문서번호 APCC-출장-16-0050(2016-01-25)

TRAVEL REPORT FORM
출장보고서

결재	선임연구원	팀장	부서장	원장
	01/24 김광형	01/25 김옥연	01/25 김형진	01/25 정진승
협조				

I. **Reporter** 보고자

Department 소속	Position 직위(직급)	Name 성명	Travel Date 날짜	Note 비고
연구본부	선임연구원	김광형	2016년1월10일 -18일	IRI, 뉴욕, 미국

II. **Major Activities** 주요업무 수행내용

1. Main Contents and Activities 주요내용 및 활동

- APCC-IRI 공동연구(BAWP 프로젝트) 관련 업무협의 및 기술이전

2. Relevance to APEC Climate Center's Activities 결론 및 소감

1) CAMDT에 대한 기술적 정보 및 잠재적 활용가능성 이해
2) CAMDT-Disease 컴포넌트 프로토타입 개발
3) BAWP 프로젝트 업무협의

[자세한 내용은 첨부된 출장보고서(IRI 출장보고서_20160118.hwp) 참조]

III. **References** 참고사항

(with attachment of any information or report in case of attendance of conferences, workshops and meetings) 학술대회, 워크숍, 회의 등 참석 시 관련 정보 및 문서 첨부

1. Presented and Collected Materials 주요 수집자료
   - 첨부1.CAMDT_user_guide_0116_2016.docx
   - 첨부2.CAMDT_EPICRICE_0115.py
2. Suggestions and Remarks 건의사항

IRI 출장 보고서

[2016.1.20. 김광형, 기후변화연구팀@연구본부/APEC 기후센터]

□ 출장개요

○ 출장자 : 기후변화연구팀 김광형

○ 출장지 : IRI, 뉴욕, 미국

○ 기　간 : 2016.1.10 - 1.18 (총 9일 중 이동시간 포함)

○ 목　적

- 출장 목적

1) CAMDT에 대한 기술적 정보 및 잠재적 활용가능성 이해 - 지난 3년여간 CAMDT를 개발해온 IRI의 전문기술(계절기후정보, 상세화, 작물모형, 운영 프로그램언어 등)을 집중 교육(자문)을 통해 단시간 내에 습득함으로써 병해충 위험도와 같은 잠재적 활용가능분야에 대한 APCC 독자적 연구수행이 가능하게 함

2) 현재 APCC에서 확보하고 있는 병해충모형(EPIRICE 등)을 활용해 CAMDT-Disease 컴포넌트 프로토타입을 개발하여 2월 기술워크숍에서 계절기후정보 Hindcast를 활용한 병해충 위험도 결과 발표

3) 기타 - BAWP 프로젝트 업무협의, 계절기후예측정보 상세화 기술에 대한 논의, 향후 프로젝트 방향성에 대한 논의 등

○ 출장을 통한 기대성과

- CAMDT와 같은 의사결정시스템을 활용한 APCC에서의 독자적 연구수행이 가능 (2016년 연구수행에 필수적임)
- CAMDT-Disease 프로토타입 개발을 통해 본 연구과제의 기

반 기술 및 예비 결과 확보 (향후 다양한 병해충모형을 활용한 CAMDT-Disease 컴포넌트의 개발이 가능)

- APCC 농업분야의 중장기 계획인 작물·토양·병해충 통합농업 모형 개발의 기반 기술 확보
- BAWP 프로젝트에 대한 전반적인 자료 확보
- BAWP 프로젝트 산출물 중 CAMDT의 향후 개발방향에 대한 의사결정에 참여함으로써, 사용자가 활용가능한 의사결정시스템을 개발하는데 기여

□ 출장일정

날짜	장소	일정
1/10	부산→뉴욕	이동 및 도착
1/11	IRI, 뉴욕	• IRI 관계자와 업무협의 • Understanding CAMDT I -System code interpretation (SCF to downscaling)
1/12	IRI, 뉴욕	• Understanding CAMDT II -System code interpretation (Downscaled SCF to DSSAT crop model)
1/13	IRI, 뉴욕	• Understanding CAMDT III -System code interpretation (Inputting various management options to DSSAT) • SCF Downscaling in the CAMDT -Understanding various downscaling skills
1/14	IRI, 뉴욕	• Development of CAMDT-Disease I -Coding the CAMDT-Disease component using Python
1/15	IRI, 뉴욕	• Development of CAMDT-Disease II -Coding the CAMDT-Disease component using Python
1/16	IRI, 뉴욕	• CAMDT-Disease simulation • Final discussion and Wrap-up
1/17-18	뉴욕→부산	이동 및 도착

□ 출장결과

○ CAMDT에 대한 기술적 정보 및 잠재적 활용가능성 이해

- 필리핀을 대상으로 하는 BAWP 프로젝트의 최종산출물 중 하나인 CAMDT(Climate-Agriculture Modeling Decision Tools)는 계절기후예측정보를 상세화 한 후 다양한 시나리오의 작물관리방법을 바탕으로 CERES-Rice 작물모형을 구동하여 예상되는 잠재적 생산수량 정보를 바탕으로 사용자들의 재배의사결정을 지원하는 의사결정도구임
- 본 출장을 통해 CAMDT에 적용된 전문기술(계절기후정보, 상세화, 작물모형, 운영 프로그램언어 등)에 대한 광범위한 교육과 분석을 통해 향후 새로운 작물이나 병해충 모형, 상세화기법을 활용한 APCC 독자적 연구수행 기반을 다짐
- 결과물: 현재 CAMDT 코드에 대한 자세한 설명 및 유저 가이드 작성 완료 (첨부1: CAMDT_user_guide_0116_2016.docx)

○ CAMDT-Disease 컴포넌트 프로토타입 개발

- CAMDT-Disease 컴포넌트는 기존 CAMDT의 개념에 병해충 계절전망 및 병해충 피해로 인한 생산량 손실 정보를 더함으로써 보다 현실적이고 종합적인 의사결정을 지원하는 추가적인 도구임
- 본 출장을 통해 기존 CAMDT 플랫폼에 병해충 모형인 EPIRICE 모형을 탑재하여, 주어진 계절예측정보로부터 상세화, EPIRICE 모형 구동, 병해충 위험도 결과의 표출로 이어지는 시스템 구축
- 본 시스템을 활용하여 2016년 2월 필리핀 현지에서 열리는 기술워크숍에서 CAMDT-Disease 컴포넌트의 프로토타입을 소개하고 과거 계절예측정보를 활용한 병해충 위험도 정보를 발표할 예정임
- 향후 CAMDT-Disease 프로토타입을 기반으로 추가적인 병해충 모형 및 상세화 기법의 탑재, 방제와 같은 의사결정 옵션의 강

화, 작물생육모형과의 통합을 통한 생산량 손실량 예측 등 추가적인 연구를 수행할 예정임

- 결과물: 파이썬과 R 코드로 이루어진 CAMDT-Disease 컴포넌트 코드 (첨부2: CAMDT_EPICRICE_0115.py)

○ BAWP 프로젝트 업무협의

- 2016년 2월 예정인 BAWP 프로젝트의 기술워크숍(Technical Workshop) 기획

(1) 워크숍 프로그램 확정

<table>
<tr><th>Time</th><th>2/17 (W)</th><th>2/18 (TH)</th><th>2/19 (F)</th></tr>
<tr><td>8-9am</td><td>-Registration</td><td rowspan="3">Practical exercise by groups on CPT, CC scenario methodology CAMDT and WEAP (I)</td><td rowspan="3">Each group prepare presentation of their exercise, highlighting how they interact with other tools (when applicable), concluding with a wishlist of how the KP present the results in operational mode</td></tr>
<tr><td>9-10am
10-11am</td><td>-Opening
-BAWP program updates
-Introduction about APCC</td></tr>
<tr><td>11am-12pm</td><td>-Break
-Overview of workshop (Amor)</td></tr>
<tr><td>12 - 1pm</td><td>lunch</td><td>lunch</td><td>lunch</td></tr>
<tr><td>1-2pm
2-3pm
3-4pm
4-5pm</td><td>- Introduction of Disease Model (1.5 hrs)

- Overview/review of CPT, CC scenarios, CAMDT, WEAP, KP (0.5 hr/tool)</td><td>Practical exercise by groups on CPT, CC scenario methodology, CAMDT and WEAP (II)</td><td>-Each group present their presentation to all
-All discuss the tools and their KP/Maprooms representation to refine wishlists</td></tr>
</table>

(2) 워크숍 첫째날 오전 시간대에 오프닝 이후 APCC에 대한 간단한 소개를 하고, 오후 시간대에 병해충모형에 대한 발표 진행 예정

(3) 초청한 병해충 전문가(Dr.Raymundo@UPLB, Dr.Trinidad @DA RFO5)에게 20-30분 정도 발표시간을 할애하여 병해충 모형 및 현지 발생상황을 설명할 수 있도록 할 것임

- 계절기후예측정보의 상세화 기법에 관한 논의: 현재 CAMDT에서 사용하는 상세화 기법은 2가지 종류를 사용하고 있음, 병해충 모형에 적합한 상세화 기법을 활용하기 위해 APCC에서 개발한 Weather generator를 이용한 상세화 기법과 기존 2개 방법과 성능평가를 통해 활용가능한 상세화 기법을 선정해야 할 필요가 있음, IRI 담당자와 함께 관련된 공동 연구를 수행하여 결과를 논문화 하기로 함
- 향후 프로젝트의 방향성에 대한 논의: 현재 IRI에서는 BAWP 프로젝트의 프로젝트 책임자로 한은진박사가 담당하고 있음, 따라서 본 연구결과의 활용성을 강화하기 위해 APCC 연구자가 한은진박사와 면밀한 협력관계를 통해 적어도 CAMDT에 관해서는 향후 2년여간의 방향성을 설정할 필요가 있음을 논의함. 보다 구체적인 내용은 2월 기술워크숍에서 논의하기로 함

□ 후속조치계획

○ CAMDT-Disease

- CAMDT-Disease 컴포넌트의 기술적 완성도를 높이고, 결합된 EPIRICE모형을 적절하게 보완하여 프로토타입 베타버전 완성 -> 2월 Tech. Workshop에서 소개
- IRI 계절기후예측정보에 대한 과거 hindcast자료를 확보하여 필리핀 비콜지역에 대한 주요 병해충 계절전망 생산하여 관측값과 비교를 통한 계절전망 활용성 평가
- APCC의 Weather generator를 이용한 상세화기술을 기존 IRI의 상세화기술과 비교하여 병해충 모형을 이용한 성능 비교
- APCC 계절기후예측정보에 대한 hindcast자료를 사용하여 IRI 계절기후예측정보와의 기후 예측성 비교 및 병해충 계절전망 예측성 비교

○ 2월 기술워크숍

- 기술워크숍 계획안 확정하여 예산집행 결재 완료 (늦어도 2월 초까지)
- 병해충 관련 외부인사 초청 및 발표 내용 조율, APCC를 포함 전체 발표 계획 안 완료, 발표자료 준비 (2월 첫째주까지)
- IRI와 필리핀 BAWP 팀과 협력하여 워크숍 venue 및 프로그램 확정 -> 기술워크숍 개최 (2월17-19일)

User-guide for CAMDT Python code

-Prepared by Eunjin Han[1] at IRI, Columbia University –
(Jan, 2016)

***What is Python?**

- Python is a programming language which is …..
- ✓ Simple and easy to learn
- ✓ Free and open source
- ✓ Cross platform – Mac, Windows, Linux….
- ✓ Interpreted – no need to compile to binary code
- ✓ **Object oriented**

*** In Python, everything is an object**

- A unique ID, or location in the computer's memory
- A set of **properties (**or **attributes)** that describe the object
- A set of **methods**, or things that the object can do

***What is Tkinter?**

- Tkinter is the Python interface to Tk, the GUI toolkit for Tcl/Tk

(source: Grayson, John E. "Python Tkinter Programming." (2011).

