxl.setJunctionTemperature(ij+1, g_Tzone[index]+ij)
        fOA = g_PctOa[index]
        for ia in range(cxl.nAhs):
            cxl.setAhsPercentOa(ia+1, fOA)
            fOA *= 0.95
        fFlow = g_AhsFlow[index]
        for ih in range(cxl.nAhs):
            setAhsFlows(cxl, ih, fFlow)
            #fFlow *= 0.95
        for ic in range(cxl.nInputControls):
            j = ic % 2
            cxl.setInputControlValue(ic+1, g_CtrlIn[j][index])
        index = index + 1
    return index

#####
↪ setAhsFlows() =====
def setAhsFlows(cxl : cxLib, ahsIndex, val):
    ahs = cxl.AHSs[ahsIndex]
    ### print(f"setAhsFlows({ahs.name})")
    # Set flow per zone floor area m3/h-m2
    Hf = 3.0 # assume h=3
    pathTypeStr = "SUPPLY"
    for path in ahs.supply_points:
        Zone = cxl.zones[path.to_zone-1]
        Vzone = Zone.volume # m3
        Azone = Vzone / Hf # m2
        Vdot = val * Azone # m3/h-m2 * m2 = m3/h
        Mdot = Vdot * 1.2041 / 3600.0 # kg/s
        ### print(f"\t{pathTypeStr}\tp{path.nr}/z{path.to_zone}:\t{val:.2f} m3/h-m2\t
        ↪ {Vzone:.2f} m3\t({Azone} m2)\t{Vdot:.2f} m3/h\t{Mdot:.5f} kg/s ")
        cxl.setAhsSupplyReturnFlow(path.nr, Mdot)
    pathTypeStr = "RETURN"

```

(continues on next page)

```

for path in ahs.return_points:
    Zone = cxl.zones[path.from_zone-1]
    Vzone = Zone.volume          # m3
    Azone = Vzone / Hf           # m2
    Vdot = val * Azone           # m3/h-m2 * m2 = m3/h
    Mdot = Vdot * 1.2041 / 3600.0 # kg/s
    ### print(f"\t{pathTypeStr}\tp{path.nr}/z{path.from_zone}:\t{val:.2f} m3/h-m2\
    ↪t{Vzone:.2f} m3\t({Azone} m2)\t{Vdot:.2f} m3/h\t{Mdot:.5f} kg/s ")
    cxl.setAhsSupplyReturnFlow(path.nr, Mdot)

↪#=====
↪main() =====
def main():
    global g_times, g_Tambt, g_Pambt, g_WSambt, g_WDambt, g_MFambt, g_Tzone, g_PctOa, ↪
    ↪g_AhsFlow, g_CtrlIn

    #----- Manage option parser
    parser = OptionParser(usage="%prog [options] arg1\n\targ1=PRJ filename\n")
    parser.set_defaults(verbose=0)
    parser.add_option("-v", "--verbose", action="store", dest="verbose", type="int", ↪
    ↪default=0,
                                help="define verbose output level: 0=Min, 1=Medium, 2=Maximum.
    ↪")
    (options, args) = parser.parse_args()

    #----- Process command line options -v
    verbose = options.verbose

    if len(args) != 1:
        parser.error("Need one argument:\n arg1 = PRJ file.")
        return
    else:
        # Get PRJ file name
        prjPath = args[0]

    if ( not os.path.exists(prjPath) ):
        print("ERROR: PRJ file not found.")
        return

    root, ext = os.path.splitext(prjPath)
    prjName = os.path.basename(root)

    msg_cmd = "Running test_OneZoneWthCtm.py: arg1 = " + args[0] + " " + str(options)
    print(msg_cmd, "\n")

    #----- Initialize Data Set for Boundary Conditions and Controls.
    #
    print(f"\n=====DataSetName = {prjName}\n=====")
    g_times, g_Tambt, g_Pambt, g_WSambt, g_WDambt, g_MFambt, g_Tzone, g_PctOa, g_
    ↪AhsFlow, g_CtrlIn = setWthCtmData(prjName)

    if verbose > 1:
        print(f"cxLib attributes =>\n{chr(10).join(map(str, dir(cxLib)))}\n")

    #----- Initialize contamx-lib object w/ wp_mode and cb_option.
    #         wp_mode = 0 => use wind pressure profiles, WTH and CTM files or associated ↪

```

(continues on next page)

```

↪API calls.
#         cb_option = True => set callback function to get PRJ INFO from the
↪ContamXState.
myPrj = cxLib(prjPath, 0, True, setWthCtmInit)

myPrj.setVerbosity(verbose)
if verbose > 1:
    print(f"BEFORE setupSimulation() \n\t nCtms={myPrj.nContaminants} \n\t nZones=
↪{myPrj.nZones} \n\t nPaths={myPrj.nPaths} \n" )

#----- Query State for Version info
verStr = myPrj.getVersion()
if verbose >= 0:
    print(f"getVersion() returned {verStr}.")

#----- Initialize Simulation -----
#-----
myPrj.setupSimulation(1)

#----- Initialize result files
fResMfList = []
fResEnvExfilList = []
root, ext = os.path.splitext(prjPath)
fNameResMf = root + "_Mf"
fNameResFlow = root + "_Flow.txt"
fNameResTotalExfil = root + "_Exfil.txt"
fNameResControl = root + "_Control.txt"
fResFlow = open(fNameResFlow, "w")
fResExfil = open(fNameResTotalExfil, "w")
for ic in range(myPrj.nContaminants):
    fName = root + "_Mf_" + myPrj.contaminants[ic] + ".txt"
    file = open(fName, "w")
    fResMfList.append(file)
    fName = root + "_Exfil_" + myPrj.contaminants[ic] + ".txt"
    file = open(fName, "w")
    fResEnvExfilList.append(file)
totalEnvExfil = [0.0] * myPrj.nContaminants
if (myPrj.nOutputControls > 0):
    fResControl = open(fNameResControl, "w")

#----- Get simulation run info
dayStart = myPrj.getSimStartDate()
dayEnd = myPrj.getSimEndDate()
secStart = myPrj.getSimStartTime()
secEnd = myPrj.getSimEndTime()
tStep = myPrj.getSimTimeStep()

#----- Calculate the simulation duration in seconds and total time steps
simBegin = (dayStart - 1) * 86400 + secStart
simEnd = (dayEnd - 1) * 86400 + secEnd
if (simBegin < simEnd):
    simDuration = simEnd - simBegin
else:
    simDuration = 365 * 86400 - simEnd + simBegin
numTimeSteps = int(simDuration / tStep)

```



```

#----- Get the current date/time after initial steady state simulation
currentDate = myPrj.getCurrentDayOfYear()
currentTime = myPrj.getCurrentTimeInSec()
if verbose > 0:
    print(f"Sim days = {dayStart}:{dayEnd}")
    print(f"Sim times = {secStart}:{secEnd}")
    print(f"Sim time step = {tStep}")
    print(f"Number of steps = {numTimeSteps}")

