contamxpy Documentation

Release 0.0.6

W. Stuart Dols, Brian J. Polidoro

Feb 06, 2023

Contents

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.

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
```

See also:

Example Driver Programs (page 16)

2 cxLib

class `contamxpy.`**`cxLib`**(*wp=0*, *cb=False*, *wthFunc=None*)

cxLib provides a wrapper around `contamx-lib`.

`wp`

Set wind pressure calculation method *`cxLib.wpMode`* (page 2)

- 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`

Set callback option *`cxLib.cbOption`* (page 2) for *`contamxpy.prjDataReadyFcnP()`* (page 16)

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

Type

bool, optional

`wthFunc`

Set user-defined function *`cxLib.wthInitFunction()`* (page 3) via this parameter. It will be called by *`contamxpy.prjDataReadyFcnP()`* (page 16), for example, to set initial ambient conditions prior to running steady-state initialization calculations.

Type

(function name), optional

`wpMode`: `int`

Wind pressure calculation mode.

cbOption: bool

Callback function option.

wthInitFunction

Weather and contaminant boundary condition initialization function.

Used to set ambient conditions prior to running steady-state initialization.

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 16) callback function if `cxLib.cbOption` (page 2) = *True*.

contaminants: list

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

Set via the `contamxpy.prjDataReadyFcnP()` (page 16) callback function if `cxLib.cbOption` (page 2) = *True*.

nZones: int

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

Set via the `contamxpy.prjDataReadyFcnP()` (page 16) callback function if `cxLib.cbOption` (page 2) = *True*.

zones: list

(Read-only) List of `contamxpy.Zone` (page 10) objects.

Set via the `contamxpy.prjDataReadyFcnP()` (page 16) callback function if `cxLib.cbOption` (page 2) = *True*.

nPaths: int

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

Set via the `contamxpy.prjDataReadyFcnP()` (page 16) callback function if `cxLib.cbOption` (page 2) = *True*.

paths: list

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

Set via the `contamxpy.prjDataReadyFcnP()` (page 16) callback function if `cxLib.cbOption` (page 2) = *True*.

nEnvPaths: int

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

Set via the `contamxpy.prjDataReadyFcnP()` (page 16) callback function if `cxLib.cbOption` (page 2) = *True*.

envPaths: list

(Read-only) List of `contamxpy.Path` (page 11) objects with connections to Ambient.

Set via the `contamxpy.prjDataReadyFcnP()` (page 16) callback function if `cxLib.cbOption` (page 2) = *True*.

nAhs: int

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

Set via the `contamxpy.prjDataReadyFcnP()` (page 16) callback function if `cxLib.cbOption` (page 2) = *True*.

AHSs: list

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

Set via the `contamxpy.prjDataReadyFcnP()` (page 16) callback function if `cxLib.cbOption` (page 2) = *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 16) callback function if `cxLib.cbOption` (page 2) = *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 16) callback function if `cxLib.cbOption` (page 2) = *True*.

inputControls: list

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

Set via the `contamxpy.prjDataReadyFcnP()` (page 16) callback function if `cxLib.cbOption` (page 2) = *True*.

outputControls: list

(Read-only) List of `contamxpy.OutputControl` (page 16) objects.

Set via the `contamxpy.prjDataReadyFcnP()` (page 16) callback function if `cxLib.cbOption` (page 2) = *True*.

nDuctJunctions: int

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

Set via the `contamxpy.prjDataReadyFcnP()` (page 16) callback function if `cxLib.cbOption` (page 2) = *True*.

ductJunctions: list

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

Set via the `contamxpy.prjDataReadyFcnP()` (page 16) callback function if `cxLib.cbOption` (page 2) = *True*.

nDuctTerminals: int

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

Set via the `contamxpy.prjDataReadyFcnP()` (page 16) callback function if `cxLib.cbOption` (page 2) = *True*.

ductTerminals: list

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

Set via the `contamxpy.prjDataReadyFcnP()` (page 16) callback function if `cxLib.cbOption` (page 2) = *True*.

nEnvTerminals: int

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

Set via the *contamxpy.prjDataReadyFcnP()* (page 16) callback function if *cxLib.cbOption* (page 2) = *True*.

envTerminals: list

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

Set via the *contamxpy.prjDataReadyFcnP()* (page 16) callback function if *cxLib.cbOption* (page 2) = *True*.

nDuctLeaks: int

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

Set via the *contamxpy.prjDataReadyFcnP()* (page 16) callback function if *cxLib.cbOption* (page 2) = *True*.

ductLeaks: list

(Read-only) List of `contamxpy.DuctLeak` objects.

Set via the *contamxpy.prjDataReadyFcnP()* (page 16) callback function if *cxLib.cbOption* (page 2) = *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(*prjPath*, *useCosim=1*)

Setup the simulation including the option to run ContamX in the co-simulation mode. Calling cxiSetupSimulation() with *useCosim* 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 cxiRegisterCallback_PrjDataReady(), and running the steady state initialization.

Parameters

- **prjPath** – *str* The file system path to the CONTAM PRJ file on which to run the simulation
- **useCosim** – *int* Select ContamX run mode:
  - 0 = run a CONTAM-only simulation,
  - 1 = run ContamX in co-simulation mode.

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(*stepForward=1*)

Run next simulation time step.

Parameters
stepForward – *int* Currently only a value of *1* is allowed to 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 6) has been called for the values obtained for the ending date and time of the simulation.

setAmbtTemperature(*T*)

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

setAmbtPressure(*P*)

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

setAmbtWindSpeed(*WS*)

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

setAmbtWindDirection(*WD*)

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

setAmbtMassFraction(*ctmNumber*, *Mf*)

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

