

Setting up TAP for your own location

NOAA HAZMAT *

April 23, 2008

Contents

1	Introduction	2
2	Background	2
2.1	Trajectory Model	2
3	Directory structure	3
3.1	The map	4
4	File formats	4
4.1	BNA format	4
4.2	SITE.TXT	5
4.3	The cube files	8
4.3.1	Numerical details of the cube files	8
4.3.2	Cube filenames	9
5	Batch GNOME	10
5.1	Running GNOME in Batch Mode	10
5.1.1	Using an Error Log	11
5.1.2	The commands	11
5.1.3	An example COMMAND.TXT file	13
5.2	The TAP Trajectory File Format	15

*Hazardous Materials Response Division, Office of Response and Restoration, National Oceanic and Atmospheric Administration, 7600 Sand Point Way NE, Seattle WA 98115, USA. Email: ORR.TAP@noaa.gov.

5.2.1	The Header	15
6	TAP Scripts for Automating Batch GNOME	17
6.1	Installation	17
6.1.1	The Python interpreter	17
6.1.2	Editor	18
6.2	Installing HazmatPy	18
6.2.1	Windows Binary Version	18
6.2.2	Compiling HazmatPy	19
6.3	The Hazmat TAP Scripts	20
6.3.1	The TAP_Setup.py file	20
6.3.2	The Procedure	21

1 Introduction

This document provides a reference about the inputs required by the TAP Viewing Engine. It seeks to provide all the information required to produce a TAP location with any trajectory model. Please let us know if any information in this document is missing or unclear, so that we can improve it.

2 Background

TAP is both an application and a methodology for planning. The methodology involves running an oil spill trajectory model many times, driven by historical data, to form an ensemble of possible oil spill trajectories that we hope represent the possible behavior of future potential oil spills. The results of all those model runs are then examined in the TAP Viewing Engine, which generates maps and statistics that can aid in spill response planning. This same data could be analysis and viewed in other ways, such as a GIS.

2.1 Trajectory Model

In order to generate the data for TAP, first you need to set up an oil spill trajectory model for the region of interest. GNOME is one such model, and a good choice for TAP, as we can provide guidance on its use. Virtually any other model could be used, with some work on translating output formats.

The other requirement for TAP is a data set of historical winds and currents with which to drive the trajectory model. The goal is to be able to hindcast enough of the winds and currents in your region of interest to capture the range of behavior of possible spills in the region.

Wind data is widely available in many regions. We feel that at least 10 years or so is required. You do need to be careful that the measured data reflects your region of interest. This can be an issue if the measurements were taken far away or in regions with strong topographic steering of the wind.

Currents in tidally dominated regions can be modeled by a wide variety of oceanographic models. Currents on the open coast can be far more challenging. Coastal currents can be highly variable and driven by complex forcing, often feeling the effects of forcing far from the region of interest.

In order to hindcast such currents, a hydrodynamic model of the region would need to be applied, with a way to provide boundary conditions for a substantial time period in the past. There may be oceanographers that study the region of interest that might be running such a model, or be able to assist you in getting it set up.

Once you've got your wind data and current model, you can set up GNOME, or your trajectory model of choice, to use those currents and your wind data to model the movement of oil in the area. You can then run GNOME in a batch mode to create a set of trajectories from random times in the past, and then process these to create the data the TAP viewer needs.

You may need to do some translating of file formats so that GNOME can read the output from your current model, but we have used GNOME with many of the major ocean current models, so we should be able to help.

3 Directory structure

The TAP Viewing Engine finds the files it needs by looking in specific places. In the same directory as the TAP application, there must be a directory (folder) called TAPDATA. This directory has all the data TAP needs to run. In this directory are three items:

- A Map file (BNA format).
- The SITE.TXT file. This file includes a large set of data about the site.

- A set of directories, one for each season represented, that hold the binary cube data.

In addition, there can optionally be a file in the same place as the executable called: `TAPCONFIG.TXT`. In this file can be commands that overwrite some of the defaults in the program.

3.1 The map

Each TAP location must have a base map that is used as the background for the display. The map file provides a set of polygons surrounding the land, in decimal latitude-longitude coordinates, in the BNA format (It can also be in the PICT format, but that would only work on a Macintosh). See section 4.1 for a description of this file format.