***What are Classes in Python?**

- A Python class is a user-defined data type
- A class provides the following object descriptions:
  - The attributes (data-members) of the object
  - The behavior of the object (methods)
  - Where behavior is inherited from other classes (superclasses)
- Calling the class as a function creates an ***instance*** of the class
- Initializing an instance: Fields (instance variables) of an instance may be initialized by including an __init__ method in the class body. This method is executed automatically when a new instance of the class is created. Python passes the instance as the first argument. It is a convention to name it *self*.

(source: Grayson, John E. "Python Tkinter Programming." (2011).

***Installing Python, PMW and (PythonWin)**

To run CAMDT, you need to have Python installed.

There are several ways to download Python, but I recommend to install the free Anaconda SciPy environment.

http://continuum.io/downloads

I want you to have Python2.7 version to avoid any compatibility issues between different versions.

Second, in order to better use TkInter (Python interface for GUI), you also need pmw[2] widget. You can have information about download and installalation here...

---

[1] Contact info: eunjin@iri.columbia.edu

[2] Pmw is a toolkit for building high-level compound widgets in Python using the Tkinter module.

http://pmw.sourceforge.net/doc/starting.html
After installing pmw, you will have a folder "pmw" under Anaconda\Lib\site-packages (in my case C:\Users\eunjin\AppData\Local\Continuum\Anaconda\Lib\site-packages\Pmw)

Third, It will be also helpful if you have PythonWin integrated development environment (IDE) to write code and debug (even though you don't need to write/debug...). [OPTIONAL!]
You can download PythonWin here (http://sourceforge.net/projects/pywin32/files/pywin32/Build%20219/). My laptop has Windows 7 - 64bit, so I downloaded 'pywin32-219.win-amd64-py2.7.exe'
==========

1) Installing Python by installing the free Anaconda SciPy environment
(https://store.continuum.io/cshop/anaconda/)

Default directory for Anaconda would be like this….
C:\Users\eunjin\AppData\Local\Continuum\Anaconda\Lib\site-packages

2) download pmw from http://pmw.sourceforge.net/doc/starting.html

And unzip the original file. Copy pmw folder to
C:\Users\eunjin\AppData\Local\Continuum\Anaconda\Lib\site-packages

3) To run the installation command "`python setup.py install` (as root)", I need to copy Python application file (both python.exe and python27.dll) to
C:\Users\eunjin\AppData\Local\Continuum\Anaconda\Lib\site-packages

4) Run the installation command "`python setup.py install` " on the DOS window which will install pmw automatically.
==========

***Installing PythonWin for the Anaconda-based Python** [OPTIONAL!]

1) download PythonWin (http://sourceforge.net/projects/pywin32/files/pywin32/Build%20219/) pywin32-219.win-amd64-py2.7.exe
2) Install the downloaded Pythonwin under the Anaconda directory (default)
3) Make a short-cut on Desktop (C:\Users\eunjin\AppData\Local\Continuum\Anaconda\Lib\site-packages\pythonwin.exe -> make a shortcut)

1. Import all required modules

CAMDT_July_2015

```
1   title = 'CAMDT User-Interface'
2
3   # Import Pmw from this directory tree.
4   import sys
5   sys.path[:0] = ['../../..']
6
7   from Tkinter import *
8   import Tkinter #Tkinter is the PYthon interface to Tk, the GUI toolkit ofr
9   import Pmw #Pmw(Python megawidgets) are composite widgets written entirely
10          #Pmw provide a convinent ways to add functionality to an appilcatio
11  import datetime    #to convert date to doy or
12  from tkFileDialog import *  # importing everyt
13   #Small addendum: tkFileDialog is renamed to tkinter.filedialog in Python
14  import subprocess  #to run executable
15  import shutil   #to remove a foler which is not empty
16  import os   #operating system
17  import numpy as np
18  import matplotlib.pyplot as plt  #to create plots
19  import fnmatch   # Unix filename pattern matching => to remove PILI0*.WTD
20  import os.path
21
22  class CAMDT:
23      sf = frame = None
24
25      def __init__(self, parent):
26
27          global startyear2
28          global endyear2
```

Import required modules

Define a class called CAMDT

Initializing an instance

2. Calling main program/module

```
2214  ##########################################################################
2215  # main program
2216  if __name__ == '__main__':
2217      root = Tkinter.Tk()  #The m
2218      #Setting application-wide d
2219      root.option_readfile('C:\\IRI\\PH\\CAMDT_G
2220
2221      Pmw.initialise(root)
2222      root.title(title)  #add title =
2223
2224      #Add a logo image as a button
2225      img = PhotoImage(file='C:\\IRI\\
2226      logo_button=Button(root, background='white', image=img).pack(side=TOP)
2227
2228      widget = CAMDT(root)  #Calling CAMDT class
2229
2230      #Add an Exit button to destroy the
2231      exitButton = Tkinter.Button(root,
2232      exitButton.pack()
2233
2234      #call Tkinter mainloop to process
2235      root.mainloop()
```

1) __name__

2) “root” toplevel

3) Change default fonts

The constructor initializes Pmw

4) Add a title on the main window

5) Add a logo image

6) Call “CAMDT” class

7) Add an “Exit” button to destroy the main window

8) Call mainloop to process events

1) A module's __name__

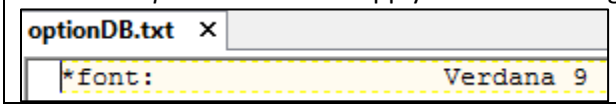
Every module has a name and statements in a module can find out the name of its module. When a module is imported for the first time, the main block in that module is run. What if we want to run the block only if the program was used by itself and not when it was imported from another module? This can be achieved using the `__name__` attribute of the module.
Every Python module has it's `__name__` defined and if this is `'__main__'`, it implies that the module is being run standalone by the user and we can do corresponding appropriate actions.
(source: http://ibiblio.org/g2swap/byteofpython/read/module-name.html)

2) Toplevel

The *Toplevel* widget provides a separate container for other widgets, such as a Frame. For simple, single-window applications, the "root" Toplevel created when you initialize Tk may be the only shell that you need.
(Source: Grayson, John E. "Python Tkinter Programming." 2011 p.32).

3) Change default fonts

Setting application-wide defaults for fonts or colors.
The purpose of this line is to set the font for all widgets except labels to Verdana 9 as indicated in a file called *optionDB.* We can apply the values using an *option_readfile* call.



4) Add a title on the main window

5) Add a logo image

6) Call "CAMDT" class

→ The "CAMDT" class has almost all components for the CAMDT GUI including creating pages, tabs and combox etc. to get user's input.

7) Add an “Exit” button to destroy the main window

8) Call mainloop to process events

Call the Tkinter *mainloop* to process events* and keep the display activated.
(Source: Grayson, John E. "Python Tkinter Programming." 2011 p.10).

***What are events?**
-Events are notifications (messages in Windows parlance) sent by the windowing system to the client code. They indicate that something has occurred or that the state of some controlled objet has changed, either because of user input or because your code has made a request which causes the server to make a change. (Source: Grayson, John E. "Python Tkinter Programming." 2011 p.96).

2. CAMDT class

2-1) First Page

```
22  class CAMDT:
23      sf = frame = None  #initially, no frame
24
25      def __init__(self, parent):  #initializing an instance
26          global startyear2   #declare global variables to take user's inputs
27          global endyear2
28          global startyear3
29          global endyear3
30          global CheckVar, CheckVar21,Check                    CheckVar51
31          global v_name
32          global planting_date
33          global Rbutton1,Rbutton2,Rbutton3,cul_Rbutton,Fbutton1,IRbutton
34          global group26,group35
35
36          # Create and pack a NoteBook.
37          notebook = Pmw.NoteBook(parent)
38          notebook.pack()
39
40   #=========================First page "Simulation Setup"
41          #Add a page to the notebook.
42          page = notebook.add('Simulation setup')
43          notebook.tab('Simulation setup').focus_set()
44         # page.configure(background = 'white')
45         # Create the "Simulation mode" conten
46         # 1) ADD SCROLLED FRAME
47          sf = Pmw.ScrolledFrame(page,usehullsize=1, hull_width=400, hull_height=220)
48          sf.pack(padx=5, pady=3, fill='both', expand=YES)
```

Declare global variables to take user’s input and write them into a param.txt

Use Pmw NoteBook widget which implements the popular property sheet motif. Methods allow a number of pages or panes to be created

1) Packer

Add a page to the notebook using “add” method

2) Add a ScrolledFrame

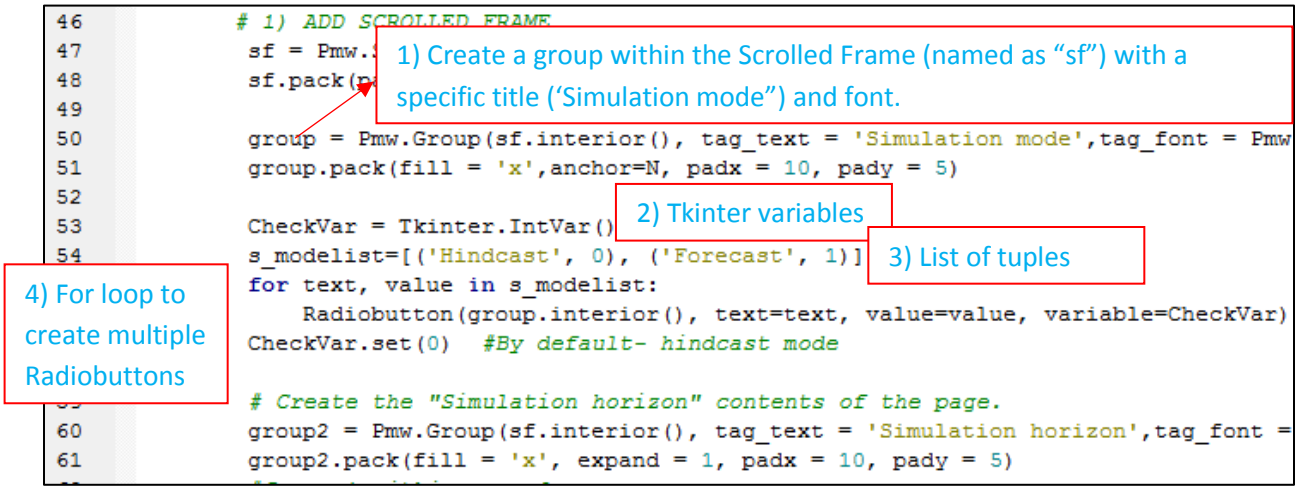
1) Packer

The Packer positions slave widgets in the master by adding them one at a time from the outside edges to the center of the window. The Packer is used to manage rows, columns and combinations of the two.
Examples:
-Side=LEFT tells the Packer to start locating the widgets in the packing list from the left-hand side of the container
-Expand=YES tells the Packer to fill the available space within its parcel; whether it does expand is controlled by the “fill” option.
-Using fill=BOTH allows the widget to use all of its parcel.
-The padx and pady options allow the widget to be packed with additional space around it.