setEnvelopeWP(*envIndex*, *WP*)

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

Parameters

- **envIndex** – *int* Index set by ContamX. See `contamxpy.Path.envIndex` (page 12).
- **WP** – *float* Wind pressure value [Pa].

setEnvelopeMF(*envIndex*, *ctmNr*, *MF*)

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

Parameters

- **envIndex** – *int* Index set by ContamX. See `contamxpy.Path.envIndex` (page 12).
- **ctmNr** – *int* Contaminant number, e.g., assigned by ContamW
- **MF** – *float* Mass fraction [kg_cont/kg_air].

setZoneAddMass(*zoneNr*, *ctmNr*, *mass*)

Add mass of contaminant to zone.

Parameters

- **zoneNr** – *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*.
- **ctmNr** – *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(*zoneNr*, *temperature*)

Set zone temperature.

Parameters

- **zoneNr** – *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*.
- **temp** – *float* Temperature [K] to set (>= 0.0).

Return type
`int`

setJunctionTemperature(*jctNr*, *temperature*)

Set Duct Junction temperature.

Parameters

- **jctNr** – *int* Junction number for which temperature should be set. Zone numbers range from *1* to `contamxpy.cxLib.nDuctJunctions` (page 4) and are assigned by *ContamW*.
- **temp** – *float* Temperature [K] to set (>= 0.0).

Return type
`int`

setAhsSupplyReturnFlow(*pathNr*, *flow*)

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

Parameters

- **pathNr** – *int* Airflow path number for which flow should be set. Path numbers range from *1* to `contamxpy.cxLib.nPaths` (page 3) and are assigned by *ContamW*.
- **flow** – *float* Mass flow rate [kg/s] to set (>= 0.0).

setAhsPercentOa(*ahsNr*, *fOA*)

Set Outdoor Air fraction for Simple AHS.

Parameters

- **ahsNr** – *int* Air-handling System number for which OA fraction is to be set. AHS numbers range from *1* to `contamxpy.cxLib.nAhs` (page 3) and are typically assigned by *ContamW*.
- **fOA** – *float* Fraction of outdoor air (0 <= fOA <= 1.0)

getZoneMassFraction(*zoneNr*, *ctmNr*)

Get the mass fraction of a zone for selected contaminant.

Parameters

- **zoneNr** – *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*.
- **ctmNr** – *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*, *ctmNr*)

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

Parameters

- **envIndex** – *int* Envelope index of airflow path for which to obtain exfiltration.

- **ctmNr** – *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 3) `contamxpy.Path.envIndex` (page 12)

getPathFlow(*pathNumber*)

Get airflow rate through path.

Parameters
pathNumber – *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(*termNumber*)

Get airflow rate through duct terminal.

Parameters
termNumber – *int* Number of duct terminal (1 -> `contamxpy.cxLib.ductTerminals` (page 4)) in the `contamxpy.cxLib.ductTerminals` (page 4) 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(*leakNumber*)

Get airflow rate through duct leak.

Parameters
leakNumber – *int* Number of duct leak (1 -> `contamxpy.cxLib.nDuctLeaks` (page 5)) in the `contamxpy.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 -> `contamxpy.cxLib.nInputControls` (page 4)) in the `contamxpy.cxLib.inputControls` (page 4) list for which to set the value.

- **val** – *float* Value to set.

Todo: Add error handling.

---

getOutputControlValue(*i*)

Get Output control value (CT_PASS).

Parameters
i – *int* Index of Output control (1 -> *contamxpy.cxLib.nOutputControls* (page 4)) in the *contamxpy.cxLib.outputControls* (page 4) 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*, *Vol*, *level_nr*, *level_name*)

Instances of *Zone* (page 10) are created via the *contamxpy.prjDataReadyFcnP()* (page 16) callback function. If the callback function is called as indicated via the *cxLib* (page 2) constructor, then *Zones* will will be accessible via the *cxLib.zones* (page 3) list.

zoneNr

Zone number, typically assigned by *ContamW*.

Type
int

zoneName

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

`Vol`

Zone volume [m^3]

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 11) 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 16) callback function. If the callback function is called as indicated via the `cxLib` (page 2) constructor, then *Paths* will will be accessible via the `cxLib.paths` (page 3) 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 16) callback function. If the callback function is called as indicated via the `cxLib` (page 2) constructor, then *AHS*s will 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 16) callback function. If the callback function is called as indicated via the `cxLib` (page 2) constructor, then *DuctTerminals* will will be accessible via the `cxLib.ductTerminals` (page 4) 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 16) callback function. If the callback function is called as indicated via the `cxLib` (page 2) constructor, then *DuctJunctions* will will be accessible via the `cxLib.ductJunctions` (page 4) 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 Control

Instances of *Controls* are created via the `contamxpy.prjDataReadyFcnP()` (page 16) callback function. If the callback function is called as indicated via the `cxLib` constructor, then *Controls* will will be accessible via the `contamxpy.cxLib.inputControls` (page 4) and `contamxpy.cxLib.outputControls` (page 4) lists.

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

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

nr

Control node number, typically assigned by *ContamW*.

name

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

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

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

typeStr = CT_SET

class contamxpy.**OutputControl**(*nr*, *name*, *strType='CT_PASS'*)

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

typeStr = CT_PASS

9 Callback Function

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

This function populates the list Attributes of `cxLib` (page 2) contaminants[], zones[], and paths[].

Parameters

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

10 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 18) section below. These drivers also import the *cxResults.py* (page 32) module to obtain and output results to text files.

10.1 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
```