4 File formats

4.1 BNA format

The BNA format is used to describe polygons in lat-long coordinates. In TAP, it is used for the base map, and to describe the receptor sites. The BNA format is a ASCII comma-delimited text format that lists polygons one by one. Each polygon is composed of a header line, and a set of coordinates. The header line looks like:

```
"Polygon Name", "1", num-points
```

The first two items are inside double quotes. The first item is the name of the polygon. It is not used in the map file by TAP. The second item is a code that indicates something about the polygon. TAP uses two codes: "1" is a land polygon, and "2" is a water polygon that is on top of land (lakes). The third item is an integer (not quoted) that is the number of points in the polygon. Each point is represented by longitude, latitude, in decimal degrees. The western hemisphere is negative longitude, and the southern hemisphere is negative latitude. The polygons are all drawn as closed (and filled) polygons. The first and last points should be the same, but if they are not, TAP will close the polygon. If you want a polyline, rather than a filled polygon (bridges, etc.) then you need to create a zero area polygon by using separate single line segments, or doubling back on the line so that the polygon has zero area. Figure 1 is a sample of a small BNA file.

```

"Carqueniz Bridge","1",2
-122.22566667,38.0648333
-122.225,38.056333
"Pablo Bridge","1",4
-122.40916667,37.933833
-122.452333,37.9385
-122.4798333,37.944333
-122.452333,37.9385
"1741","1",8
-122.095757,37.453915
-122.097054,37.454048
-122.097580,37.454239
-122.096130,37.456726
-122.095413,37.455772
-122.095009,37.455418
-122.094810,37.455116
-122.095757,37.453915

```

Figure 1: A Sample BNA file. In this example, the second item is drawn as a polyline, because the last points doubles back on the second point.

4.2 SITE.TXT

The SITE.TXT file provides most of the data used to set up a TAP location. The SITE.TXT file included with the TAP demo is fairly clearly commented. There should be no blank lines in the file.

The header of the file is whitespace delimited, with all text strings in double quotes. In the SITE.TXT file, any information following a “//” is a comment, and ignored by the program. The first line in the file is:

```
"SAN FRANCISCO BAY"
```

it is the name given to the region. The second line is:

```
"SF.BNA" "BNA"
```

The name of the map file, and its type. BNA is recommended, Pict is the other option.

```
500 SPILLS
```

The number of individual spills modeled (individual scenarios). This has to be the same number in every cube.

2 SEASONS

The number of seasons modeled. This can be one or more

```
"Mar - Aug" "SFMS" "Summer"  
"Sept - Feb" "SF_W" "Winter"
```

The description and name of the seasons. One line for each season. The first field is what gets displayed to the user in the pop-up menu. The second field is the prefix of the filenames of the cube files, so that, in this example, the filenames of the Summer cube files are: SFMS0001.bin, SFMS0002.bin, and so on. The third field is optional, and is the name of the directory that the cube files are stored in. Each season is stored in a separate directory. If this field is omitted, the directory name is taken to be the same as the first field (what the user sees). Note: make sure that this is a valid filename.

10 TIMES

The number of output times in each cube.

```
120 "5 days"  
96 "4 days"  
72 "3 days"  
60 "2.5 days"  
48 "2 days"  
36 "1.5 days"  
24 "1 day"  
12 "12 hours"  
6 "6 hours"  
1 ""
```

The times (from the release) of the output data. The first field is in hours. The second field is the string that is presented to the user on the pop-up menu. If the second field is an empty string, the data will not be available to the user. These times should be presented in reverse order (largest first).

1 AMOUNTS

The number of preset spill volumes set for the user.

1000 barrels

The preset volumes set for the user (can be more than one. see the above line. The allowed units can be any of: **barrels**, **gallons**, **metric tons**, or **cubic meters**. The units are not case sensitive.

3 LOCS

The number of preset Levels of Concern provided in a pop-up menu for the user.

10 barrels

5 barrels

1 barrels

The volumes preset for the LOC menu. Possible units are the same as above.

185 SITES

The number of receptor sites.

"#1", "1", 19

-122.5306473, 37.8214531

-122.5305634, 37.8162155

...

The polygons of the receptor sites. The polygons are in BNA format (see Section 4.1 There should be one polygon for each receptor site. By convention they are labeled as “#1”, “#2”, etc., but you could use any text label you want.

After the receptor site polygons is the following line:

233 CUBES

This line indicates the number of data cubes (one for each spill start location).