(Source: Grayson, John E. "Python Tkinter Programming." 2011 p.79).

2) ScrolledFrame

The ScrolledFrame widget is to provide a Frame widget with associated horizontal and vertical scrollbars.
The scrollbars can be dynamic, which means that a scrollbar will only be displayed if it is necessary – if the frame is smaller than the surrounding clipping frame, the scrollbar will be hidden.
(Source: Grayson, John E. "Python Tkinter Programming." 2011 p.79).




```
46          # 1) ADD SCROLLED FRAME
47          sf = Pmw.
48          sf.pack(p
49
50          group = Pmw.Group(sf.interior(), tag_text = 'Simulation mode',tag_font = Pmw
51          group.pack(fill = 'x',anchor=N, padx = 10, pady = 5)
52
53          CheckVar = Tkinter.IntVar()
54          s_modelist=[('Hindcast', 0), ('Forecast', 1)]
            for text, value in s_modelist:
                Radiobutton(group.interior(), text=text, value=value, variable=CheckVar)
            CheckVar.set(0)   #By default- hindcast mode

            # Create the "Simulation horizon" contents of the page.
60          group2 = Pmw.Group(sf.interior(), tag_text = 'Simulation horizon',tag_font =
61          group2.pack(fill = 'x', expand = 1, padx = 10, pady = 5)
```

1) Group

The Group widget provides a convenient way to place a labeled frame around a group of widgets.
(Source: Grayson, John E. "Python Tkinter Programming." 2011 p.57).

2) Tkinter variables

There are several ways for Pmw widgets to provide access to the widget's content. Tk provides the ability to link the current value of many widgets (such as text) to an application variable. Tkinter does not support this mode, instead it provides a *Variable* class which may be subclassed to give access to the *variable, textvariable, value* and other options within the widget. Currently, Tkinter supports *StringVar, IntVar, DoubleVar* and *BoolenVar*. These objects define *get* and *set* methods to access the widget.
(Source: Grayson, John E. "Python Tkinter Programming." 2011 p.152).

CheckVar.set(0) → CheckVar is set with 0 (hindcast mode) by default

3) List of tuples to set integer variables for each Radiobutton

Two sequence classes – lists and tuples types in Python. Lists and tuples have a lot in common. The major difference is that the elements of a list can be modified in place but a tuple is immutable: you have to deconstruct and then reconstruct a tuple to change individual elements.

```
Lst = [1,2,3,4]                 # Integer list
Lst=[[1,2,3],[ 'a', 'b', 'c']]  #List of lists
Lst=[(1, 'a'),(2, 'b'),(3, 'c')]  #List of tuples
```

(Source: Grayson, John E. "Python Tkinter Programming." 2011 p.5).

s_modelist=[('Hindcast', 0), ('Forecast', 1)] → s_modelist was created to assign 0 and 1 to Hindcast

and Forecast Radiobutton respectively.

4) For loop to create multiple Radiobuttons

The *Radiobutton* widget is for exclusive selection: selecting one button deselects any button already selected. (Source: Grayson, John E. "Python Tkinter Programming." 2011 p.37).

Text and *values* are from the s_modelist=[('Hindcast', 0), ('Forecast', 1)]
For each Radiobutton, IntVar is assigned.

```
59          # Create the "Simulation horizon" contents of the page
60          group2 = Pmw.Group(sf.int                                    w.logic
61          group2.pack(fill = 'x', e
62          #frame 1 within group2
63          fm_gp21=Frame(group2.interior())
64  -       startyear2 = Pmw.EntryField(fm_gp21
65              label_text = 'Start Year(4digit):',
66              validate = {'validator': 'numeric',                  ' : 2020, 'minstrict' : 0})
67            #  modifiedcommand = self.changed)
68          startyear2.pack(fill = 'x', padx = 10, pady = 5)
69           #starting month
70        #  mth_list = ("January","February","March", "April", "May", "June", "July", "August",
71          mth_list = ("1","2","3","4","5","6","7","8","9","10","11","12","-99")
72  -       startmonth2 = Pmw.ComboBox(fm_gp21, label_text='Start Month:', labelpos='wn',
73                         listbox_width=15, dropdown=1,
74                         selectioncommand = self.saveMonth21,
75                         scrolledlist_items=mth_list,
76                         entryfield_entry_state=DISABLED)
77          startmonth2.pack(fill = 'x', padx = 10, pady = 5)
78          # Display the first colour.
79          first = mth_list[12]
80          #print 'first=', first
81          startmonth2.selectitem(first)
82          fm_gp21.pack(fill='x', expand=1,padx=10,side=LEFT)
83
84          #frame 2 within group2
85          fm_gp22=Frame(group2.interior())
86          #end year
87  -       endyear2 = Pmw.EntryField(fm_gp22, labelpos = 'w',
88              label_text = 'End Year(4digit):',
89              validate = {'validator': 'numeric', 'min' : 1950, 'max' : 2020, 'minstrict' : 0})
90          endyear2.pack(fill = 'x', padx = 10, pady = 5)
91
92           #ending month
93  -       endmonth2 = Pmw.ComboBox(fm_gp22, label_text='End Month:', labelpos='wn',
```

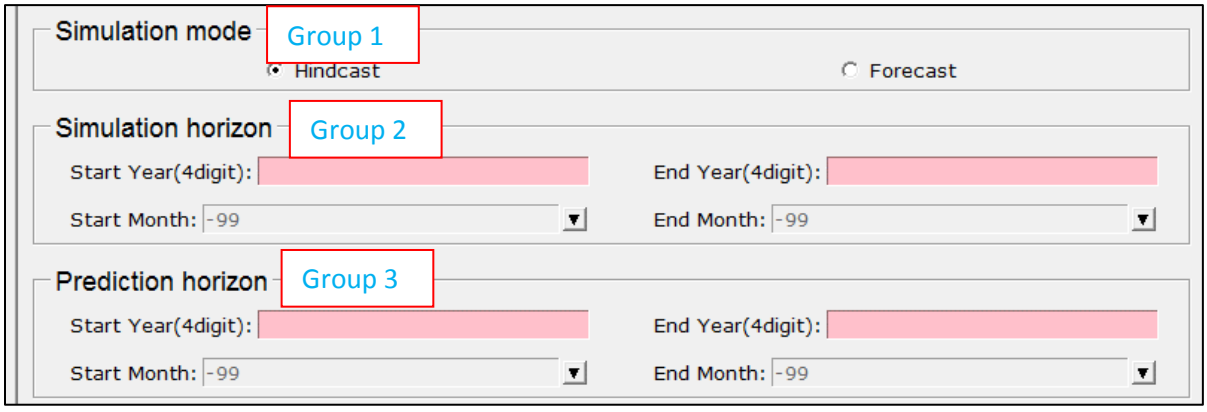
1) Create a second group for "simulation horizon"

2) Create a frame for start year and month

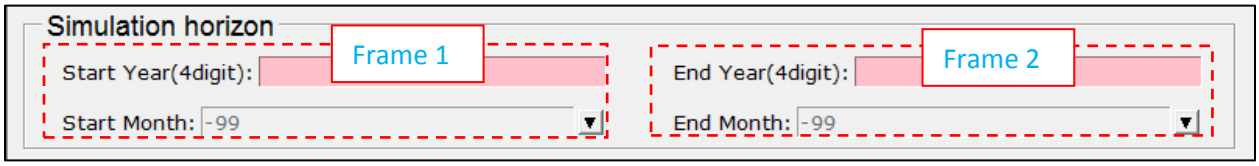
3) EntryField

4) ComboBox

1) Create a second group for "simulation horizon"



2) Create a frame for start year and month



3) EntryField

The *EntryField* widget is an *Entry* widget with associated validation methods. The built-in validation provides validators for integer, hexadecimal, alphabetic, alphanumeric, real, time and date data formats. Some of the controls that may be placed on the validation include checking conformity with the selected data format and checking that entered data is between minimum and maximum limits. (Source: Grayson, John E. "Python Tkinter Programming." 2011 p.56).

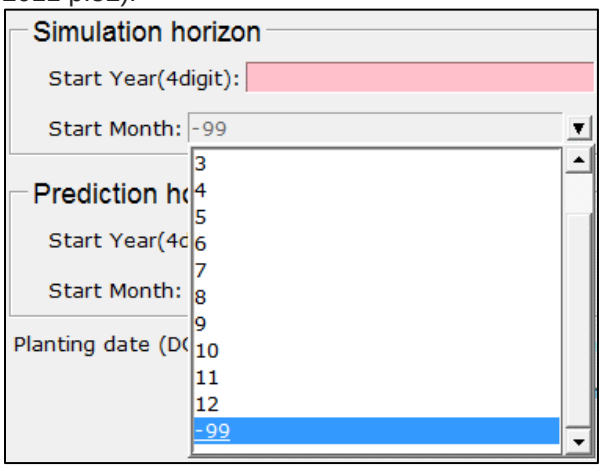
```
startyear2 = Pmw.EntryField(fm_gp21, labelpos = 'w',
    label_text = 'Start Year(4digit):',
    validate = {'validator': 'numeric', 'min' : 1950, 'max' : 2020, 'minstrict' : 0})
```

- fm_gp21→ create the EntryField within the frame, fm_gp21
- labelpos = 'w' → label position is “west’
- label_text = 'Start Year(4digit):' → label has a text, 'Start Year(4digit):'
- validate = {'validator': 'numeric', 'min' : 1950, 'max' : 2020, 'minstrict' : 0}) → entered data type should be ‘numeric’ in a range of min = 1950 and max=2020.