#----- Output initial results.
##cxr.printZoneMf(myPrj, currentDate, currentTime, myPrj.nZones, myPrj.
↪nContaminants)\
# Write headers
for ic in range(myPrj.nContaminants):
    cxr.writeMfZones(fResMfList[ic], True, myPrj, currentDate, currentTime, ic)
    cxr.writeEnvExfil(fResEnvExfilList[ic], True, myPrj, -1, -1, -1)
cxr.writeAirflowRates(fResFlow, True, myPrj, -1, -1)
if ( myPrj.nOutputControls > 0 ):
    cxr.writeControls(fResControl, True, myPrj, -1, -1)

# Write initial values
for ic in range(myPrj.nContaminants):
    cxr.writeMfZones(fResMfList[ic], False, myPrj, currentDate, currentTime, ic)
    cxr.writeAirflowRates(fResFlow, False, myPrj, currentDate, currentTime)
if ( myPrj.nOutputControls > 0 ):
    cxr.writeControls(fResControl, False, myPrj, currentDate, currentTime)

#-----
#----- Run Transient Simulation -----
#-----

wthIndex = 1

for i in range(numTimeSteps):
    #-----
    #----- Tasks to perform BEFORE current time step.
    #-----
    wthIndex = setWthCtm( myPrj, currentDate, currentTime, tStep, wthIndex)

    #-----
    # Run next time step.
    #-----
    myPrj.doSimStep(1)

    #-----
    #----- Tasks to perform AFTER current time step.
    #-----
    currentDate = myPrj.getCurrentDayOfYear()
    currentTime = myPrj.getCurrentTimeInSec()

    #----- Output results of time step just performed.
    for ic in range(myPrj.nContaminants):
        cxr.writeMfZones(fResMfList[ic], False, myPrj, currentDate, currentTime, ↪
↪ic)
        cxr.calcEnvExfil(myPrj, totalEnvExfil)
        for ic in range(myPrj.nContaminants):
            cxr.writeEnvExfil(fResEnvExfilList[ic], False, myPrj, currentDate, ↪
↪currentTime, ic)

```

(continues on next page)

(continued from previous page)

```
cxr.writeAirflowRates(fResFlow, False, myPrj, currentDate, currentTime)
if( myPrj.nOutputControls > 0 ):
    cxr.writeControls(fResControl, False, myPrj, currentDate, currentTime)

#----- End of simulation loop

#----- Output total envelope exfiltration.
#      Should be the same as CSM file.
print(f"totalEnvExfil[] final:\n\t{myPrj.contaminants}\n\t{totalEnvExfil}")
for i in range(myPrj.nContaminants):
    fResExfil.write(f"{myPrj.contaminants[i]}\t")
fResExfil.write("\n")
for i in range(myPrj.nContaminants):
    fResExfil.write(f"{totalEnvExfil[i]}\t")
fResExfil.write("\tkg\n")

#-----
myPrj.endSimulation()

for ic in range(myPrj.nContaminants):
    fResMfList[ic].close
    fResEnvExfilList[ic].close
fResFlow.close()
fResExfil.close()
if( myPrj.nOutputControls > 0 ):
    fResControl.close()

# --- End main() ---#

if __name__ == "__main__":
    main()
```

test_cxcffi.py

```
from contamxpy import cxLib
import cxResults as cxr
import os
from optparse import OptionParser

# ===== main() =====
# This program takes the full name of a PRJ file and simply runs the simulation
# from beginning to end.

def main():
    # ----- Manage option parser
    parser = OptionParser(usage="%prog [options] arg1\n\targ1=PRJ filename\n")
    parser.set_defaults(verbose=0)
    parser.add_option("-v", "--verbose", action="store", dest="verbose", type="int",
        ↪ default=0,
        ↪ help="define verbose output level: 0=Min, 1=Medium, 2=Maximum.
        ↪ ")
    (options, args) = parser.parse_args()

    # ----- Process command line options -v
```

(continues on next page)

```

verbose = options.verbose

if len(args) != 1:
    parser.error("Need one argument:\n  arg1 = PRJ file.")
    return
else:
    prjPath = args[0]

if not os.path.exists(prjPath):
    print("ERROR: PRJ file not found.")
    return

msg_cmd = "Running test_cxcffi.py: arg1 = " + args[0] + " " + str(options)
print(msg_cmd, "\n")

if verbose > 1:
    print(f"cxLib attributes =>\n{chr(10).join(map(str, dir(cxLib)))}\n")

#----- Initialize contamx-lib object w/ wp_mode and cb_option.
#      wp_mode = 0 => use wind pressure profiles, WTH and CTM files or associated
↪API calls.
#      cb_option = True => set callback function to get PRJ INFO from the
↪ContamXState.
myPrj = cxLib(prjPath, 0, True)
myPrj.setVerbosity(verbose)
if verbose > 1:
    print(f"BEFORE setupSimulation() \n\tnCtms={myPrj.nContaminants}\n\tnZones=
↪{myPrj.nZones}\n\tnPaths={myPrj.nPaths}\n" )

#----- Query State for Version info
verStr = myPrj.getVersion()
if verbose >= 0:
    print(f"getVersion() returned {verStr}.")

#----- Setup Simulation for PRJ
sim_not_ok = myPrj.setupSimulation(1)
if (sim_not_ok > 0):
    print(f"ABORT - sim_not_ok Returned by setupSimulation() = {sim_not_ok}")
    print(" => invalid simulation parameters for co-simulation.")
    myPrj.endSimulation()
    return

# ----- Get simulation run info
dayStart = myPrj.getSimStartDate()
dayEnd   = myPrj.getSimEndDate()
secStart = myPrj.getSimStartTime()
secEnd   = myPrj.getSimEndTime()
tStep    = myPrj.getSimTimeStep()

simBegin = (dayStart - 1) * 86400 + secStart
simEnd   = (dayEnd - 1) * 86400 + secEnd

# ----- Calculate the simulation duration in seconds and total time steps
simBegin = (dayStart - 1) * 86400 + secStart
simEnd   = (dayEnd - 1) * 86400 + secEnd
if (simBegin <= simEnd):
    simDuration = simEnd - simBegin

```

(continues on next page)

```

else:
    simDuration = 365 * 86400 - simEnd + simBegin
numTimeSteps = 0
if (simDuration != 0):
    numTimeSteps = int(simDuration / tStep)
    print(f"PRJ settings => Transient simulation w/ {numTimeSteps} time steps.")
else:
    print("PRJ settings => Steady state simulation.")