-122.410448, 37.863448

-122.386002, 38.021599

-122.462997, 37.833302

...

The following lines are the locations, in comma separated values in decimal degrees of Longitude, Latitude, of the spill locations, one for each cube. It is assumed that the spill locations are the same for all seasons.

4.3 The cube files

The receptor site hit data is stored in binary files known as the cube files. There is one cube file for each spill start location, for each represented season. We call them “cubes” because the data can be thought of as a 3-d array of numbers. The axis are:

Time One point for each of the output times specified in the `SITE.TXT` file. We usually use ten output times.

Sites One point for each receptor site, in the order specified in the `SITE.TXT` file.

Spills One point for each realization. Each “spill” is a single modeled realization of an oil spill from the given spill location. The differences between the spills are a result of different environmental conditions (winds, tides, etc.). We usually use 500 spills, which are each started at a different time, randomly sampled from the records we have available. Each of the cubes should have the same set of start times, to keep the results consistent.

Each cube is then a 3-d array, with each element in the array corresponding to the amount of oil that has passed through a given receptor site, by a given time, for a given realization. The amount recorded is cumulative, so that oil that hits one site, and then re-floats and moves on to a another site is counted at both sites, and the amount of oil at a site monotonically increases with time. If the same oil hits a site, moves away, and then comes back to the same site, it is not counted again.

4.3.1 Numerical details of the cube files

In order to minimize storage space, each value in the cube is stored in a single (eight bit) byte, as an unsigned integer (number between 0 and 255). This number is computed from the fraction of the entire amount of spilled oil at that site with the following exponential relationship:

$$x = \text{CEIL}(255f^{\frac{1}{1.5}}) \quad (1)$$

where x is a the number between 0 and 255 that is stored in the cube file, f is the fraction of the total spill that has passed through the site by the given

time, for the given spill, and the CEIL operator is the “ceiling” operator which rounds up to the next integer. CEIL is used in order to give a slightly more conservative result than ordinary rounding would. This exponential relationship is used in order to give better resolution for small amounts of oil than large. For example, the distinction between one and two barrels impacting a site is likely to be more important than the distinction between 50 and 51 barrels impacting a site. In a Lagrangian element model, f is the number of LEs that hit the site, divided by the total number of LEs used in the model run.

Order of the data in the file While we like to think of the cubes as 3-d arrays, and while many computer languages let us store and access them that way, the data must be put linearly into the file. The sequence of data in the file is what is produced by the following pseudo-code:

```
for Time in Times:
  for Site in Sites:
    for Spill in Spills:
      WriteToFile(Cube[Spill,Site,Time])
```

Each value is a single eight bit byte, as described above.

We use a program written in the Python programming language (<http://www.python.org>) to generate these cubes. Please contact us if you would like a copy to adapt for your own use.

4.3.2 Cube filenames

The names of the cube files are a combination of a four letter prefix (defined in the `site.txt` file) and the number of the cube, padded on the left with zeros to four digits, and the `.BIN` extension. This results in an “8.3” filename, which should be valid on any system. By padding the numbers with zeros, the cube files can be alphabetically sorted correctly. For example, if the prefix is defined as “PRFX” in the `SITE.TXT` file, the cubes would be named: `PRFX0001.BIN`, `PRFX0002.BIN`, etc.

There is a complete set of cube files, one for each start site, for each season modeled. A given numbered cube should correspond to the same start site in all seasons.

5 Batch GNOME

If you want to use GNOME as the oil spill model to generate TAP cubes, you'll need to perform the following steps:

1. Setup a GNOME Save file for your region that includes historical data for the time period you are covering. This should always include wind data, and often also tides, river flow records, and any other scaling factors used in the Save file.
2. Set up a GNOME command file (or a set of them, if running on multiple machines).
3. Run GNOME in batch mode to generate your trajectories.
4. Run `BuildCubes.py` (small custom program written in the Python Programming language: www.python.org) to generate your cubes. It is available from `ORR.TAP@noaa.gov`
5. Generate a `TAP SITE.TXT` file for the TAP viewer, and put the Cubes generated in the appropriate place in the `TAPDATA` folder

Setting up the GNOME save file is documented elsewhere.

NOAA/Hazmat has developed a series of Python scripts that will do much of this for you. Those scripts are documented in Section 6.

