# PyHistory: A time-history and event package for time-dependent simulations.

Paul F. Dubois
Lawrence Livermore National Laboratory
dubois1@llnl.gov

Version 2
August, 1998

# PyHistory creates customized time-history data files and manages events.

There is usually too much data to store all the data every time step. The problem is to sample the data of interest and store it.

Different users have different requirements for the frequency and contents of history files. These may change *during* a run.

There is a general need for reacting to events and scheduling operations at odd times.

PyHistory's design is based on a very successful Basis package.



■
Paul F. Dubois
dubois1@llnl.gov

# The user specifies sets of named expressions which are to be sampled at specified times.

The sample values can be stored in memory or a variety of file formats.

Sample collection can be governed by user-specified conditions.

PyHistory can be used to schedule events and to react to problem conditions.



■
Paul F. Dubois
dubois1@llnl.gov

# Many calculations have an iterative step that can be thought of as “time”.

We measure progress with “cycles” and “time”.

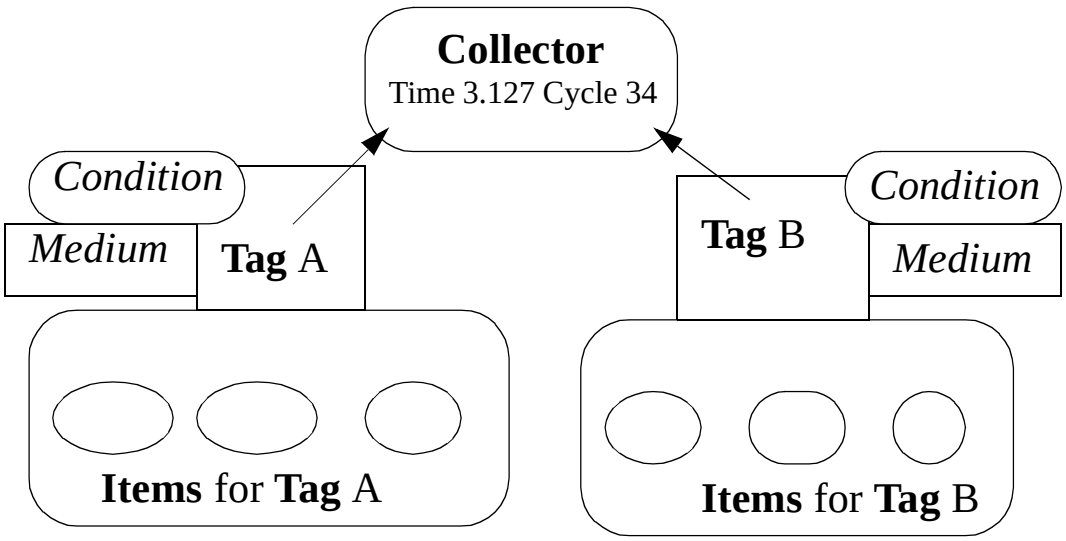
- A “cycle” is one iteration.
- There may be a natural quantity to think of as “time”. It can be any real number whose value increases strictly monotonically with each cycle. Usually it is the simulated time.



■
Paul F. Dubois
dubois1@llnl.gov

# PyHistory has three main concepts: items, tags, and collectors.

- **Item**: an expression and the context in which to evaluate it using the **eval** function, and a name for the resulting time history.
- **Tag**: a collection of items to be sampled at the same times, under the same *conditions*, with the resulting histories stored in the same *medium*.
- **Collector**: a manager of a set of tags that exist in the same "time" universe. The "time" that advances a collector may be any strictly monotonic sequence of values, but usually is the "time" of the simulation (default: **history.collector**).






■
Paul F. Dubois
dubois1@llnl.gov

# The user chooses a storage medium by using the appropriate tag-creating function.

- **textfile_tag (“filename”)**
  Histories stored as ASCII text, by record.
- **columnarfile_tag (“filename”)**
  Histories stored as labeled text columns
- **PDBfile_tag (“filename”)**
  Histories stored in Pact/PDB self-describing binary files.
- **event_tag (“x_big_enough”, “my_function (y, x)”)**
  Named event, whose sampling triggers a function call.
  Does not store histories.