#----- Get the current date/time after initial steady state simulation
currentDate = myPrj.getCurrentDayOfYear()
currentTime = myPrj.getCurrentTimeInSec()
if verbose > 0:
    print(f"Sim days = {dayStart}:{dayEnd}")
    print(f"Sim times = {secStart}:{secEnd}")
    print(f"Sim time step = {tStep}")
    print(f"Number of steps = {numTimeSteps}")

#----- Initialize result files
fResMfList = []
root, ext = os.path.splitext(prjPath)
fNameResFlow = root + "_Flow.txt"
fResFlow = open(fNameResFlow, "w")
for ic in range(myPrj.nContaminants):
    fName = root + "_Mf_" + myPrj.contaminants[ic] + ".txt"
    file = open(fName, "w")
    fResMfList.append(file)

#----- Output initial results.
###cxr.printZoneMf(myPrj, currentDate, currentTime, myPrj.nZones, myPrj.
↪nContaminants)\
# Write headers
for ic in range(myPrj.nContaminants):
    cxr.writeMfZones(fResMfList[ic], True, myPrj, currentDate, currentTime, ic)
cxr.writeAirflowRates(fResFlow, True, myPrj, -1, -1)

# Write initial values
for ic in range(myPrj.nContaminants):
    cxr.writeMfZones(fResMfList[ic], False, myPrj, currentDate, currentTime, ic)
cxr.writeAirflowRates(fResFlow, False, myPrj, currentDate, currentTime)

#----- Run Simulation
for i in range(numTimeSteps):
    # Tasks to perform BEFORE current time step.

    myPrj.doSimStep(1)

    # Tasks to perform AFTER current time step.
    currentDate = myPrj.getCurrentDayOfYear()
    currentTime = myPrj.getCurrentTimeInSec()
    if verbose > 1:
        print(f"{i}\t{currentDate},{currentTime}")

    for ic in range(myPrj.nContaminants):
        cxr.writeMfZones(fResMfList[ic], False, myPrj, currentDate, currentTime, ↪
↪ic)
        cxr.writeAirflowRates(fResFlow, False, myPrj, currentDate, currentTime)

```

(continues on next page)

(continued from previous page)

```
myPrj.endSimulation()

for ic in range(myPrj.nContaminants):
    fResMfList[ic].close
    fResFlow.close()

# --- End main() ---#

if __name__ == "__main__":
    main()
```

test_OneFloorWpcAddMf.py

```
from contamxpy import cxLib
import cxResults as cxr
import os, sys
from optparse import OptionParser

## TEST CASE - testOneZoneWpc-UseApi.prj

# =====
# DATA =====
#
# 00:00, 01:00, 01:05, 06:00, 06:05, 12:00, 12:05,
# 13:00, 18:00, 18:05, 24:00
# time in seconds.
times = [
    0, 3600, 3900, 21600, 21900, 43200, 43500,
    46800, 64800, 65100, 86400 ]
# p1wp[] wind pressure units, Pa.
p1wp = [ 2.508542, 2.508542, 2.508542, 2.508542, 2.508542, 2.508542, -2.508542,
    -2.508542, -2.508542, -2.508542, -2.508542 ]
# p2wp[] wind pressure units, Pa.
p2wp = [ -2.508542, -2.508542, -2.508542, -2.508542, -2.508542, -2.508542, 2.508542,
    2.508542, 2.508542, 2.508542, 2.508542 ]
# p1mf0[] mass fraction units, kg_contaminant/kg_air.
p1mf0 = [
    0, 0.01, 0.01, 0.00, 0, 0, 0,
    0, 0, 0, 0 ]
# p2mf0[] mass fraction units, kg_contaminant/kg_air.
p2mf0 = [
    0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0 ]
# p1mf1[] mass fraction units, kg_contaminant/kg_air.
p1mf1 = [
    0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0 ]
# p2mf1[] mass fraction units, kg_contaminant/kg_air.
p2mf1 = [
    0, 0, 0, 0, 0, 0, 0,
    0.02, 0.00, 0, 0 ]

# ===== Set_
# Ambt Data =====
# It is taken for granted that various properties of the PRJ are known:
# - the envIndex of the two flow paths
# - the number of contaminants in the PRJ
#
def setEnvelopePathsWPCInit( cxl ):
```

(continues on next page)

```

    ### print(f"setEnvelopePathsWPCInit({t_index})")
    cxl.setEnvelopeWP(1, p1wp[0])      # envIndex 1
    cxl.setEnvelopeWP(2, p2wp[0])      # envIndex 2
    cxl.setEnvelopeMF(1, 0, p1mf0[0])  # envIndex 1, contaminant 0
    cxl.setEnvelopeMF(2, 0, p2mf0[0])  # envIndex 2, contaminant 0
    cxl.setEnvelopeMF(1, 1, p1mf1[0])  # envIndex 1, contaminant 1
    cxl.setEnvelopeMF(2, 1, p2mf1[0])  # envIndex 2, contaminant 1

# Update conditions when appropriate.
def setEnvelopePathsWPC( cxl, time, date, step, t_index ) -> int:
    # Check if it's time to change to the next set of Ambient data.
    ### print(f"setEnvelopePathsWPC({time},{date},{step},{t_index})")
    if t_index >= len(times):
        return t_index
    if (time == times[t_index]):
        ### print(f"Setting Envelope data...")
        cxl.setEnvelopeWP(1, p1wp[t_index])
        cxl.setEnvelopeWP(2, p2wp[t_index])
        cxl.setEnvelopeMF(1, 0, p1mf0[t_index])
        cxl.setEnvelopeMF(2, 0, p2mf0[t_index])
        cxl.setEnvelopeMF(1, 1, p1mf1[t_index])
        cxl.setEnvelopeMF(2, 1, p2mf1[t_index])
        t_index = t_index + 1
    return t_index

↪#=====
↪main() =====
def main():
    #----- Manage option parser
    parser = OptionParser(usage="%prog [options] arg1\n\targ1=PRJ filename\n")
    parser.set_defaults(verbose=0)
    parser.add_option("-v", "--verbose", action="store", dest="verbose", type="int",
↪default=0,
                        help="define verbose output level: 0=Min, 1=Medium, 2=Maximum.
↪")
    (options, args) = parser.parse_args()

    #----- Process command line options -v
    verbose = options.verbose

    if len(args) != 1:
        parser.error("Need one argument:\n arg1 = PRJ file.")
        return
    else:
        prjPath = args[0]

    if ( not os.path.exists(prjPath) ):
        print("ERROR: PRJ file not found.")
        return

    msg_cmd = "Running test_cxcffi.py: arg1 = " + args[0] + " " + str(options)
    print(msg_cmd, "\n")

    if verbose > 1:
        print(f"cxLib attributes =>\n{chr(10).join(map(str, dir(cxLib)))}\n")

```

```

#----- Initialize contamx-lib object w/ wp_mode and cb_option.
#         wp_mode = 1 => use WPC-like API calls.
#         cb_option = True => set callback function to get PRJ INFO from the
↳ ContamXState.
myPrj = cxLib(prjPath, 1, True, setEnvelopePathsWPCInit)
myPrj.setVerbosity(verbose)
if verbose > 1:
    print(f"BEFORE setupSimulation() \n\tnCtms={myPrj.nContaminants} \n\tnZones=
↳ {myPrj.nZones} \n\tnPaths={myPrj.nPaths} \n" )

#----- Query State for Version info
verStr = myPrj.getVersion()
if verbose >= 0:
    print(f"getVersion() returned {verStr}.")