5.1 Running GNOME in Batch Mode

The `COMMAND.TXT` file allows generation of Splot File series to be automated. A Command file is a text file with GNOME commands. The first line of a Command file must be:

```
[GNOME Command FILE]
```

Primary method Launching GNOME with a command file in the GNOME directory:

If there is a Command file called `COMMAND.TXT` in the GNOME directory when GNOME is launched, GNOME will execute the commands in the file and then quit.

Secondary Method Opening/Executing command files from within GNOME:

The Windows version of GNOME also supports the execution of a command file through the normal “Open” file. If you open a file which is recognized as a GNOME command file, GNOME will execute the commands in the file. Note that GNOME will not automatically quit in this case.

The Macintosh version of GNOME also supports the execution of a command file through the normal “Open” file, but in order to use this feature you will have to set the file’s creator to GNOME’s signature COSM and the file’s type to .SAV . Not having the file’s type be TEXT make this method somewhat cumbersome. An alternative method is to use drag and drop of a command file. In order to get the system to allow you to drop the file onto GNOME, you must make the creator of the text file be GNOME’s signature ‘COSM’ but you can leave the file’s type as TEXT. GNOME will execute the commands in the file, but will not automatically quit.

5.1.1 Using an Error Log

Under typical GNOME usage, any Warning, Error or Note alerts would stop the model and wait for user to hit the OK button. To prevent this behavior, you can specify an Error Log file. If an error log file is specified, the text of the alert will be appended to the error log file. In the case of Warning and Note alerts, the command file continues to execute. In the case of an Error alert, execution of the command file is halted.

5.1.2 The commands

In typical usage, the first part of your command file will setup the data for the scenario(s) you wish to run. There are two ways to do this:

1. Use a command to open a GNOME save file.
2. Use a series of commands of the same type used to setup a location file.

In the second part of the command file, you would typically include one or more lines of the RUNSPILL message which runs a scenario and outputs the LE Files.

The RUNSPILL message Here is an example message which includes all the required parameters:

```
MESSAGE runSpill;T0 model; startRelTime 19,07,2000,17,00;
runDurationInHrs 18;timeStepInMinutes 30; numLEs
1000;startRelPos 123.7667 W 46.21667 N;outputStepInMinutes
60;outputPath :Trajectories\Start01\Time001.bin;
```

The `outputPath` parameter specifies the relative path of the generated Trajectory file. Path elements are separated with Windows-style backslashes (\). If the given path is a file, a TAP Trajectory File (Sec. 5.2) will be created. If the given path is a directory, a set of binary LE files will be created, one for each time step. That format is documented elsewhere. The TAP Trajectory File format is much more compact, and the only one we use for TAP.

Optional parameters for the RUNSPILL message:

`endRelTime`, `endRelPos`, `pollutantType`, `totalMass`, `massUnits`, `z`.

- The default for `endRelTime` is `startRelTime`.
- The default for `endRelPos` is `startRelPos`.
- The default for `pollutantType` is Non-weathering.
- The default for `totalMass` is `numLEs`.
- The default for `massUnits` is Barrels.
- The default for `z` is 0.0. Note: `z` is measured in meters.

`pollutantType` KEYWORDS:

- CONSERVATIVE or "Non-Weathering"
- BUNKER or "Fuel Oil #6"
- MEDIUMCRUDE or "Medium Crude"
- IFO or "Fuel Oil #4"
- DIESEL
- JP4 or "Kerosene / Jet Fuels"

- GAS or "Gasoline"

When there are two keywords for the same oil, the first is the one used in the MOSS files and the second is the one used in the GNOME interface. You may use either one in the command file. Note: These key words are not case sensitive.

massUnits KEYWORDS:

- BARRELS
- GALLONS
- CUBICMETERS
- KILOGRAMS
- METRICTONS
- SHORTTONS

Additional Messages:

- MESSAGE close; TO model;
- MESSAGE clearWinds; TO model;
- MESSAGE clearSpills; TO model;
- MESSAGE reset; TO model;