■
Paul F. Dubois
dubois1@llnl.gov

# The text file output is suitable for simple data.

```
Record 1   3      0.03
       xx   1.5
       yy   2.25
       ww   3.1
       some_label'my label'
Record 2   5      0.05
       xx   2.5
       yy   6.25
Record 3   7      0.07
       xx   3.5
       yy   12.25
Record 4   9      0.09
       xx   4.5
       yy   20.25
```



■
Paul F. Dubois
dubois1@llnl.gov

# Columnar text file output may be useful as input to other programs.

| x | y | z |
|---|---|---|
| 0.2 | 0.12 | [0.2, 0.4472135955] |
| 0.4 | 0.28 | [0.4, 0.632455532034] |
| 0.6 | 0.48 | [0.6, 0.774596669241] |
| 0.8 | 0.72 | [0.8, 0.894427191] |
| 1.0 | 1.0 | [1.0, 1.0] |
| 1.2 | 1.32 | [1.2, 1.09544511501] |
| 1.4 | 1.68 | [1.4, 1.18321595662] |
| 1.6 | 2.08 | [1.6, 1.26491106407] |
| 1.8 | 2.52 | [1.8, 1.3416407865] |
| 2.0 | 3.0 | [2.0, 1.41421356237] |
| 2.2 | 3.52 | [2.2, 1.48323969742] |
| 2.4 | 4.08 | [2.4, 1.54919333848] |



■
Paul F. Dubois
dubois1@llnl.gov

# There is a set of tag operations that set the frequency and conditions of sampling.

***frequency* specifies start / stop / interval in cycles or time**

mytag.frequency (0, 100, 5)
mytag.frequency (0.0, 100.0, 5.0)

***at_times* and *at_cycles* select one or more specific times or cycles**

mytag.at_times (3.0, 5.6, 12.8)
mytag.at_cycles (5, 10, 12)

***when* sets a logical condition on sampling**

mytag.when (“x > 1.4”)



■
Paul F. Dubois
dubois1@llnl.gov

# There are five simple steps to using PyHistory.

1. Create a tag using the desired tag-creation function.
2. Specify the tag's sampling schedule.
3. Create the items desired in this tag.
4. Call the history collector at the end of each time step.
5. Make one special last call to the history collector.

**The user has a great deal of flexibility.**

Tags can be created, and their schedules changed, at any time.

Items cannot be added to tags once sampling has begun, ensuring that history files can be assumed to be sets of uniform records.

The package can be extended by the user in many ways.



■
Paul F. Dubois
dubois1@llnl.gov

# PyHistory example: storing the histories of two expressions in a PDB file.

**Step 1. Create a tag using the desired tag-creation function.**

- We use PDBfile_tag to make a Pact/PDB file to contain the histories.

**Step 2. Specify the tag's sampling schedule.**

- Sample every.01 seconds of simulated time between t = 0.0 and t = 10.0.
- However, sampling is to occur only when `x > 1.2`.

**Step 3. Create the items desired in this tag.**

1. Expression: the value of the variable `x`.
   Desired history name: `x`.
2. The value of the expression `hydro.y.x[2] / 1.e4`.
   Desired history name: `yx2`.

**Step 4. Call the history collector at the end of each time step.**

All the tags we have created have been registered with this collector. The collector supervises the activation of each tag at the appropriate times.

**Step 5. Make one special last call to the history collector.**

If appropriate to the sampling schedule, a "final value" is added to each history.



■
Paul F. Dubois
dubois1@llnl.gov

# PyHistory example: storing the histories of two expressions in a PDB file.

```
import hydro # the simulation!

# Step 1: Create a tag that uses PDB files.
from pdb_history import *
maytag = PDBfile_tag (“maytag”)
# Step 2. Specify the tag’s sampling schedule.
maytag.frequency (0.0, 10.0, 0.01)
maytag.when (“x > 1.4”)
# Step 3. Create the items desired in this tag.
maytag.item (“x”)
maytag.item (“hydro.y.x[2] / 1.e4”, “yx2”)
# Step 4. Call the history collector at the end of each time step.
for cycle in range (100):
    x = ....
    time = hydro.advance (x)
    collector.sample (cycle, time) # call this every cycle
# Step 5. Make one special last call to the history collector.
collector.sample_final (cycle, time)
```



■
Paul F. Dubois
dubois1@llnl.gov

# The collector's method "sample_final" handles sampling at the last time point of interval conditions.

Note the difference between **sample** and **sample_final**:

**collector.sample** (cycle, time)
Sample every item whose condition is true.

**collector.sample_final** (cycle, time)
Sample every item whose *final* condition is true.

Time- or cycle-based interval conditions have a "final" condition that is true if the stop time has not yet been reached. This ensures that a sample will be made at the last time in such cases, even if the last step does not occur at the interval time.