#----- Setup Simulation for PRJ
myPrj.setupSimulation(1)

#----- Initialize result files
fResMfList = []
fResEnvExfilList = []
root, ext = os.path.splitext(prjPath)
fNameResFlow = root + "_Flow.txt"
fNameResTotalExfil = root + "_Exfil.txt"
fNameResControl = root + "_Control.txt"
fResFlow = open(fNameResFlow, "w")
fResExfil = open(fNameResTotalExfil, "w")
for ic in range(myPrj.nContaminants):
    fName = root + "_Mf_" + myPrj.contaminants[ic] + ".txt"
    file = open(fName, "w")
    fResMfList.append(file)
    fName = root + "_Exfil_" + myPrj.contaminants[ic] + ".txt"
    file = open(fName, "w")
    fResEnvExfilList.append(file)
totalEnvExfil = [0.0] * myPrj.nContaminants
if( myPrj.nOutputControls > 0 ):
    fResControl = open(fNameResControl, "w")

dayStart = myPrj.getSimStartDate()
dayEnd   = myPrj.getSimEndDate()
secStart = myPrj.getSimStartTime()
secEnd   = myPrj.getSimEndTime()
tStep    = myPrj.getSimTimeStep()

simBegin = (dayStart - 1) * 86400 + secStart
simEnd   = (dayEnd - 1) * 86400 + secEnd

#----- Calculate the simulation duration in seconds and total time steps
if (simBegin < simEnd):
    simDuration = simEnd - simBegin
else:
    simDuration = 365 * 86400 - simEnd + simBegin
numTimeSteps = int(simDuration / tStep)

#----- Get the current date/time after initial steady state simulation
currentDate = myPrj.getCurrentDayOfYear()
currentTime = myPrj.getCurrentTimeInSec()

```

(continues on next page)

```

if verbose > 0:
    print(f"Sim days = {dayStart}:{dayEnd}")
    print(f"Sim times = {secStart}:{secEnd}")
    print(f"Sim time step = {tStep}")
    print(f"Number of steps = {numTimeSteps}")

#----- Output initial results.
# Write headers
for ic in range(myPrj.nContaminants):
    cxr.writeMfZones(fResMfList[ic], True, myPrj, currentDate, currentTime, ic)
    cxr.writeEnvExfil(fResEnvExfilList[ic], True, myPrj, -1, -1, -1)
cxr.writeAirflowRates(fResFlow, True, myPrj, -1, -1)
if( myPrj.nOutputControls > 0 ):
    cxr.writeControls(fResControl, True, myPrj, -1, -1)

# Write initial values
for ic in range(myPrj.nContaminants):
    cxr.writeMfZones(fResMfList[ic], False, myPrj, currentDate, currentTime, ic)
    cxr.writeAirflowRates(fResFlow, False, myPrj, currentDate, currentTime)
if( myPrj.nOutputControls > 0 ):
    cxr.writeControls(fResControl, False, myPrj, currentDate, currentTime)

#-----
#----- Run Transient Simulation -----
#-----

wpcIndex = 1

for i in range(numTimeSteps):
    #-----
    #----- Tasks to perform BEFORE current time step.
    #-----
    wpcIndex = setEnvelopePathsWPC( myPrj, currentTime, currentDate, tStep, ↵
↵wpcIndex)

    #-----
    # Run next time step.
    #-----
    myPrj.doSimStep(1)

    #-----
    #----- Tasks to perform AFTER current time step.
    #-----
    currentDate = myPrj.getCurrentDayOfYear()
    currentTime = myPrj.getCurrentTimeInSec()

    #----- Add mass to zone.
    # NOTE: This will show up in the result output of both CONTAM and this driver.
    if( currentTime == (2*3600)):
        myPrj.setZoneAddMass(1, 2, 1.0)
    elif( currentTime == (15*3600)):
        myPrj.setZoneAddMass(2, 2, 2.0)

    #----- Output results of time step just performed.
    for ic in range(myPrj.nContaminants):
        cxr.writeMfZones(fResMfList[ic], False, myPrj, currentDate, currentTime, ↵
↵ic)
    cxr.calcEnvExfil(myPrj, totalEnvExfil)

```

(continues on next page)

(continued from previous page)

```
    for ic in range(myPrj.nContaminants):
        cxr.writeEnvExfil(fResEnvExfilList[ic], False, myPrj, currentDate,
↪currentTime, ic)
        cxr.writeAirflowRates(fResFlow, False, myPrj, currentDate, currentTime)
        if ( myPrj.nOutputControls > 0 ):
            cxr.writeControls(fResControl, False, myPrj, currentDate, currentTime)

    #----- End of simulation loop

    #----- Output total envelope exfiltration.
    #      Should be the same as CSM file.
    print(f"totalEnvExfil[] final:\n\t{myPrj.contaminants}\n\t{totalEnvExfil}")
    for i in range(myPrj.nContaminants):
        fResExfil.write(f"{myPrj.contaminants[i]}\t")
    fResExfil.write("\n")
    for i in range(myPrj.nContaminants):
        fResExfil.write(f"{totalEnvExfil[i]}\t")
    fResExfil.write("\tkg\n")

    myPrj.endSimulation()

    for ic in range(myPrj.nContaminants):
        fResMfList[ic].close
        fResEnvExfilList[ic].close
    fResFlow.close()
    fResExfil.close()
    if ( myPrj.nOutputControls > 0 ):
        fResControl.close()

# --- End main() ---#

if __name__ == "__main__":
    main()
```

test_MultiRun.py

```
from contamxpy import cxLib
import cxResults as cxr
import cxRunSim

import os, sys
from optparse import OptionParser
import time
import threading
import multiprocessing

def checkFile(ext, fname):
    checksOut = True
    root, extension = os.path.splitext(fname)
    if extension != ext:
        checksOut = False
    elif not os.path.exists(fname):
        checksOut = False
    return checksOut
```

(continues on next page)

```

→ #=====
→ main() =====
# Run simulations on multiple PRJ files using synchronous, multithreading,
# and multiprocessing (fails pickling CFFi-wrapped contamx-lib).
# Provides timing of execution.
#
# Usage: test_MultiRun.py prjFile.lst -m1
# prjFile.lst - text file contains
# PRJs: test_GetPrjInfo.prj, test_OneFloorWpcAddMf.prj, test_OneZoneWthCtm.prj,
→ valThreeZonesWthCtm.prj

if __name__ == "__main__":
    METHOD = "Synchronous"

    # ----- Manage option parser
    parser = OptionParser(usage="%prog [options] arg1\n\targ1=list filename")
    parser.set_defaults(test_run=False)
    parser.add_option("-m", "--method", action="store", dest="run_method", type="int",
→ default=0,
                        help="Define concurrency method: 0=Synchronous, 1=Multithread,
→ 2=Multiprocess, 3=ProcessPool.")
    parser.add_option("-t", action="store_true", dest="test_run",
                        help="Test creation of set of files to run, but do not run
→ the simulations.")
    parser.set_defaults(verbose=0)
    parser.add_option("-v", "--verbose", action="store", dest="verbose", type="int",
→ default=0,
                        help="Define verbose output level: 0=Min, 1=Medium, 2=Maximum.
→ ")
    (options, args) = parser.parse_args()

    if len(args) != 1:
        parser.error("Need one argument:\n arg1 = List file.")
        sys.exit(1)
    else:
        List_file_in = args[0]

    if ( not os.path.exists(List_file_in) ):
        print("ERROR: List file not found.")
        sys.exit(1)