5.1.3 An example COMMAND.TXT file

[GNOME COMMAND FILE]

```
-- this would clear all maps etc in case we are not launching GNOME
MESSAGE close; TO model;
```

```
-----
-- Set the error log file
-----
```

```
MESSAGE setField; TO model; errorLog :MyFileName.txt;
```

```
-----
```

```

-- Either open a save file
-----
-- MESSAGE open; TO model; PATH yourRelativePathHere;

-----
-- Or use normal messages from wizard commands in location file
-----

MESSAGE setfield;TO model;timeStep 0.10;
--
MESSAGE createMap;TO model;TYPE vector; NAME CR Estuary Map;
PATH :Columbia River:Crest.bna;
--
MESSAGE createMover;TO CR Estuary Map;TYPE Cats;
NAME River Flow.CUR; PATH :Columbia River:River Flow.cur;
--
MESSAGE createMover;TO CR Estuary Map;TYPE Cats; NAME CR Tides.CUR;
PATH :Columbia River:CR Tides.cur
MESSAGE setfield;TO CR Tides.CUR; scaleType constant;
refP 123.7667 W 46.21667 N; scaleValue -1;
timeFile :Columbia River:Shio.txt;

MESSAGE createMover;TO Universal Map;TYPE Random; NAME Diffusion;
MESSAGE setfield;TO Diffusion; coefficient 200000;
uncertaintyFactor 2

-----
-- specific setup message not typically used in LocationFiles
-----
MESSAGE clearWinds; TO model;
MESSAGE createMover;TO Universal Map;TYPE Wind; NAME Variable Wind;
PATH :Columbia River:WindData.wnd;
MESSAGE setfield;TO Variable Wind; windage 0.02 0.05;

-----
-- specific messages for the TAP run
-----
MESSAGE runSpill;TO model; startRelTime 19,07,2000,17,00;
runDurationInHrs 18;timeStepInMinutes 30; numLEs 1000;
startRelPos 123.7667 W 46.21667 N;outputStepInMinutes 60;

```

```

outputPath :Trajectories\Start01\Time001.bin;

MESSAGE runSpill;TO model; pollutantType Kerosene / Jet Fuels;
totalMass 5000; massUnits GALLONS;startRelTime 19,08,2000,17,00;
endRelTime 19,08,2000,21,30; runDurationInHrs 18;
timeStepInMinutes 30;numLEs 1000;startRelPos 123.7667 W 46.21667 N;
endRelPos 123.7667 W 46.24N; outputStepInMinutes 60;
outputPath :Trajectories\Start01\Time001.bin;

-----
-- if we wanted to quit
--MESSAGE quit; TO model;

```

5.2 The TAP Trajectory File Format

Running GNOME in batch mode generates a set of TAP trajectory files, which are put in the location specified by the `outputPath` parameter to the `runSpill` message in the `command.txt` file. If you are using NOAA's code for reading these files, you don't need to know the details, but it could be helpful for debugging, or if you need to customize the code or write your own.

The trajectory file format is a binary format with a text header. This allows the user to view the header in any text editor, and examine the parameters used in the run. The data itself is binary, so that it can be more compact, and be written and read faster.

5.2.1 The Header

The header is in ascii text format, with DOS/Windows style line endings (`\r\n`), regardless of the system on which GNOME was run. This allows the files to be transferred between systems, and still be read with any text editor.

The header has a tagged format, with each section having a tag inside square brackets (`[]`). Following the section tag is a set of key:value pairs separated by a colon. Following the `[BINARY DATA]` (and a `\r\n`) is the binary data itself. The sections and data are fairly self explanatory, and best illustrated with an example:

```

[FILE INFORMATION]
  File type: TAPRUN
  File version: 1
[RUN PARAMETERS]
  Model start time: January 29, 2000 23:00

```

```
Run duration: 240 hours
Output time step in seconds: 720
Number of output steps: 1201
Number of LEs: 1000
GNOME version: 1.2.5
GNOME platform: Windows
Note:
[BINARY FORMAT]
  Endian: little
  Longitude: LONG
  Latitude: LONG
  Bit flags: CHAR notReleased beached offMaps evaporated notOnSurface
[BINARY DATA]
(binary data here)
```

Most the key:tag pairs should be clear in meaning from the example. The [BINARY FORMAT] section is defined as follows:

Endian: indicates the endian format of the binary data. For example, PPC processors uses big endian, and Intel processors use little endian. If the file is processed on a different type of machine than GNOME was run on, the data will need to be byteswapped.

The rest of the data describes the structure of the binary data. For each LE, for each time step, the following data is stored: Latitude, Longitude, assorted flags. Latitude and Longitude are stored as integers, representing micro degrees. To convert to floating point degrees, divide by $10e-6$. This format allows us to get plenty of resolution with minimal storage. Each structure totals 9 bytes:

- Longitude as a 4 byte C long
- Latitude as a 4 byte C long
- Flags as a 1 byte C char

The meaning of the flags is specified in order, so that:

flag & 1 is true means that the LE has not been released

flag & 2 is true means the LE is beached

etc.