10.2 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 simply be run via the generic driver program *test_cxcffi.py* (page 25). This driver 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

10.3 WPC-like API Cases

This case can be run via the *test_OneFloorWpcAddMf.py* (page 27) 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
```

Corresponding non-API version:

- test_OneFloorWpcAddMf.prj
  - test_OneFloor.wpc

10.4 Driver Programs

test_OneZoneWthCtm.py

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

(continues on next page)

(continued from previous page)

```
                        ])

    dataSetNum = 1
    if prjName == "test_OneZoneWthCtmStack-UseApi":
        dataSetNum = 2
        #                             0      6:00:00 12:00:00 13:00:00 24:00:00
        times_data  = np.array([      0,        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):
        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]
```

(continues on next page)

(continued from previous page)

```
        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}\
↪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 prjDataReadyFcnP() in order to set
#   ambient boundary conditions for steady-state initialization.
# NOTE: No parameters in order to utilize as argument to cxLib
#   constructor.
#
def setWthCtmInit( cxl ):
    global g_times, g_Tambt, g_Pambt, g_WSambt, g_WDambt, g_MFambt, g_Tzone, g_PctOa,␣
↪g_AhsFlow, g_CtrlIn
    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
    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,␣
```

(continues on next page)

(continued from previous page)

```
↪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.Vol                    # 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"
    for path in ahs.return_points:
        Zone = cxl.zones[path.from_zone-1]
        Vzone = Zone.Vol                    # 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)
```

(continues on next page)

(continued from previous page)

```
↪#=====================================================================================␣
↪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=====\nDataSetName = {prjName}\n=====\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/ wpMode and cbOption.
    #      wpMode = 0 => use wind pressure profiles, WTH and CTM files or associated␣
↪API calls.
    #      cbOption = True => set callback function to get PRJ INFO from the␣
↪ContamXState.
    myPrj = cxLib(0, True, setWthCtmInit)
    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
```

(continues on next page)

(continued from previous page)

```
    verStr = myPrj.getVersion()
    if verbose >= 0:
        print(f"getVersion() returned {verStr}.")