    # Process command line options
    TEST_RUN = options.test_run
    verbose = options.verbose
    run_method = options.run_method

    strMethods = ["Synchronous", "Multi-thread", "Multi-process", "ProcessPool"]

    #----- Initialize lists -----
    #
    # Each line in List file should contain {nItemsPerLine} items.
    nItemsPerLine = 1
    prj_list = []
    cxLib_list = []

```

(continues on next page)

```

#----- Create LOG File -----
#
print(f"Method = {strMethods[run_method]}")
print(f"TEST_RUN = {str(TEST_RUN)}")
print("Working Directory = " + os.getcwd() )

#----- Read List file and populate file lists.
# Each line contains the path of a PRJ file.
# Comments are denoted with "#".
nSimsToRun = 0
nLinesRead = 0
fd_List = open(List_file_in, "r")
for line_in_fdList in fd_List:
    if( not line_in_fdList.rstrip('\n') ):
        continue
    # Remove white space and provide line items as list.
    itemsOnLineOfListfile = [x1.strip(' \n') for x1 in line_in_fdList.split(',')]
    if( itemsOnLineOfListfile[0][:1] == '#' ):
        # Skip comment lines in List File.
        continue
    nLinesRead += 1
    if( len(itemsOnLineOfListfile) != nItemsPerLine ):
        print(f"Incorrect number of parameters on line:
→{str(itemsOnLineOfListfile)}")
        continue
    # Verify file types have correct extensions.
    bSkipFile = False
    for file in itemsOnLineOfListfile:
        root, ext = os.path.splitext(file)
        if not checkFile(".prj", file):
            # File not found or incorrect type => skip this set of files
            bSkipFile = True
            print(f"x File not found or incorrect type: {file}
→{str(itemsOnLineOfListfile)}")
            continue
        if bSkipFile == True:
            continue

    # Add PRJ path to list.
    prj_list.append(itemsOnLineOfListfile[0])

    print(f"o {str(nSimsToRun+1)} Simulation to run: {str(itemsOnLineOfListfile)}
→")
    nSimsToRun += 1
    # End

print(f"Number of lines read = {str(nLinesRead)}")
print(f"Number of sims to run = {str(nSimsToRun)}")
if( nSimsToRun == 0 ):
    print("*** ERROR: No simulations to run. ***")
    sys.exit(0)

#-----
#----- Run Simulations -----
#-----
if( TEST_RUN == False ):
    print(f"Run simulations on: {prj_list}.")

```

(continues on next page)

```

time_start = time.time()

#----- Instantiate list of contamx-lib objects
for i in range(nSimsToRun):
    cxLib_list.append(cxLib(prj_list[i], 0, False))

jobs = []
results = []

if(run_method == 0):          # No concurrency
    for myPrj in cxLib_list:
        cxRunSim.run_sim(myPrj)
elif(run_method == 1):       # Threading
    for myPrj in cxLib_list:
        t = threading.Thread(target=cxRunSim.run_sim, args=(myPrj,))
        jobs.append(t)
elif(run_method == 2):       # Multi-processing - Process
    # This method fails due to attempts to pickle
    # CFFI-wrapped contamx-lib.
    for myPrj in cxLib_list:
        p = multiprocessing.Process(target=cxRunSim.run_sim, args=(myPrj,))
        jobs.append(p)
        ### TypeError: cannot pickle '_cffi_backend.__CDataOwnGC' object
else:
    # This method seems to fail due to attempts to pickle
    # CFFI-wrapped contamx-lib, but errors may not be displayed.
    ncpu = multiprocessing.cpu_count()
    NUM_WORKERS = ncpu - 1
    pool = multiprocessing.Pool( processes = NUM_WORKERS )
    print(f"NumCPUs = {ncpu}, NUM_WORKERS = {NUM_WORKERS}")
    results = [pool.apply_async(cxRunSim.run_sim, args=(myPrj,)) for myPrj in
↪cxLib_list]
    pool.close()
    pool.join()

if(run_method == 1 or run_method == 2):
    for j in jobs:
        j.start()
    for j in jobs:
        j.join()

# End if(TEST_RUN)

print(f"elapsed time = {time.time()-time_start} sec")

sys.exit(0)

#--- End main() ---#

```

test_OneZoneSS.py

```
from contamxpy import cxLib
import cxResults as cxr
import os
import sys
from optparse import OptionParser

# =====
↳ setWthCtmInit() =====
# Set the initial conditions for the SetAmbt API test.
# This function is set as a parameter to instantiation of cxLib
# to be called by the contamxpy.prjDataReadyFcnP() in order to set
# ambient boundary conditions for steady-state initialization.
# NOTE: The parameter {cxl as cxLib} is passed through contamx-lib to
# contamxpy.prjDataReadyFcnP() which in turn passes it to this function
# to distinguish the instance of cxLib to be used.
#

def setWthCtmInit( cxl ):
    cxl.setAmbtPressure(101325.0)      # Pa
    cxl.setAmbtWindSpeed(7.397)       # m/s
    cxl.setAmbtWindDirection(270)     # deg
    cxl.setAmbtMassFraction(0, 0.0023254) # 0.0023254 kg/kg = 2800 mg/m3
    cxl.setAmbtTemperature(293.15)     # K

#_
↳ =====
↳ main() =====

def main():
    # ----- Manage option parser
    parser = OptionParser(usage="%prog [options] arg1\n\targ1=PRJ filename\n")
    parser.set_defaults(verbose=0)
    parser.add_option("-v", "--verbose", action="store", dest="verbose", type="int",
↳ default=0,
                        help="define verbose output level: 0=Min, 1=Medium, 2=Maximum.")
    (options, args) = parser.parse_args()

    # ----- Process command line options -v
    verbose = options.verbose

    if len(args) != 1:
        parser.error("Need one argument:\n\targ1 = PRJ file.")
        return
    else:
        # Get PRJ file name
        prjPath = args[0]

    if ( not os.path.exists(prjPath) ):
        print("ERROR: PRJ file not found.")
        return

    msg_cmd = "Running test_OneZoneSS.py: arg1 = " + args[0] + " " + str(options)
```

(continues on next page)

```

print(msg_cmd, "\n")

if verbose > 1:
    print(f"cxLib attributes =>\n{chr(10).join(map(str, dir(cxLib)))}\n")

# ----- Initialize contamx-lib object w/ wp_mode and cb_option.
#         wp_mode = 0 => use wind pressure profiles, WTH and CTM files or associated
↪API calls.
#         cb_option = True => set callback function to get PRJ INFO from the
↪ContamXState.
myPrj = cxLib(prjPath, 0, True, setWthCtmInit)

myPrj.setVerbosity(verbose)
if verbose > 1:
    print(f"BEFORE setupSimulation() \n\t nCtms={myPrj.nContaminants} \n\t nZones=
↪{myPrj.nZones} \n\t nPaths={myPrj.nPaths} \n" )

# ----- Query State for Version info
verStr = myPrj.getVersion()
if verbose >= 0:
    print(f"getVersion() returned {verStr}.")