Both methods are called on the last cycle. PyHistory is smart enough not to duplicate the sample at the last time.



■
Paul F. Dubois
dubois1@llnl.gov

# The collector has facilities for tag management.

The most frequently used facility is collector.check (), used to verify that items have been specified correctly.

You can get a reference to a tag from the collector, manage the order in which tags are handled each cycle, or even delete a tag that it no longer wanted.

**collector.tag** (tagname)

Get the tag named tagname.

**collector.position** (tagname)

Get the position in the taglist of tagname.

**collector.tagnames** ()

Names of the tags in this HistoryCollector.

**collector.add** (tag, position = -1)

Add a tag to this history collector in the desired position.

**collector.delete** (tagname)

Remove a tag from this history collector.

**collector.check** (tagname = '')

Check each item in the tag(s) and print a list of those that throw an exception when collected. With no argument, checks all tags in the collector.



■
Paul F. Dubois
dubois1@llnl.gov

# PyHistory example, using text file output, and an “event”.

```
import hydro
# Step 1: Create a tag that uses text files.
from history import *
maytag = textfile_tag (“maytag”)
# Step 2. Specify the tag’s sampling schedule.
maytag.frequency (0.0, 10.0, 0.01)
maytag.when (“x > 1.4”)
# Step 3. Create the items desired in this tag.
maytag.item (“x”)
maytag.item (“hydro.y.x[2] / 1.e4”, “yx2”)
# Step 3a. If x gets bigger than 20.0, call a function “cool_off ()”.
#               (Test the condition every 5 cycles.)
def cool_off ():
        hydro.y= x / 2.0
myevent = event (“problem too hot”, “cool_off ()”)
myevent.frequency (0, 10000, 5)
myevent.when (“x > 20.0”)
# Step 4. Call the history collector at the end of each time step.
for cycle in range (100):
        x = ....
        time = hydro.advance (x)
        collector.sample (cycle, time) # call this every cycle
# Step 5. Make one special last call to the history collector.
collector.sample_final (cycle, time)
```



■
Paul F. Dubois
dubois1@llnl.gov

# PyHistory allows user extensions through the use of inheritance.

**New media:**

- You inherit from HistoryMedium and redefine a few features such as *begin_record*, *end_record*, and *_write*.

**New conditions:**

- You inherit from HistoryCondition and create a new kind of condition, such as being true every year except Leap Year.

**New user conveniences:**

- You inherit from Tag to simplify use of your new condition and medium.
- See following example of adding specific times to a cycle interval specification.



■
Paul F. Dubois
dubois1@llnl.gov

# Users can also make more sophisticated use of PyHistory's existing classes.

## Create fancy sampling schedule

Here is an example of a tag sampled every 10 cycles *and* at four specific times.

```
from history import *
import HistoryCondition
c1 = HistoryCondition.Cycles (0, 1000, 10)
c2 = HistoryCondition.TimeList ([0.1, 1.0, 10., 100.])
c3 = HistoryCondition.Or (c1, c2)

tag1 = textfile_tag ("tag1")
tag1.set_condition (c3)
```

## Use multiple collectors

- If a program contains more than one iterative process, you can create a collector for each "time universe".
- An optional argument to the tag-creation functions chooses the collector.



■
Paul F. Dubois
dubois1@llnl.gov

# PyHistory example: inventing a new kind of condition.

```
class EmptyList (Condition):
    “A condition that is true if a certain list is empty.”
    def __init__ (self, the_list):
        Condition.initialize (self)
        self.set (the_list)
    def set (self, the_list):
        self.target = the_list
    def _value (self, cycle, time):
        "Is this true at the time given by the clock?"
         t = (len (self.target) == 0)
        return (t, t)

mytag.set_logical_condition (EmptyList (somelist))
```



■
Paul F. Dubois
dubois1@llnl.gov

# PyHistory has been released for general distribution.

## The LLNL Python Distribution

The easiest way to get PyHistory is by obtaining the “LLNL Python distribution” located at:

ftp-icf.llnl.gov
/pub/python/LLNLDistribution.tgz

PyHistory is located in the Lib/history subdirectory. If you wish to use PDB histories, you also need to build and install Pact and the Python PDB extension.

## Documentation

The document is at this Web site:

`http://xfiles.llnl.gov/PyHistory`

This tutorial is part of that site.



■
Paul F. Dubois
dubois1@llnl.gov