*Note: Due to this validator, the EntryField has pink color initially. If a user give correct input (i.e., numeric value greater than 1950 and less than 2020, the pink color will be gone. However, if the user give incorrect input, it will not allow the incorrect input to be written.

4) ComboBox

The *ComboBox* widget allows the user to select from a list of options. The list may be displayed permanently or as a dropdown list. Using the dropdown form results in GUIs which require much less space to implement complex interfaces. (Source: Grayson, John E. "Python Tkinter Programming." 2011 p.52).




```
mth_list = ("1","2","3","4","5","6","7","8","9","10","11","12","-99")
startmonth2 = Pmw.ComboBox(fm_gp21, label_text='Start Month:', labelpos='wn',
            listbox_width=15, dropdown=1,
```

selectioncommand = self.saveMonth21,
scrolledlist_items=mth_list,
entryfield_entry_state=DISABLED)

- mth_list → list to contain values for dropdown list
- fm_gp21→ create the ComboBox within the frame, fm_gp21
- dropdown=1 → dropdown is activated (true)
- selectioncommand = self.saveMonth21 → When a user selects a number on the dropdown list, it calls a function, "saveMonth21" which takes the selected variable (by default 'string' data type) and save it as integer variable, *month21*

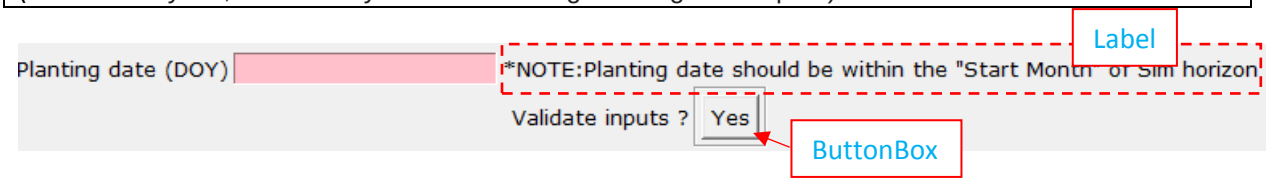
```
2126    -     def saveMonth21(self, string):
2127              global month21
2128             # print 'Text-startmonth2 is: ', integer
2129              month21=int(string)
```

- scrolledlist_items=mth_list → items for the ComboBox are from the list mth_list
- entryfield_entry_state=DISABLED → to disable the combobox' entry (i.e. user is not allowed to enter any input on the entryfield of the ComboBox

```
144     #Planting date
145     #assign frame 1
146             fm1=Frame(sf.interior())
147     -       planting_date=Pmw.EntryField(fm1, labelpos='w', label_text='Planting date (DOY)',
148                 validate = {'validator': 'numeric', 'min' : 1, 'max' : 366, 'minstrict' : 0})
149             planting_date.pack(side=LEFT) #,
150             label11= Label(fm1, text='*NOTE:                e should be within the "Start Month" o
151             label11.pack(anchor=N)
152             fm1.pack(fill='x', expand=1,padx=10,side=TOP)
153
154             # Create and pack the ButtonBox.
155     -       buttonBox = Pmw.ButtonBox(sf.interior(),
156                     labelpos = 'w',
157                     label_text = 'Validate inputs ?',
158                     frame_borderwidth = 2,
159                     frame_relief = 'groove')
160             buttonBox.pack()
161             #self.buttonBox.pack(fill = 'both', expand = 1, padx = 10, pady = 10)
162
163             # Add some buttons to the ButtonBox.
164             buttonBox.add('Yes', command = self.validate_in)
```

1) Label

Label widgets are used to display text or images.
(Source: Grayson, John E. "Python Tkinter Programming." 2011 p.35).



2) buttonBox

The *ButtonBox* widget is to implement a number of buttons and it is usually used to provide a *command area* within an application. (Source: Grayson, John E. "Python Tkinter Programming." 2011 p.51).

buttonBox.add('Yes', command = self.validate_in)

- Add a button with a label “Yes”. If a user click this button, it will activate a command “self.validate_in” and start to run the function “validate_in”.

```
2144    -     def validate_in(self):
2145              print 'start year is ', startyear2.getvalue()
2146              print 'end year is ', endyear2.getvalue()
2147              print 'start month2 is ', month21
2148              print 'end month2 is ', month22
2149              print 'start month3 is ', month31
2150              print 'end month3 is ', month32
2151              year21=int(startyear2.getvalue())
2152              year22=int(endyear2.getvalue())
2153              year31=int(startyear3.getvalue())
2154              year32=int(endyear3.getvalue())
2155              print 'start year2 is ', year21
2156              print 'end year2 is ', year22
2157              print 'start year3 is ', year31
2158              print 'end year3 is ', year32
2159              print 'simulation mode is ', CheckVar.get()
2160              print 'Plabtubg date is ', planting_date.getvalue()
```

=> The function, “validate_in” check all user’s inputs including starting year, month, ending year and month etc. and then generates an error message if the input does not make sense (e.g., starting year is greater than ending year).

2-2) Second Page

```
167   #=========================Second page for "Seasonal
168           # Add two more empty pages.
169           page2 = notebook.add('Temporal Downscaling')
170           notebook.tab('Temporal Downscaling').focus_s
171
172          # 1) ADD SCROLLED FRAME
173           sf_p2 = Pmw.ScrolledFrame(page2) #,usehullsize=1, h
174           sf_p2.pack(padx=5, pady=3, fill='both', expand=YES)
175
176           #assign frame 1
177           fm1=Frame(sf_p2.interior())
178
179           # Radio button to select "Downscaling method"
180           group21 = Pmw.Group(fm1, tag_text = 'Weather Realization (Temporal Downscaling) Method',tag_font =
181           group21.pack(fill = 'both', side=TOP, padx = 10, pady = 5)
182
183           CheckVar21 = Tkinter.IntVar()
184           down_method=[('FResampler', 0), ('Stocha
185   -       for text, value in down_method:
186               Radiobutton(group21.interior(), text=text, value=value, variable=CheckVar21).pack(side=LEFT,ex
187           CheckVar21.set(1)  #By default- Stochastic Disag
188
189           # Create button to launch the dial
190   -       down_button=Tkinter.Button(fm1,
191                   text = 'Click to add more
192                   command = self.getmoreinput,bg='gray70',font = ("weight bold",10)).pack(side=TOP,anchor=N)
193
194           # Create the "Downscaling method" contents of the page.
195           group22 = Pmw.Group(fm1, tag_text = 'FResampler',tag_font = Pmw.logicalfont('Helvetica', 0.7))
196           group22.pack(fill = 'x', side=LEFT,anchor=N,expand=YES, padx = 10, pady = 5)
```

Add the second page to set up temporal downscaling method

Add a ScrolledFrame

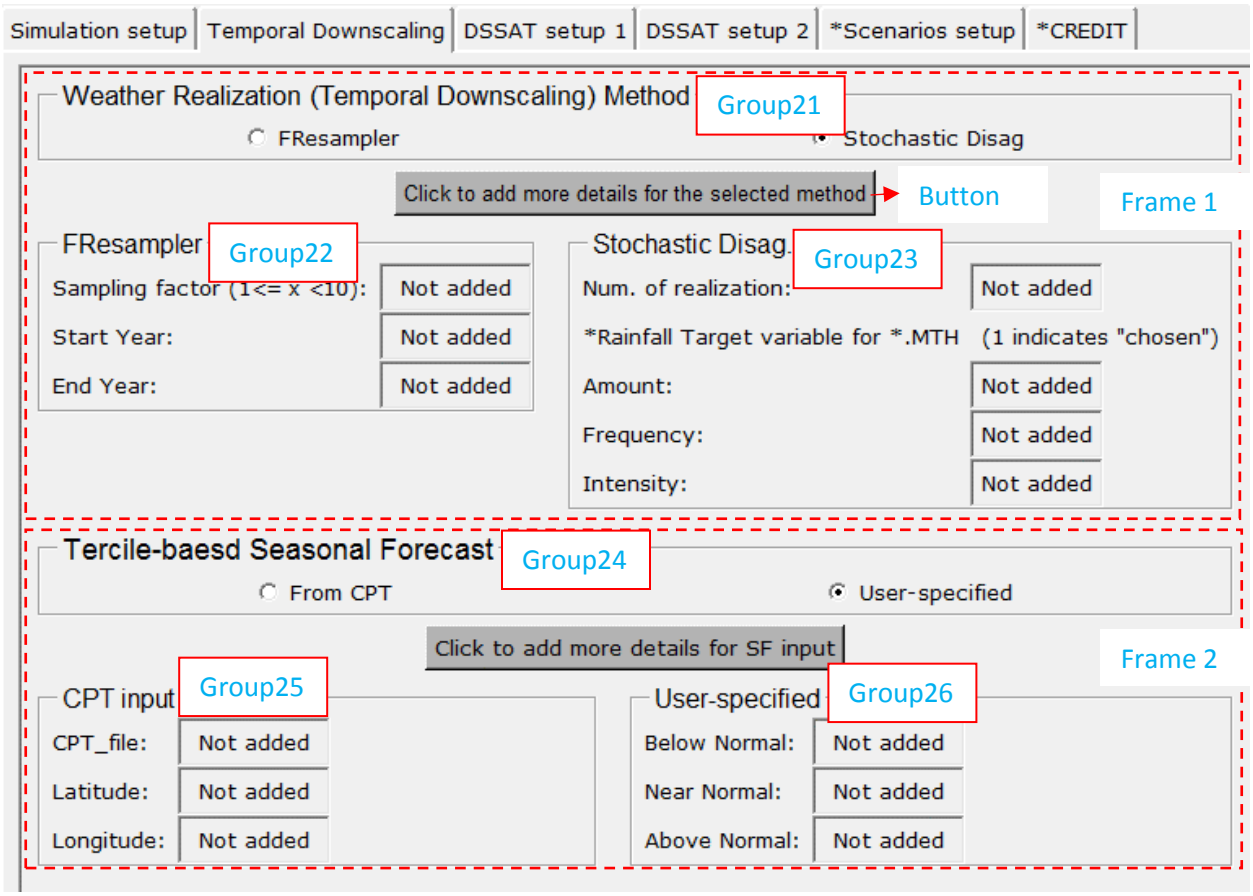
1) Frame

Add a Radiobutton to select downscaling method (either FResampler or Stochastic Disag.)

2) Button

1) Frame

Frame widgets are containers for other widgets.
(Source: Grayson, John E. "Python Tkinter Programming." 2011 p.33).