Where & is the bitwise and operator.

The struct for each LE is written out in sequence, all LEs for each time step, followed by the next timestep, etc. There are therefore a total of $9 \times \text{Number of LEs} \times \text{Number of Output Steps}$ bytes of binary data.

6 TAP Scripts for Automating Batch GNOME

NOAA/Hazmat has developed a series of scripts, written in the Python language (www.python.org), that facilitate setting up GNOME command files and building the TAP `site.txt` file and TAP cubes.

6.1 Installation

6.1.1 The Python interpreter

The Python interpreter must be installed to run these scripts. Python is open source and freely available. On Mac OS-X and many Unix systems, it comes with the system software. On Windows it must be downloaded and installed separately. There are a number of sources of information about Python on the Internet, beginning with the Python home page: www.python.org. That page is the best source for documentation and information about the language. There are also many published books about Python. While, strictly speaking, you don't need to know anything about the language to run the scripts, a basic understanding of the syntax will be helpful, as some scripts do need modification.

One of the strengths of Python is its extensibility. One of the extensions available is called `numpy`, and it is used heavily in the Hazmat code, so it must also be installed.

Windows There are a number of sources of installers for Python for Windows, starting with the Python home page. You can also install Python and the Numeric extension separately from:

- <http://www.python.org/download/>
- <http://www.scipy.org/Download>

You should use the latest stable versions (Python2.5.2 and Numpy 1.0.4 as of this writing).

OS-X Apple ships recent versions of OS-X with the Python interpreter included. You should use at least OS-X version 10.4, as previous versions had python configured differently, and used an older version. With OS-X version 10.5 (Leopard), you can use the built-in python. ON 10.4, you should use the installer and associated libraries found here:

- <http://pythonmac.org/packages/py25-fat/index.html>

Unix / Linux If a recent version of Python was not supplied with your system, it, and the numpy package, can be downloaded from the Python and numpy web sites:

- <http://www.python.org/download/>
- <http://www.scipy.org/Download>

Get the most recent stable versions.

6.1.2 Editor

You will need a programmers text editor to edit the config files and python code. Windows Notepad or Mac TextEdit will do in a pinch, but a more full featured editor is a good idea. It is particularly nice if it has a Python mode. If you are already have a favorite, the chances are good that there is a Python mode available. If you don't currently have a favorite programmers editor, SciTe is a good, simple choice for Windows or *nix:

- <http://www.scintilla.org/SciTE.html>

On Mac OS-X, TextWrangler is a good, free choice. Jedit, Peppy, or Komodo are other good choices.

6.2 Installing HazmatPy

Before Using the TAP scripts, a library of routines must be installed into the python installation on your system. Most of the code in these routines is straight Python code, and is just ready to use. However, a few routines are written in C and must be compiled for your system.

6.2.1 Windows Binary Version

As Windows does not provide a standard “system compiler”, compiling the extensions yourself can be a bit difficult. If you're lucky, you may be able to simply install a binary distribution. It will look something like this:

`Hazmat-1.0.win32-py2.3.exe`

It is a Windows installer. It needs to have Python already installed to work.

If you are running a different version of Python that the binary version was compiled against (Python2.3 in the above example), you can compile it yourself from the HazmatPy source distribution, as directed below.

6.2.2 Compiling HazmatPy

Windows: MSVC Windows does not come with a system compiler, so you must have a compiler installed in order to compile the extensions. The Binary Python distributions recommended above are compiled with MS VisualC++ v7. As far as I can tell, extensions must be compiled with the same version of MSVC, that is, version 7.0. The command line tools must be properly installed for MSVC.

Windows: MinGW MSVC is the easiest option on Windows (if you have the right version). However, an alternative is to use the free MinGw compiler (<http://www.mingw.org>) instead. The process for doing this can be found at: <http://www.mingw.org/MinGWiki/index.php/Python%20extensions>

Once MinGw is installed and working, and the python lib has been patched as suggested, you can build the package with:

```
python setup.py build --compiler=mingw32
```

and install it with:

```
python setup.py install --skip-build
```

A binary installer for windows can be created with:

```
python setup.py bdist_wininst --skip-build
```

Linux/ Unix If your Python installation was built for your system, it should be all set to go with the native compiler, usually gcc.