# ----- Initialize Simulation -----
# Returns 1 if not ok, else 0
sim_not_ok = myPrj.setupSimulation(1)
if (sim_not_ok > 0):
    print(f"ABORT - sim_not_ok Returned by setupSimulation() = {sim_not_ok}")
    print(" => invalid simulation parameters for co-simulation.")
    myPrj.endSimulation()
    return

# ----- Get simulation run info
dayStart = myPrj.getSimStartDate()
dayEnd   = myPrj.getSimEndDate()
secStart = myPrj.getSimStartTime()
secEnd   = myPrj.getSimEndTime()
tStep    = myPrj.getSimTimeStep()

buildingVolume = 0.0
for zone in myPrj.zones:
    buildingVolume += zone.volume
buildingMass = 1.2041 * buildingVolume

# ----- Calculate the simulation duration in seconds and total time steps
simBegin = (dayStart - 1) * 86400 + secStart
simEnd   = (dayEnd - 1) * 86400 + secEnd
if (simBegin <= simEnd):
    simDuration = simEnd - simBegin
else:
    simDuration = 365 * 86400 - simEnd + simBegin
numTimeSteps = 0
if (simDuration != 0):
    numTimeSteps = int(simDuration / tStep)
    print(f"ABORT - PRJ settings => Transient simulation w/ {numTimeSteps} time
↪steps.")

```

```

    return
else:
    print("PRJ settings => Steady state simulation.")

    # ----- Get the current date/time after initial steady state simulation
    currentDate = myPrj.getCurrentDayOfYear()
    currentTime = myPrj.getCurrentTimeInSec()
    if verbose > 0:
        print(f"Sim days = {dayStart}:{dayEnd}")
        print(f"Sim times = {secStart}:{secEnd}")
        print(f"Sim time step = {tStep}")
        print(f"date = {currentDate} time = {currentTime}")

    # ----- Output SS results.
    cxr.printZoneMf(myPrj, currentDate, currentTime, myPrj.nZones, myPrj.
    ↪nContaminants)
    cxr.writeAirflowRates(sys.stdout, True, myPrj, currentDate, currentTime)
    cxr.writeAirflowRates(sys.stdout, False, myPrj, currentDate, currentTime)

    # Calculate building Air Change Rate
    sumAbsFlows = 0.0
    for path in myPrj.paths:
        if path.envIndex > 0:
            flows = myPrj.getPathFlow(path.nr)
            netPathFlow = flows[0] + flows[1]
            sumAbsFlows += abs(netPathFlow)
    buildingAcr = 0.5 * 3600. * sumAbsFlows / buildingMass
    if verbose > 0:
        print(f"sumAbsFlows = {sumAbsFlows} [kg/s]")
        print(f"buildingVolume = {buildingVolume} [m3]")
        print(f"buildingMass = {buildingMass} [kg]")

    print(f"buildingAcr = {buildingAcr} [1/h]")
    print("Performed SS simulation.")
    myPrj.endSimulation()

# --- End main() ---#

if __name__ == "__main__":
    main()

```

cxRunSim.py

```

import contamxpy
from contamxpy import cxLib as cxl

def run_sim(cxl):
    # ----- Setup Simulation for PRJ
    print(f"===== run_sim(cxLib:{cxl}, prj:{cxl.prj_file_path})")
    cxl.setupSimulation(1)

    # ----- Get simulation run info
    dayStart = cxl.getSimStartDate()
    dayEnd = cxl.getSimEndDate()

```

(continues on next page)

(continued from previous page)

```
secStart = cxl.getSimStartTime()
secEnd   = cxl.getSimEndTime()
tStep    = cxl.getSimTimeStep()

# ----- Calculate the simulation duration in seconds and total time steps
simBegin = (dayStart - 1) * 86400 + secStart
simEnd   = (dayEnd - 1) * 86400 + secEnd
if (simBegin < simEnd):
    simDuration = simEnd - simBegin
else:
    simDuration = 365 * 86400 - simEnd + simBegin
numTimeSteps = int(simDuration / tStep)

# ----- Get the current date/time after initial steady state simulation
for i in range(numTimeSteps):
    cxl.doSimStep(1)
cxl.endSimulation()
```

cxResults.py

```
#!/ python3
import contamxpy as cxLib

def writeMfZones(file, header, cxl: cxLib, date, time, ctmNum):
    """Output zone mass fractions to text file."""
    nz = cxl.nZones
    if header is True:
        file.write("Day\tTime")
        for iz in range(nz):
            file.write(f"\t{cxl.zones[iz].name}")
        file.write("\n")
    else:
        file.write(f"{date}\t{time}")
        for iz in range(nz):
            MF = cxl.getZoneMassFraction(cxl.zones[iz].nr, ctmNum)
            file.write(f"\t{MF}")
        file.write("\n")

def printZoneMf(cxl: cxLib, date, time, nz, nc):
    """Output zone mass fractions to stdout. """
    # cxl.zones[0-nZones-1], CONTAM => zones[1..nZones]
    # cxl.contaminants AND CONTAM contaminants[0..nContaminants-1]
    for z in range(nz):
        for c in range(nc):
            MF = cxl.getZoneMassFraction(cxl.zones[z].nr, c)
            print(f"Day {date}\tTime {time}\tZone {cxl.zones[z].nr}\t{cxl.
→contaminants[c]}\t{MF} kg/kg")

def calcEnvExfil(cxl: cxLib, totalEnvExfil):
    """Calculate the total envelope exfiltration for each Contaminant. """
    nCtm = cxl.nContaminants
    for ic in range(nCtm):
```

(continues on next page)


```

        for path in cxl.envPaths:
            mass = cxl.getEnvelopeExfil(path.envIndex, ic)
            totalEnvExfil[ic] = totalEnvExfil[ic] + mass

def writeEnvExfil(file, header, cxl: cxLib, date, time, ctmNum):
    """Write envelope exfiltration for each envelope flow path to file."""
    np = cxl.nEnvPaths
    if header is True:
        file.write("Day\tTime")
        for path in cxl.envPaths:
            file.write(f"\t{path.nr}")
        file.write("\tkg\n")
    else:
        file.write(f"{date}\t{time}")
        for path in cxl.envPaths:
            Mass = cxl.getEnvelopeExfil(path.envIndex, ctmNum)
            file.write(f"\t{Mass}")
        file.write("\n")

def writeAirflowRates(file, header: bool, cxl: cxLib, date: int, time: int):
    """Write airflows to file."""
    # Paths, DuctsTerminals, and DuctLeaks.
    # Paths include simple AHS (Rec, OA, Exh).
    AHS_I = int("0x0070", 16) # implicit (R/O/X) AHS paths
    AHS_S = int("0x0008", 16) # system supply or return path
    AHS_R = int("0x0010", 16) # recirculation flow path (R)
    AHS_O = int("0x0020", 16) # outside air flow path (O)
    AHS_X = int("0x0040", 16) # exhaust flow path (X)

    np = cxl.nPaths
    if header is True:
        file.write("Day\tTime")
        for path in cxl.paths:
            strHeader = f"p{path.nr}"
            if (path.flags & AHS_I):
                if (path.flags & AHS_R):
                    strHeader += f"_ahs-{path.ahs_nr}-Rec"
                elif (path.flags & AHS_O):
                    strHeader += f"_ahs-{path.ahs_nr}-OA"
                elif (path.flags & AHS_X):
                    strHeader += f"_ahs-{path.ahs_nr}-Exh"
            elif (path.flags & AHS_S):
                strHeader += f"_ahs-{path.ahs_nr}-SR"
            file.write(f"\t{strHeader}")
        for term in cxl.ductTerminals:
            file.write(f"\tt{term.nr}")
        for leak in cxl.ductLeaks:
            file.write(f"\tl{leak.nr}")
        file.write("\t\nNet flows kg/s\n\n")
    else:
        file.write(f"{date}\t{time}")
        for path in cxl.paths:
            flows = cxl.getPathFlow(path.nr)
            file.write(f"\t{flows[0]} + {flows[1]}")
        for it in range(cxl.nDuctTerminals):

```

(continues on next page)