2) Button

Strictly, *buttons* are *labels* that react to mouse and keyboard events. You bind a method call or callback that is invoked when the button is activated. (Source: Grayson, John E. "Python Tkinter Programming." 2011 p.36).

```
    # Create button to launch the dialog
    down_button=Tkinter.Button(fm1,
        text = 'Click to add more details for the selected method',
        command = self.getmoreinput,bg='gray70',font = ("weight
bold",10)).pack(side=TOP,anchor=N)
```

→Add a button to activate a new window which is to get more detailed information about the selected downscaling method. If a user click this button, it will activate a command “self.getmoreinput” and start to run the function “getmoreinput”.

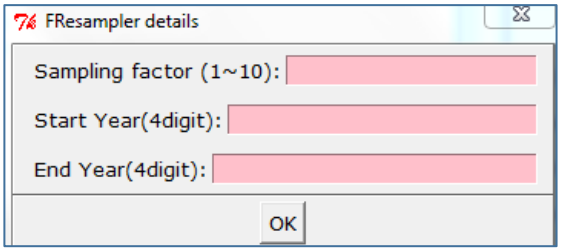
```
2064    -     def getmoreinput(self):
2065              print 'simulation mode is '
                  if CheckVar21.get() == 0:
                      self.FRdialog.activate(
                      print 'FResampler Sampl
                      self.label12.configure(
                      self.label14.configure(text=self.FRyear1.getvalue(),background='hone
2071                  self.label16.configure(text=self.FRyear2.getvalue(),background='ho
2072    -         else:
                      self.DisaggDialog.activate()
                      print 'Stochastic DisAg. target (amount) is ', CheckVar31.get()
                      self.label22.configure(text=self.nRealz.getvalue(),background='honey
                      self.label26.configure(text=CheckVar31.get(),background='honeydew1')
                      self.label28.configure(text=CheckVar32.get(),background='honeydew1')
                      self.label20.configure(text=CheckVar33.get(),background='honeydew1')
```

If FResampler is chosen, activate a new dialog (a)

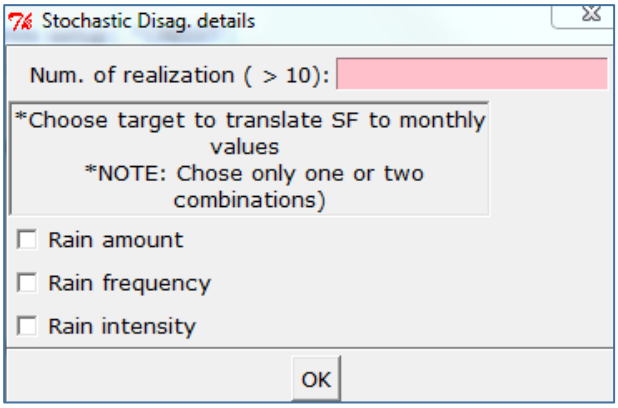
If Stochastic Disag. is chosen, activate a new dialog (b)

Get user's inputs from the new dialog (a) and save them on labels in the group 22 above. If the new inputs are added, the color of the labels turns to 'honeydew' color from originally grey (see (c)).

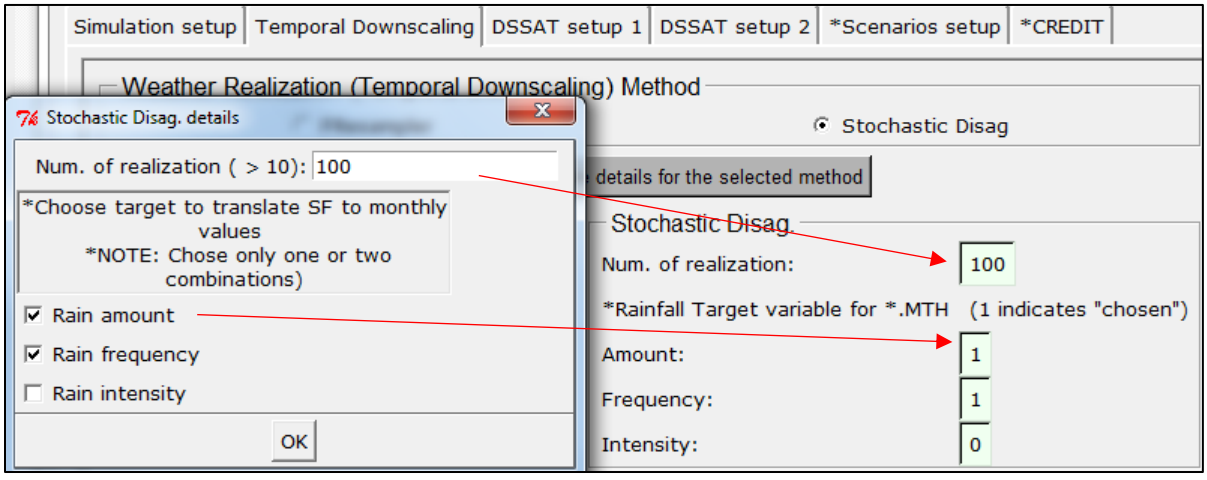
(a)



(b)



(c)



```
        #to add more specification for FResampler
        #contents for FResampler
        self.FRdialog = Pmw.Dialog(parent, title='FResampler details')
-       self.SampFactor = Pmw.EntryField(self.FRdialog.interior(), labelpos =
            label_text = 'Sampling factor (1~10):',
            validate = {'validator': 'real', 'min' : 1.0, 'max' : 9.99, 'mins
            #validate = {'validator': 'real', 'min' : 0.1, 'max' : 1, 'minstr
        self.SampFactor.pack(fill = 'x', padx = 10, pady = 5)
        #-start year of historic weather records
-       self.FRyear1 = Pmw.EntryField(self.FRdialog.interior(), labelpos = 'w',
            label_text = 'Start Year(4digit):',
            validate = {'validator': 'numeric', 'min' : 1910, 'max' : 2020, 'minstrict' : 0})
        self.FRyear1.pack(fill = 'x', padx = 10, pady = 5)
        #-last year of historic weather records
-       self.FRyear2 = Pmw.EntryField(self.FRdialog.interior(), labelpos = 'w',
            label_text = 'End Year(4digit):',
            validate = {'validator': 'numeric', 'min' : 1910, 'max' : 2020, 'minstrict' : 0})
        self.FRyear2.pack(fill = 'x', padx = 10, pady = 5)
        self.FRdialog.withdraw()

        #to add more specification for Stochastic Disag.
        #contents for Stochastic Disag.
        self.DisaggDialog = Pmw.Dialog(parent, title='Stochastic Disag. detai
        #-number of realization
-       self.nRealz = Pmw.EntryField(self.DisaggDialog.interior(), labelpos =
            label_text = 'Num. of realization ( > 10):',
            validate = {'validator': 'numeric', 'min' : 10, 'max' : 500, 'min
        self.nRealz.pack(fill = 'x',side=TOP,anchor=W,padx = 10, pady = 5)
        #-checkbutton for target values to create *.MTH for DisAg.exe
        label=Tkinter.Label(self.DisaggDialog.interior(), text="*Choose targe
        label.pack(side=TOP,anchor=W)
        CheckVar31 = Tkinter.IntVar()
        CheckVar32 = Tkinter.IntVar()
        CheckVar33 = Tkinter.IntVar()
-       b1 = Tkinter.Checkbutton(self.DisaggDialog.interior(), text = 'Rain amount',
                                 variable = CheckVar31,onvalue = 1, offvalue = 0).pack(side=TOP,anchor=W)
-       b2 = Tkinter.Checkbutton(self.DisaggDialog.interior(), text = 'Rain frequency',
                                 variable = CheckVar32,onvalue = 1, offvalue = 0).pack(side=TOP,anchor=W)
-       b3 = Tkinter.Checkbutton(self.DisaggDialog.interior(), text = 'Rain intensity',\
                                 variable = CheckVar33,onvalue = 1, offvalue = 0).pack(side=TOP,anchor=W)
        self.DisaggDialog.withdraw()
```

1) Dialog

Add "EntryField"s on the new dialog (a) to get more details on the FResampler

Add "EntryField" and Checkbutton on the new dialog (b) to get more details on the Stochastic Disag.

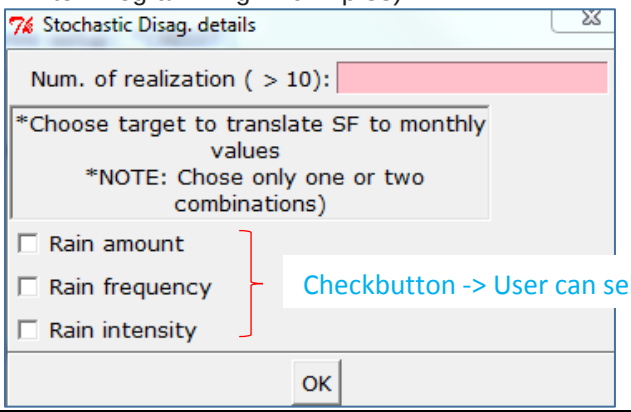
2) Checkbutton

1) Dialog

The *Dialog* widget provides a simple way to create a *toplevel* containing a *ButtonBox* and a child site area.