Mac OS-X You will need the latest version of the complete OS-X Developer Tools installed. I think they are now called XCode tools. They can be downloaded for free from the Apple Developers Connection web site.

All platforms Python comes with a built in system for configuring, compiling, and installing extensions to the python interpreter, called distutils. It makes it very easy to compile and install packages on a properly configure system.

1. Start up a command line interpreter (“DOS box” on Windows, “Terminal” on OS-X).
2. change directory to the HazmatPy directory.
3. type

```
python setup.py build
```

There may be a few warnings, but it should build successfully.

4. To install, type `python setup.py --install`. On OS-X, you may need to put a `sudo` command before the python command, as it is being installed into system directories.

6.3 The Hazmat TAP Scripts

Hazmat has developed a series of Python scripts to help run TAP. They are:

BuildStartTimes.py a script that examines a set of time series files used by GNOME, and computes a set of valid start times that do not overlap with any of the “holes” in the time series file.

BuildCommandfiles.py A script that computes a set of command files for Batch GNOME.

BuildCubes.py A script that builds a set of “cubes” for the TAP viewer from the trajectories output by GNOME.

BuildSite.txt.py A script that builds the `site.txt` file for the TAP viewer.

All of these scripts load all the necessary config data from the single config file: `TAP_Setup.py`. This is a python file, rather than a regular text file for two reasons:

1. A parser didn’t have to be written for a custom text file format.
2. You can use the full Python syntax and language to specify data structures and do computations in the file.

For example, if you have lat, long data in (degrees, minutes) format, rather than decimal degrees, you can simply put, for example:

```
lat = 38 + 12/60.0
```

and the conversion is done for you when the file is imported.

6.3.1 The TAP_Setup.py file

The `TAP_Setup.py` contains all the information required to set up and run GNOME and the various scripts required to set up a TAP. There should be no need to edit any other files. The file is full of comments (denoted by a `#`), and examples that should serve to explain the contents. Some familiarity with Python syntax and the workings of TAP will be helpful.

6.3.2 The Procedure

1. Edit the `TAP_Setup.py` file to suite your situation. The scripts should have been delivered with a test package that should be all set to go. The file is fairly well commented. When in doubt, leave the defaults as they are set, you can always go back and change anything later. Also, it's a good idea to comment out existing settings, rather than delete them (to a point), so that you retain the examples. A Python-aware editor is quite helpful for this.
2. Run `python BuildStartTimes.py`. This should create a set of start times files, one for each season.
3. Run `python BuildCommandFiles.py`. This will create a set of `Command.txt` files for GNOME. If you are running one single processor machine, and want to do it as one big job, then you only need one. This is set in `TAP_Setup.py`, with the `NumTrajMachines` parameter.
4. Start Up GNOME, and open the command file you want to run, with the file-open menu item. You may have to change the file select dialog to show you "all files", to see the command file. GNOME must be in diagnostic mode to run command files. If it is not in diagnostic mode, go to the file-preferences menu item, select the "mode" tab, then click "diagnostic" for both options, and then click "OK".

After selecting the command file, GNOME should start running in batch mode. It will only display a little bit of what's going on on the screen, but the window will be locked, and can not be minimized or moved. You should be able to use other programs, however (at least on Windows XP). On Windows, you can run more than one instance of GNOME at once if you have the most recent version. This can be helpful if you have more than one processor, or for testing.

When GNOME is done, it will have created a set of trajectory files in the location specified in `TAPSetup.py`.

5. Run `python BuildCubes.py`. This will build TAP cubes from the trajectories generated by GNOME and the grid definition (or polygon receptor sites) defined in `TAPSetup.py`. They will be stored in the location specified in `TAPSetup.py`.

Note that multiple copies of `python BuildCubes.py` can be run at once, and the script will test to see what's already been built. This can be useful if you are building many cubes and want to make use of multiple machines and/or multiple processors on one machine.

6. Run `python BuildSite.txt.py`. This script creates a `site.txt` file for the TAP Viewer that matches all the parameters in `TAPSetup.py`.
7. Run `python BuildViewer.py`. This will copy the `site.txt` file and Cubes to the appropriate place for the TAP Viewer to use them. You must also have `TAP.EXE` in that directory.
8. Run `TAP.EXE`. It should run with your new dataset.