```

        flow = cxl.getDuctTerminalFlow(it + 1)
        file.write(f"\t{flow}")
    for il in range(cxl.nDuctLeaks):
        flow = cxl.getDuctLeakFlow(il + 1)
        file.write(f"\t{flow}")
    file.write("\n")

def writeControls(file, header: bool, cxl: cxLib, date: int, time: int):
    nc = cxl.nOutputControls
    if header is True:
        file.write("Day\tTime")
        for control in cxl.outputControls:
            file.write(f"\t{control.name}")
        file.write("\n")
    else:
        file.write(f"{date}\t{time}")
        for i in range(cxl.nOutputControls):
            val = cxl.getOutputControlValue(i+1)
            file.write(f"\t{val}")
        file.write("\n")

```

12 NIST Developer Notes

Calling hierarchy within `contamxpy.cxLib.setupSimulation()` (page 6):

```

struct ContamXState * cxs

SetupSimulation(cxs, projectPath)
{
    contamx(cxs) {
        prj_read(cxs, prjPath)
        sim_data(cxs) {
            cxs->nafnd = afnd_set(cxs)
            cxs->nafpt = afpt_set(cxs)
        }
        setup_cosim_lists(cxs)
    }
}

```

12.1 Number of items in the *state*/PRJ

- *nzone* => Includes “standard” zones and Implicit Simple AHS (SAHS) Supply and Return zones. These items are referenced by `cxs->ZoneList[]`.
- *npath* => Includes “standard” paths, SAHS Inlets and Outlets, and Implicit SAHS paths (OA -> Supply, Return -> Supply, Return -> Exhaust). These items are referenced by `cxs->PatList[]`
- *nduct* => Includes all duct segments in the PRJ. These items are referenced in `cxs->DcList[]`.
- *njct* => Includes “standard” junctions and terminals referenced by `cxs->JctList[]`

12.2 Lists created by ContamX for simulation:

The following lists are created by calls from *sim_data()*:

- **afnd0** => List of *AF_NODE*s created by ContamX in *afnd_set()*.

Includes *cxs->nafnd* items set upon return from *afnd_set()*:

- *nzone* items from *cxs->ZoneList[]*
- *njct* items from *cxs->JctList[]*
- *cxs->pambt* is the last node in the list which is also referenced by *cxs->ambt.pafn*.

- **afpt0** => List of *AF_PATH*s created by ContamX in *afpt_set()*.

Includes *cxs->nafpt* items set upon return from *afpt_set()*:

- *npath* items from *cxs->PathList[]*.
- *nduct* items from *cxs->DcList[]*.
- Each Terminal junction (*AF_NODE*s: junction -> to_zone).
- Each duct Leak (*AF_NODE*s: junction -> containing_zone). *AF_PATH*s for Leaks are created in *afpt_set()* based on junction leakage area data field, *JCT_DSC->CA*.

12.3 Co-simulation lists created by *contamx-lib*:

setup_cosim_lists(ContamXState* cxs)

```
struct ContamXState {
    ...
    struct cosimState cosim
    {
        AF_NODE** cosim_zone_list; // list of pointers to zones [1:cxs->nzone]
        AF_NODE** cosim_jct_list;  // list of pointers to junction nodes [1:cxs->
↪njct]
        AF_PATH** cosim_path_list; // list of pointers to path links [1:cxs->npath]
        AHS_DSC** cosim_ahs_list;  // list of pointers to Simple AHSs [1:cxs->nahs]
        AF_PATH** cosim_oap_list;  // list of pointers to AHS outdoor air path_
↪[1:cxs->nahs]
        AF_PATH** cosim_term_list; // list of pointers to terminal links [1:cxs->
↪cosim.num_cosim_terms]
        AF_PATH** cosim_leak_list; // list of pointers to terminal links [1:cxs->
↪cosim.num_cosim_leaks]
        CT_NODE** cosim_inode_list; // list of pointers to input control nodes [1:cxs->
↪cosim.num_cosim_inodes]
        CT_NODE** cosim_onode_list; // list of pointers to output control nodes_
↪[1:cxs->cosim.num_cosim_onodes]

        // NOTE: these values are determined by setup_cosim_lists() function.
        IX num_cosim_inodes; // number of input control nodes (CT_SET w/ names)
        IX num_cosim_onodes; // number of output control nodes (CT_PAS w/ names)
        IX num_cosim_terms;  // number of terminals
        IX num_cosim_leaks;  // number of junction leaks
    }
}
```

12.4 contamx-lib Functions that Reference *AF_NODES*

cxiSetZoneAddMass()

This function currently only works for zones *1* to *cxs->nzone*, which includes Implicit SAHS zones.

Todo: *cxiSetZoneAddMass()* will only work for zones having a non-zero mass.

```
IX cxiSetZoneAddMass(void* contamXState, IX zoneNumber, IX ctmNumber, R8 addMass)
{
    double addMf = 0.0;
    struct AF_NODE *pn = cxs->cosim.cosim_zone_list[zoneNumber];
    if(pn->M > 0.0)
    {
        addMf = addMass / pn->M;
    }
    pn = cxs->cosim.cosim_zone_list[zoneNumber];
    pn->Mf[ctmNumber] += addMf;
}
```

cxiSetZoneTemperature()

This function currently only works for zones *1* to *cxs->nzone*, which includes Implicit SAHS zones.

```
IX cxiSetZoneTemperature(void* contamXState, IX zoneNumber, R8 temperature)
{
    cxs->cosim.cosim_zone_list[zoneNumber]->T = temperature;
}
```

cxiSetJunctionTemperature()

This function works for junctions *1* to *cxs->njct* which includes terminals.