2) Checkbutton

Checkbutton widgets are used to provide on/off selections for one or more items. Unlike radiobuttons there is no interaction between checkbuttons. (Source: Grayson, John E. "Python Tkinter Programming." 2011 p.38).



```
239         #assign frame 2
240         fm2=Frame(sf_p2.interior())
241         # Radio button to select "How to get seasonal forecast"
242         group24 = Pmw.Group(fm2, tag_text = 'Tercile-baesd Seasonal Forecast',tag_font = Pmw.logicalfont('H
243         group24.pack(fill = BOTH, side=TOP, padx = 10, pady = 5)
244
245         CheckVar22 = Tkinter.IntVar()
246         SF_data=[('From CPT', 0), ('User-specified',
247 -       for text, value in SF_data:
248             Radiobutton(group24.interior(), text=text
249         CheckVar22.set(1)  #By default- FResampler
250
251         # Create button to launch the dialog for seasona
252 -       down_button22=Tkinter.Button(fm2,
253                 text = 'Click to add more details for SF
254                 command = self.getSFinput,bg='gray70').pa
255
256         # Create contents for seasonal forecast from CPT
257         group25 = Pmw.Group(fm2, tag_text = 'CPT input',t
258         group25.pack(fill = 'x', anchor=N,side=LEFT,expar
259         #Label to fill user-input for downscaling detail
260         self.label31 = Label(group25.interior(), text='CPT_file:', padx=5, pady=5)
261         self.label31.grid(row=0,column=0, sticky=W) #rowspan=1,columnspan=1)
262         self.label32 = Label(group25.interior(), text='Not added',relief='sunken',width=10,padx=5, pady=5)
263         self.label32.grid(row=0,column=1, sticky=W) #rowspan=1,columnspan=1)
264         self.label33 = Label(group25.interior(), text='Latitude:', padx=5, pady=5)
265         self.label33.grid(row=1,column=0, sticky=W) #rowspan=1,columnspan=1)
266         self.label34 = Label(group25.interior(), text='Not added',relief='sunken',width=10, padx=5, pady=5)
267         self.label34.grid(row=1,column=1, sticky=W) #rowspan=1,columnspan=1)
268         self.label35 = Label(group25.interior(), text='Longitude:', padx=5, pady=5)
269         self.label35.grid(row=2,column=0, sticky=W) #rowspan=1,columnspan=1)
270         self.label36 = Label(group25.interior(), text='Not added',relief='sunken',width=10, padx=5, pady=5)
271         self.label36.grid(row=2,column=1, sticky=W) #rowspan=1,columnspan=1)
272
273         #Create contents for user-specified seasonal forecast input (BN-NN-AN)
274         group26 = Pmw.Group(fm2, tag_text = 'User-specified',tag_font = Pmw.logicalfont('Helvetica', 0.7))
275         group26.pack(fill = 'x', anchor=N,side=LEFT,expand=1, padx = 10, pady = 5)
276         self.label41 = Label(group26.interior(), text='Below Normal:', padx=5, pady=5)
277         self.label41.grid(row=0,column=0, sticky=W) #rowspan=1,columnspan=1)
278         self.label42 = Label(group26.interior(), text='Not added',relief='sunken',width=10, padx=5, pady=5)
279         self.label42.grid(row=0,column=1, sticky=W) #rowspan=1,columnspan=1)
280         self.label43 = Label(group26.interior(), text='Near Normal:', padx=5, pady=5)
```

Add Radiobuttons to select options for how to get seasonal forecast (either CPT output file or user-specified)

Add Button to get more details about the selected seasonal forecast input.

1) Clicking this button will generate new pop-up windows.

1) Button for seasonal forecast input sources

```
2078 -      def getSFinput(self):
2079            print 'Seasonal forecast is from ', CheckVar22.get()
                if CheckVar22.get() == 0:
                    self.CPTDialog.activate()
                   # print 'CPT_file is ', sel                tvalue()
                   #  self.label32.configure(text=self.CPT_file.getvalue())
                     self.label32.configure(text=CPT_path)
2085                 self.label34.configure(text=self.latitude.getvalue())
2086                 self.label36.configure(text=self.longitude.getvalue())
                else:
                    self.UserDialog.activate()
                    print 'Below normal is ', self.SampFactor.getvalue()
                    self.label42.configure(text=self.BN.getvalue(),background='honeydew1')
                    Near_normal=100-float(self.BN.getvalue())-float(self.AN.getvalue())
                    self.label44.configure(text=str(Near_normal),background='honeydew1')
                    self.label46.configure(text=self.AN.getvalue(),background='honeydew1')
```

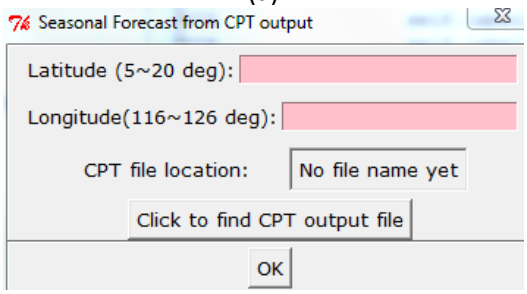
If 'from CPT" is chosen, activate a new dialog (a)

1-a) CPTDialog

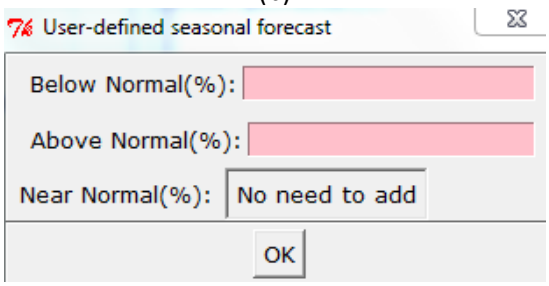
If 'User-specified" is chosen, activate a new dialog (b)

1-b) UserDialog

(a)



(b)



1-a) CPTDialog

```
959         #to add more specification for CPT-based seasonal forecast.
960         self.CPTDialog = Pmw.Dialog(parent, title='Seasonal Forecast from CPT output')
961         fmcpt=Frame(self.CPTDialog.interior())
962         #-latitude of the target location
963 -       self.latitude = Pmw.EntryField(fmcpt, labelpos = 'w',
964             label_text = 'Latitude (5~20 deg):',
965             validate = {'validator': 'real', 'min' : 5, 'max' : 20, 'minstrict' : 0})
966         self.latitude.pack(fill = 'both', expand = 1,anchor=W,padx = 10, pady = 5)
967         #-longitude of the target location
968 -       self.longitude = Pmw.EntryField(fmcpt, labelpos = 'w',
969             label_text = 'Longitude(116~126 deg):',
970             validate = {'validator': 'real', 'min' : 116, 'max' : 126, 'minstrict' : 0}
971         self.longitude.pack(fill = 'x', anchor=W,padx = 10, pady = 5)
972         fmcpt.pack(side=TOP)
973         fmcpt2=Frame(self.CPTDialog.interior())
974         #CPT output file
975         self.CPT_label0=Label(fmcpt2, text='CPT file location:', padx=5, pady=5)
976         self.CPT_label0.pack(fill = 'x', side=LEFT, anchor=W, padx = 10, pady = 5)
977         self.CPT_label = Label(fmcpt2, text='No file name yet',relief='sunken', padx=5,
978         self.CPT_label.pack(fill = 'x', side=LEFT, anchor=W, padx = 10, pady = 5)
979         fmcpt2.pack(side=TOP)
980         fmcpt3=Frame(self.CPTDialog.interior())
981        # Create button to get file path
982 -       self.file_button=Tkinter.Button(fmcpt3
983                 text = 'Click to find CPT outp
984                 command = self.getCPTfile).pac
985         fmcpt3.pack(side=TOP)
986         #self.CPT_file = Pmw.EntryField(self.C
987         #    label_text = 'CPT file location:'
988         #    validate = {'validator': 'alphabe
989         #self.CPT_file.pack(fill = 'x', padx = 10, pady = 5)
990         self.CPTDialog.withdraw()
```

If user click the button “Click to find CPT output file”, it will activate “getCPTfile” function to open a pop-up window to search for a target file.

```
2021     #find CPT output file path
2022 -   def getCPTfile(self):
2023         global CPT_path
2024         CPT_path = askopenfilename(initialdir="C:\\",
2025         print 'CPT filename is', CPT_path
2026         self.CPT_label.configure(text=CPT_path)
2027         self.label32.configure(background='honeydew1')
2028         self.label34.configure(background='honeydew1')
2029         self.label36.configure(background='honeydew1')
```

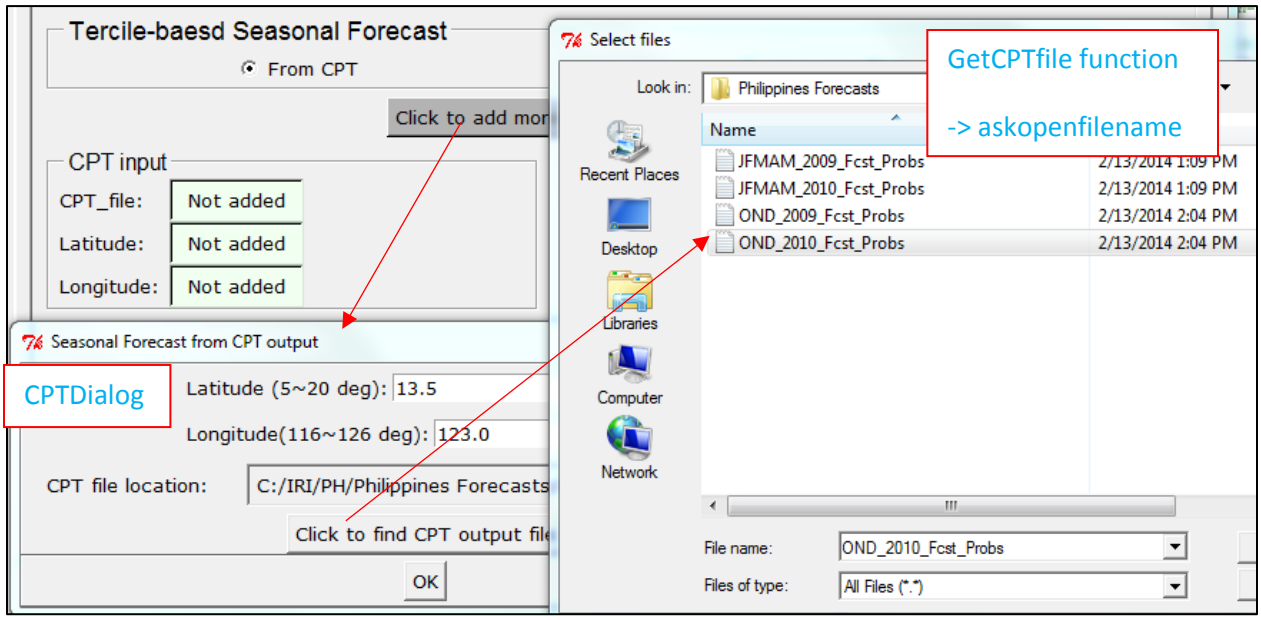
* Use “askopenfilename” to get a file name to open. Initial searching directory is set as “C:\\”

In order to use this, tkFileDialog module should be imported.

*askopenfilename

tkFileDialog
If you want to open or save a file or to choose a directory using a filedialog you dont need to

implement it on your own. The module ***tkFileDialog*** is just for you.
(source: http://tkinter.unpythonic.net/wiki/tkFileDialog)



*Example of seasonal Climate Forecast from CPT output → Based on the given latitude and longitude, the seasonal forecast (BN-NN-AN) will be determined from the CPT output file as shown below.