    #-----------------------------------------------------------------
    #------------------------------------ Initialize Simulation -----
    #-----------------------------------------------------------------
    myPrj.setupSimulation(prjPath, 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.
```

(continues on next page)

(continued from previous page)

```
↪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)
        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")
```

(continues on next page)

(continued from previous page)

```
    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, sys
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
    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
```

(continues on next page)

(continued from previous page)

```
    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/ wpMode and cbOption.
    #       wpMode = 0 => use wind pressure profiles, WTH and CTM files or associated␣
↪API calls.
    #       cbOption = True => set callback function to get PRJ INFO from the␣
↪ContamXState.
    myPrj = cxLib(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
    myPrj.setupSimulation(prjPath, 1)

    #----- 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)

    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()
    if verbose > 0:
        print(f"Sim days = {dayStart}:{dayEnd}")
```

(continues on next page)

(continued from previous page)

```
        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.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)

    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
```

(continues on next page)

(continued from previous page)

```
↪#==================================================================================================␣
↪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 ):
    ### 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
```

(continues on next page)

(continued from previous page)

```
↪#==========================================================================␣
↪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/ wpMode and cbOption.
    #      wpMode = 1 => use WPC-like API calls.
    #      cbOption = True => set callback function to get PRJ INFO from the␣
↪ContamXState.
    myPrj = cxLib(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(prjPath, 1)

    #----- Initialize result files
    fResMfList = []
    fResEnvExfilList = []
    root, ext = os.path.splitext(prjPath)
    fNameResFlow = root + "_Flow.txt"
    fNameResTotalExfil = root + "_Exfil.txt"
```

(continues on next page)

(continued from previous page)

```
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()
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 -----
```

(continues on next page)

(continued from previous page)

```
    #------------------------------------------------------------------
    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)
        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
```

(continues on next page)

(continued from previous page)

```
    fResFlow.close()
    fResExfil.close()
    if( myPrj.nOutputControls > 0 ):
        fResControl.close()

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

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

cxResults.py

```
#! python3
import contamxpy as cxLib

# Output zone mass fractions to text file.
def writeMfZones(file, header, cxl : cxLib, date, time, ctmNum):
    nz = cxl.nZones
    if header == True:
        file.write(f"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")

# Output zone mass fractions to stdout.
def printZoneMf(cxl : cxLib, date, time, nz, nc):
    # 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")

# Calculate the total envelope exfiltration for each Contaminant.
def calcEnvExfil(cxl : cxLib, totalEnvExfil):
    nCtm = cxl.nContaminants
    for ic in range(nCtm):
        for path in cxl.envPaths:
            mass = cxl.getEnvelopeExfil(path.envIndex, ic)
            totalEnvExfil[ic] = totalEnvExfil[ic] + mass

# Write envelope exfiltration for each envelope flow path.
def writeEnvExfil(file, header, cxl : cxLib, date, time, ctmNum):
    np = cxl.nEnvPaths
    if header == True:
        file.write(f"Day\tTime")
        for path in cxl.envPaths:
            file.write(f"\t{path.nr}")
```

(continues on next page)

(continued from previous page)

```
        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")

# Write airflows to file.
# Paths, DuctsTerminals, and DuctLeaks.
# Paths include simple AHS (Rec, OA, Exh).
def writeAirflowRates(file, header:bool, cxl:cxLib, date:int, time:int):
    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 == True:
        file.write(f"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\"Net flows kg/s\"\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):
            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 == True:
        file.write(f"Day\tTime")
        for control in cxl.outputControls:
            file.write(f"\t{control.name}")
```

(continues on next page)

(continued from previous page)

```
        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")
```

11 NIST Developer Notes

Calling hierarchy within `contamxpy.cxLib.setupSimulation()` (page 5):

```
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)
    }
}
```

11.1 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[]*

11.2 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*.

11.3 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
    }
}
```

11.4 `contamx-lib` Functions that Reference *AF_NODE*s

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;
```

(continues on next page)

(continued from previous page)

```
    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;
}
```

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

C

D

E

F

G

I

L

N

O

P

R

S

T

V

W

X

Y

Z