```
IX cxiSetJunctionTemperature(void* contamXState, IX jctNumber, R8 temperature)
{
    cxs->cosim.cosim_jct_list[jctNumber]->T = temperature;
}
```

12.5 TO DO

Todo: Add error handling.

Todo: *relHt* of all *AF_PATH* items, i.e., Paths, DuctJunctions, and DuctTerminals, are 0.0. *AF_PATH* in ContamX does not include a *relHt* field. *relHt* is used to set the absolute coordinate *Z* which in turn is used to set relative node heights *Ht_m* and *Ht_n*. CHECK the ramifications of this for multiple levels. *Z* and other coordinates may not be relevant except for WPC-like API functions.

Todo: Need to test and establish precedents between Control values determined in PRJ, `cxiSetInputControlValue()`, and `cxiSetZoneTemperature()`. Currently, an Input Control applied to the temperature of a zone will override a `cxiSetZoneTemperature()` for the timestep applied. Currently, it is best to utilize `cxiSetZoneTemperature()` to establish zone temperatures instead of applying controls to Tzone.

Todo: Test errors, e.g., terminal number and leak number out of range in `getDuctTerminalFlow()` and `getDuctLeakFlow()`.

Note: No wrapper is provided for `cxiSetUseVolumeFlows()`. This API function is specific to EnergyPlus.

Index

A

AHS (*class in contamxpy*), 13
ahs_nr (*contamxpy.Path attribute*), 13
AHSs (*contamxpy.cxLib attribute*), 4

C

cb_option (*contamxpy.cxLib attribute*), 3
containing_zone (*contamxpy.DuctJunction attribute*), 16
contaminants (*contamxpy.cxLib attribute*), 3
Control (*class in contamxpy*), 16
cxLib (*class in contamxpy*), 2

D

doSimStep() (*contamxpy.cxLib method*), 7
DuctJunction (*class in contamxpy*), 15
ductJunctions (*contamxpy.cxLib attribute*), 5
DuctLeak (*class in contamxpy*), 16
ductLeaks (*contamxpy.cxLib attribute*), 5
DuctTerminal (*class in contamxpy*), 14
ductTerminals (*contamxpy.cxLib attribute*), 5

E

endSimulation() (*contamxpy.cxLib method*), 7
envIndex (*contamxpy.DuctJunction attribute*), 16
envIndex (*contamxpy.DuctTerminal attribute*), 15
envIndex (*contamxpy.Path attribute*), 13
envPaths (*contamxpy.cxLib attribute*), 4
envTerminals (*contamxpy.cxLib attribute*), 5

F

flags (*contamxpy.DuctJunction attribute*), 16
flags (*contamxpy.DuctTerminal attribute*), 14
flags (*contamxpy.Path attribute*), 12
flags (*contamxpy.Zone attribute*), 11
from_zone (*contamxpy.Path attribute*), 12

G

getCurrentDayOfYear() (*contamxpy.cxLib method*), 7
getCurrentTimeInSec() (*contamxpy.cxLib method*), 7
getDuctLeakFlow() (*contamxpy.cxLib method*), 10
getDuctTerminalFlow() (*contamxpy.cxLib method*), 10
getEnvelopeExfil() (*contamxpy.cxLib method*), 9
getOutputControlValue() (*contamxpy.cxLib method*), 10
getPathFlow() (*contamxpy.cxLib method*), 9

getSimEndDate() (*contamxpy.cxLib method*), 6
getSimEndTime() (*contamxpy.cxLib method*), 7
getSimStartDate() (*contamxpy.cxLib method*), 6
getSimStartTime() (*contamxpy.cxLib method*), 6
getSimTimeStep() (*contamxpy.cxLib method*), 6
getVersion() (*contamxpy.cxLib method*), 6
getZoneMassFraction() (*contamxpy.cxLib method*), 9

I

InputControl (*class in contamxpy*), 16
inputControls (*contamxpy.cxLib attribute*), 4

L

level_name (*contamxpy.Zone attribute*), 11
level_nr (*contamxpy.Zone attribute*), 11

N

nAhs (*contamxpy.cxLib attribute*), 4
name (*contamxpy.AHS attribute*), 13
name (*contamxpy.Control attribute*), 16
name (*contamxpy.Zone attribute*), 11
nContaminants (*contamxpy.cxLib attribute*), 3
nDuctJunctions (*contamxpy.cxLib attribute*), 5
nDuctLeaks (*contamxpy.cxLib attribute*), 5
nDuctTerminals (*contamxpy.cxLib attribute*), 5
nEnvPaths (*contamxpy.cxLib attribute*), 4
nEnvTerminals (*contamxpy.cxLib attribute*), 5
nInputControls (*contamxpy.cxLib attribute*), 4
nOutputControls (*contamxpy.cxLib attribute*), 4
nPaths (*contamxpy.cxLib attribute*), 4
nr (*contamxpy.AHS attribute*), 13
nr (*contamxpy.Control attribute*), 16
nr (*contamxpy.DuctJunction attribute*), 16
nr (*contamxpy.DuctTerminal attribute*), 14
nr (*contamxpy.Path attribute*), 12
nr (*contamxpy.Zone attribute*), 11
nZones (*contamxpy.cxLib attribute*), 3

O

OutputControl (*class in contamxpy*), 17
outputControls (*contamxpy.cxLib attribute*), 5

P

Path (*class in contamxpy*), 12
path_exh (*contamxpy.AHS attribute*), 14
path_oa (*contamxpy.AHS attribute*), 14
path_rec (*contamxpy.AHS attribute*), 14
paths (*contamxpy.cxLib attribute*), 4
prj_file_path (*contamxpy.cxLib attribute*), 2, 3

prjDataReadyFcnP () (in module contamxpy), 17

R

relHt (contamxpy.DuctTerminal attribute), 15

S

setAhsPercentOa () (contamxpy.cxLib method), 9

setAhsSupplyReturnFlow () (contamxpy.cxLib method), 8

setAmbtMassFraction () (contamxpy.cxLib method), 7

setAmbtPressure () (contamxpy.cxLib method), 7

setAmbtTemperature () (contamxpy.cxLib method), 7

setAmbtWindDirection () (contamxpy.cxLib method), 7

setAmbtWindSpeed () (contamxpy.cxLib method), 7

setEnvelopeMF () (contamxpy.cxLib method), 8

setEnvelopeWP () (contamxpy.cxLib method), 7

setInputControlValue () (contamxpy.cxLib method), 10

setJunctionTemperature () (contamxpy.cxLib method), 8

setupSimulation () (contamxpy.cxLib method), 6

setVerbosity () (contamxpy.cxLib method), 5

setZoneAddMass () (contamxpy.cxLib method), 8

setZoneTemperature () (contamxpy.cxLib method), 8

T

to_zone (contamxpy.DuctTerminal attribute), 15

to_zone (contamxpy.Path attribute), 12

V

verbose (contamxpy.cxLib attribute), 3

volume (contamxpy.Zone attribute), 11

W

wp_mode (contamxpy.cxLib attribute), 3

wth_init_function (contamxpy.cxLib attribute), 3

X

X (contamxpy.DuctTerminal attribute), 14

X (contamxpy.Path attribute), 13

Y

Y (contamxpy.DuctTerminal attribute), 15

Y (contamxpy.Path attribute), 13

Z

Z (contamxpy.DuctTerminal attribute), 15

Z (contamxpy.Path attribute), 13

Zone (class in contamxpy), 11

zone_ret (contamxpy.AHS attribute), 13

zone_sup (contamxpy.AHS attribute), 14

zones (contamxpy.cxLib attribute), 4