OND_2009_Fcst_Probs.txt

```
xmlns:cpt=http://iri.columbia.edu/CPT/v10/
cpt:ncats=3
cpt:field=prcp, cpt:C=1, cpt:clim_prob=0.333333333333, cpt:T=2009-10/12, cpt:nrow=30, cpt:ncol=22, cpt:row=Y, c
```

Longitude

Latitude	117.250000000	117.750000000	118.250000000	118.750000000	119.250000000	119.750000000	120.250
19.7500000000	-999.000000000	-999.000000000	-999.000000000	-999.000000000	-999.000000000	-999.000000000	
19.2500000000	-999.000000000	-999.000000000	-999.000000000	-999.000000000	-999.000000000	-999.000000000	
18.7500000000	-999.000000000	-999.000000000	-999.000000000	-999.000000000	-999.000000000	-999.000000000	
18.2500000000	-999.000000000	-999.000000000	-999.000000000	-999.000000000	-999.000000000	-999.000000000	
17.7500000000	-999.000000000	-999.000000000	-999.000000000	-999.000000000	-999.000000000	-999.000000000	
17.2500000000	-999.000000000	-999.000000000	-999.000000000	-999.000000000	-999.000000000	-999.000000000	
16.7500000000	-999.000000000	-999.000000000	-999.000000000	-999.000000000	-999.000000000	-999.000000000	
16.2500000000	-999.000000000	-999.000000000	-999.000000000	-999.000000000	-999.000000000	36.4795321705	
15.7500000000	-999.000000000	-999.000000000	-999.000000000	-999.000000000	-999.000000000	35.1974406612	
15.2500000000	-999.000000000	-999.000000000	-999.000000000	-999.000000000	-999.000000000	38.6092297598	
14.7500000000	-999.000000000	-999.000000000	-999.000000000	-999.000000000	-999.000000000	-999.000000000	
14.2500000000	-999.000000000	-999.000000000	-999.000000000	-999.000000000	-999.000000000	-999.000000000	
13.7500000000	-999.000000000	-999.000000000	-999.000000000	-999.000000000	-999.000000000	-999.000000000	
13.2500000000	-999.000000000	-999.000000000	-999.000000000	-999.000000000	-999.000000000	-999.000000000	
12.7500000000	-999.000000000	-999.000000000	-999.000000000	-999.000000000	-999.000000000	-999.000000000	
12.2500000000	-999.000000000	-999.000000000	-999.000000000	-999.000000000	-999.000000000	50.0836299273	
11.7500000000	-999.000000000	-999.000000000	-999.000000000	-999.000000000	-999.000000000	49.1025050307	
11.2500000000	-999.000000000	-999.000000000	-999.000000000	-999.000000000	53.5592449163	54.5095328949	

BN (%)

1-b) UserDialog

```
992             #to add more specification for user-defined seasonal forecast.
993             self.UserDialog = Pmw.Dialog(parent, title='User-defined seasonal forecast')
994     -       self.BN = Pmw.EntryField(self.UserDialog.interior(), labelpos = 'w',
995                 label_text = 'Below Normal(%):',
996                 validate = {'validator': 'real', 'min' : 1, 'max' : 99, 'minstrict' : 0})
997             self.BN.pack(fill = 'x', padx = 10, pady = 5)
998             #-Above normal
999     -       self.AN = Pmw.EntryField(self.UserDialog.interior(), labelpos = 'w',
1000                label_text = 'Above Normal(%):',
1001                validate = {'validator': 'real', 'min' : 1, 'max' : 99, 'minstrict' : 0})
1002            self.AN.pack(fill = 'x', padx = 10, pady = 5)
1003            #-Near Normal
1004            self.NN=Label(self.UserDialog.interior(), text='Near Normal(%):', padx=5, pady
1005            self.NN_box = Label(self.UserDialog.interior(), text='No need to add',relief='
1006            self.UserDialog.withdraw()
```

2-3) Third Page to set up DSSAT inputs

```
290     #=========================Third page for "DSSAT ba
291             page3 = notebook.add('DSSAT setup 1')
292             notebook.tab('DSSAT setup 1').focus_set()
293            # 1) ADD SCROLLED FRAME
294             sf_p3 = Pmw.ScrolledFrame(page3) #,usehullsize=1, hull_width=700, hull_height=220)
295             sf_p3.pack(padx=5, pady=3, fill='both', expand=YES)
296
297             #assign frame 1
298             fm31=Frame(sf_p3.interior())
299             fm_in0=Frame(fm31,width=290)
300             # Radio button to select "Irrigation method"
301             group31 = Pmw.Group(fm_in0, tag_text = 'Irrigation option',tag_font = Pmw.logicalfont('Helvetic
302             group31.pack(fill='both', expand=1, side=TOP, padx = 10, pady = 2)
303
304             Rbutton1 = Tkinter.IntVar()
305             irr_option=[('Rainfed', 0), ('Irrigated', 1
306     -       for text, value in irr_option:
307                 Radiobutton(group31.interior(), text=text, value=value, variable=Rbutton1).pack(side=TOP,an
308             Rbutton1.set(0)  #By default- Rainfed
309
310             # Radio button to select "Planting method"
311             group32 = Pmw.Group(fm_in0, tag_text = 'Planting method',tag_font = Pmw.logicalfont('Helvetica',
312             group32.pack(fill='both', expand=1,side=TOP, padx = 10, pady = 2)
313
314             Rbutton2 = Tkinter.IntVar()
315             plt_option=[('Dry seed', 0), ('Transplantin
316     -       for text, value in plt_option:
317                 Radiobutton(group32.interior(), text=text, value=value, variable=Rbutton2).pack(side=TOP,an
318             Rbutton2.set(1)  #By default- Transplanting
319             fm_in0.pack(fill = 'both', expand = 1,side=LEFT)
320             fm_in1=Frame(fm31)
321             # set up "Planting details"
322             group33 = Pmw.Group(fm_in1, tag_text = 'Planting details',tag_font = Pmw.logicalfont('Helvetica
323             group33.pack(fill = 'x', side=TOP, padx = 10, pady = 2)
```

Add a new page to set up DSSAT inputs

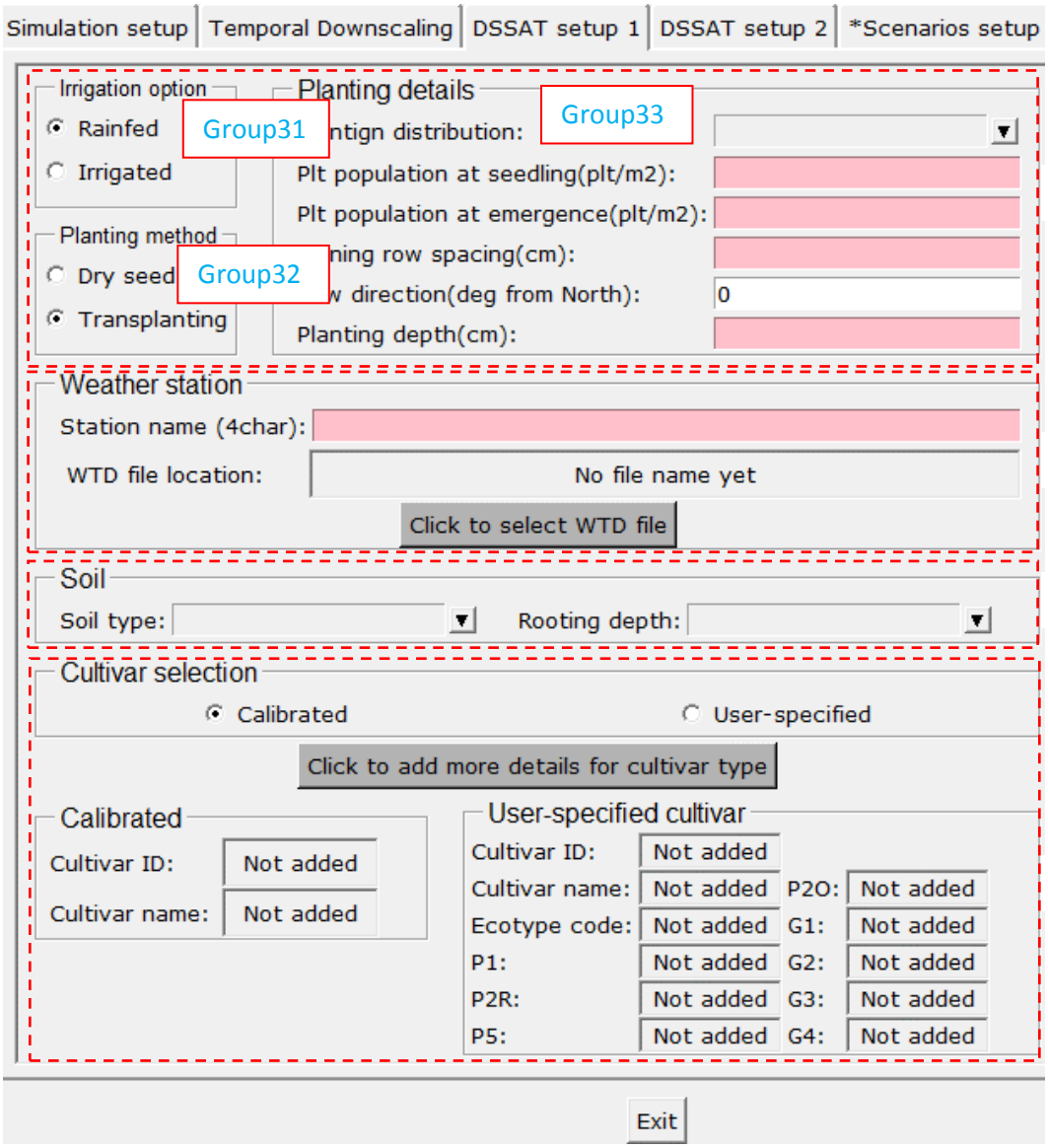
Add a ScrolledFrame

Frame

Add Radiobuttons to select irrigation options

Add Radiobuttons to select planting method

****Widgets used to create the rest of pages are similar to the previous ones. Therefore, all details for the pages 3 and 4 are skipped.**

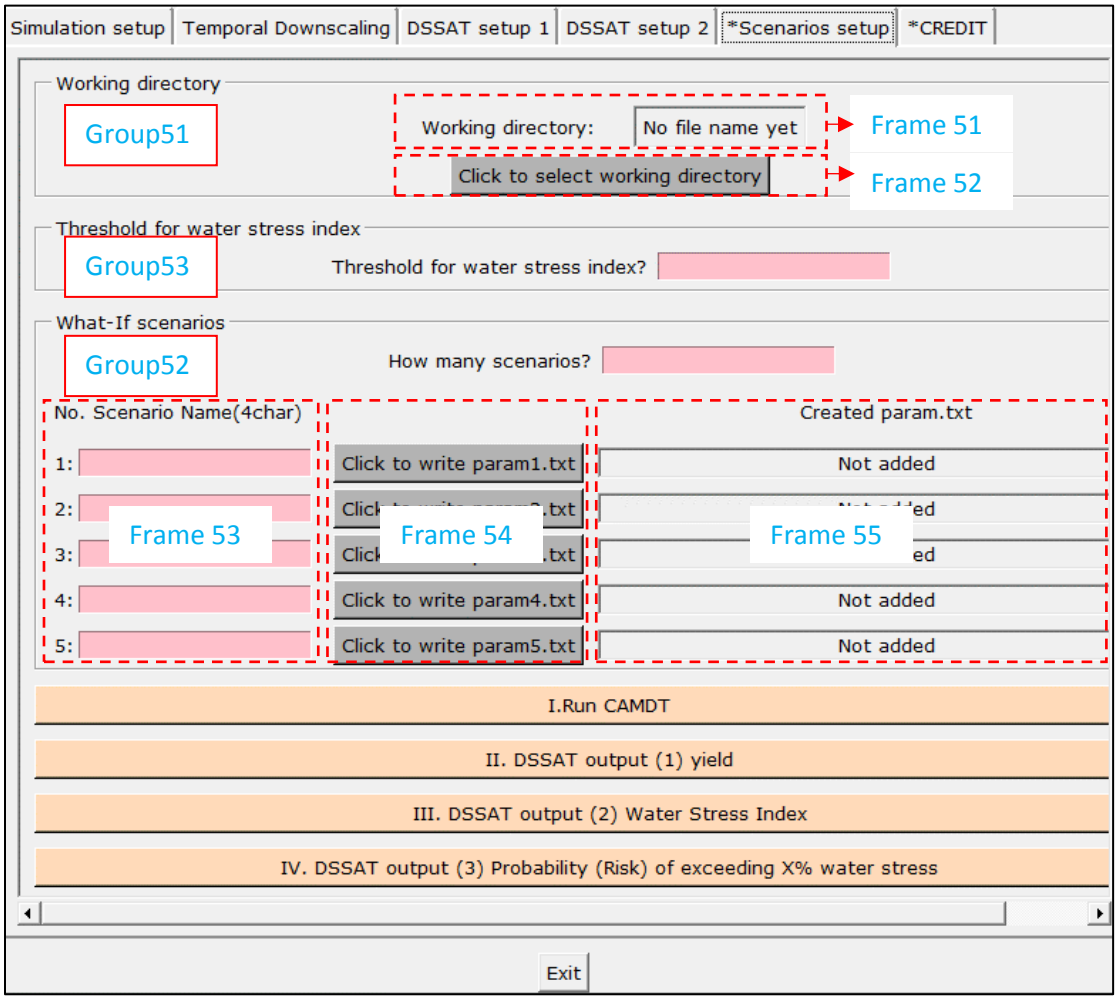

Simulation setup
Temporal Downscaling
DSSAT setup 1
DSSAT setup 2
*Scenarios setup
Irrigation option
Rainfed
Irrigated
Group31
Planting method
Dry seed
Transplanting
Group32
Planting details
Group33
ntign distribution:
Plt population at seedling(plt/m2):
Plt population at emergence(plt/m2):
ning row spacing(cm):
v direction(deg from North):
0
Planting depth(cm):
Frame 1
Weather station
Station name (4char):
WTD file location:
No file name yet
Click to select WTD file
Frame 2
Soil
Soil type:
Rooting depth:
Frame 3
Cultivar selection
Calibrated
User-specified
Click to add more details for cultivar type
Calibrated
Cultivar ID:
Not added
Cultivar name:
Not added
User-specified cultivar
Cultivar ID:
Not added
Cultivar name:
Not added
P2O:
Not added
Ecotype code:
Not added
G1:
Not added
P1:
Not added
G2:
Not added
P2R:
Not added
G3:
Not added
P5:
Not added
G4:
Not added
Frame 4
Exit

2-5) Fifth Page to set up What-If scenarios

```
679     #=========================5th  page for "DSSAT
680             page5 = notebook.add('*Scenarios setup
681             notebook.tab('*Scenarios setup').focus_set()
682
683            # 1) ADD SCROLLED FRAME
684             sf_p5 = Pmw.ScrolledFrame(page5) #,usehullsiz                l_height=220)
685             sf_p5.pack(padx=5, pady=3, fill='both', expan
686
687             group51 = Pmw.Group(sf_p5.interior(), tag_text = 'Working directory')
688             group51.pack(fill = 'x', expand = 1, anchor=N, padx = 10, pady = 5)
689             #Working directory
690             fm51=Frame(group51.interior())
691             self.WDir_label0=Label(fm51, text='Working directory:', padx=5, pady=5)
692             self.WDir_label0.pack(fill = 'x', side=LEFT, anchor=W, padx = 10, pady = 5)
693             self.WDir_label = Label(fm51, text='No file name yet',relief='sunken', padx=5, pady=5)
694             self.WDir_label.pack(fill = 'x', side=LEFT, anchor=W, padx = 10, pady = 5)
695             fm51.pack(side=TOP)
696             fm52=Frame(group51.interior())
697            # Create button to get file path
698 -           self.file_button=Tkinter.Button(fm52,
699                     text = 'Click to select working directory',
700                     command = self.getWdir,bg='gray70').pack(side=TOP,anchor=N)
701             fm52.pack(side=TOP)
```

Add a new page to set up what-if scenarios

Add a ScrolledFrame



```
778         #2. Run CAMDT
779 -       button62=Tkinter.Button(sf_p5.interior(), text = 'I.Run CA
780                 command = self.RunCAMDT,bg='peachpuff1').pack(fill
781
782         #3. Retrieve DSSAT simulation output/graphs (1) yield
783 -       button63=Tkinter.Button(sf_p5.interior(),text = 'II. DSSAT
784                 command = self.Yield_Analysis,bg='peachpuff1').pack
785
786         # Create button to execute graphs (2) water stress index
787 -       button64=Tkinter.Button(sf_p5.interior(), text = 'III. DSSAT ou
                  mmand = self.WSI_Analysis,bg='peachpuff1').pack(fill
790         # Create button to execute graphs (3)prob of exceeding X% water
791 -       button65=Tkinter.Button(sf_p5.interior(), text = 'IV. DSSAT out
792                 command = self.Risk_Analysis,bg='peachpuff1').pack(fill
```

If user click the button "I. Run CAMDT", it will run "RunCAMDT" function (see (a) below).

Run "Yield_Analysis" function (see (b) below).

Run "WSI_Analysis"

Run "Risk_Analysis"

(a) RunCAMDT

```
1657 -  def RunCAMDT(self):
1658 -      entries = ("AveStress.txt", "SumStress.txt",
1659                 "ET.OUT","OVERVIEW.OUT","PlantN.OU
1660                 "SoilNiBal.OUT","SoilNoBal.OUT","S
1661                 "SolNBalSum.OUT","Summary.OUT","We
1662 -       if self.label54.cget("text") != "Not added":
1663             #change directory
1664             os.chdir(Wdir_path)
1665             #Delete *.WTD from previous runs
1666             WTD_names=self.stn_name.getvalue()+"0*.WT
1667 -           for file in os.listdir('.'):
1668 -               if fnmatch.fnmatch(file, WTD_names):
1669                     # print file
1670 -                   try:
1671                         os.remove(file)
1672 -                   except Exception,e:
1673                         print e
1674
1675             print 'new directory is', os.getcwd()
1676             print 'self.label54.cget:', self.label54.cge
1677             param_txt ="param_"+self.name1.getvalue()+"_
1678             args = "CAMDT_exe " + param_txt
1679             #===Run executable with argument
1680             subprocess.call(args) #, stdout=FNULL, st
1681             #create a new folder to save outputs of t
1682             new_folder=self.name1.getvalue()+"_output
1683 -           if os.path.exists(new_folder):
1684                 shutil.rmtree(new_folder)    #remove existing folder
1685             os.makedirs(new_folder)
1686             #copy outputs to the new folder
1687             dest_dir=Wdir_path + "\\"+new_folder
1688             print 'dest_dir is', dest_dir
1689 -           for entry in entries:
1690                 source_file=Wdir_path + "\\" + entry
1691                 shutil.move(source_file, dest_dir)
1692 -       if self.label55.cget("text") != "Not added": #
```

List of output file names which will be moved to a new directory

If the label 54 in Frame 55 is NOT same as "Not added" (which means a scenario name and param*.txt is created successfully), then change working dir to the specified one.

*Note: "print" statements are only for debugging purpose (i.e. not related to any modeling processes

1) Run executable (CAMDT_exe.exe) with an argument

Output files (specified in the "entries" list above) after running DSSAT will be moved to a newly created folder for the specific scenario.

1) Run executable using subporcess

The subprocess module allows you to spawn new processes, connect to their input/output/error pipes, and obtain their return codes.

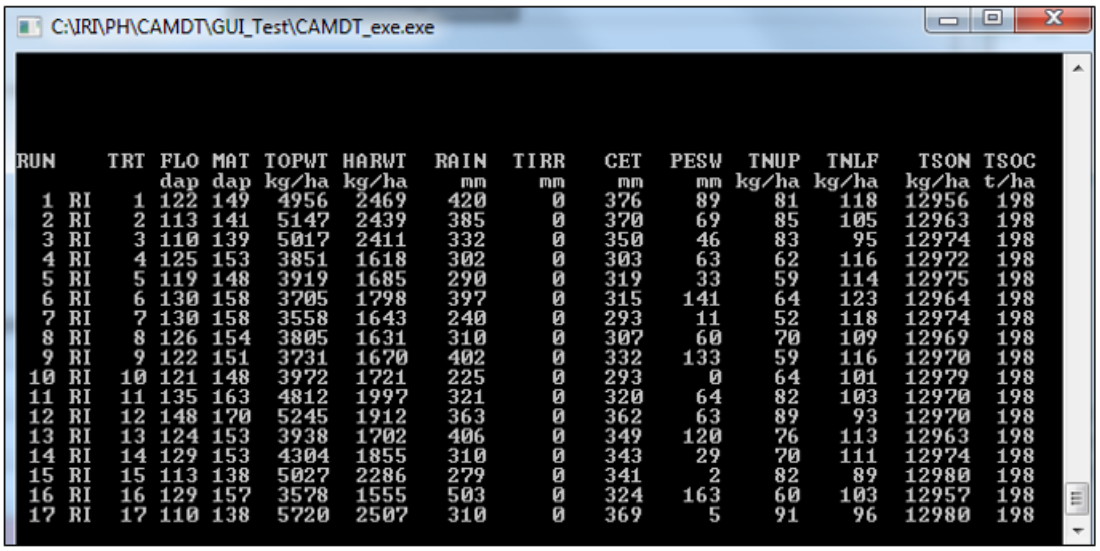
The recommended way to launch subprocesses is to use the following convenience functions.

*subprocess.call(args, *, stdin=None, stdout=None, stderr=None, shell=False)*

Run the command described by args. Wait for command to complete, then return the returncode attribute. (source: https://docs.python.org/2/library/subprocess.html)

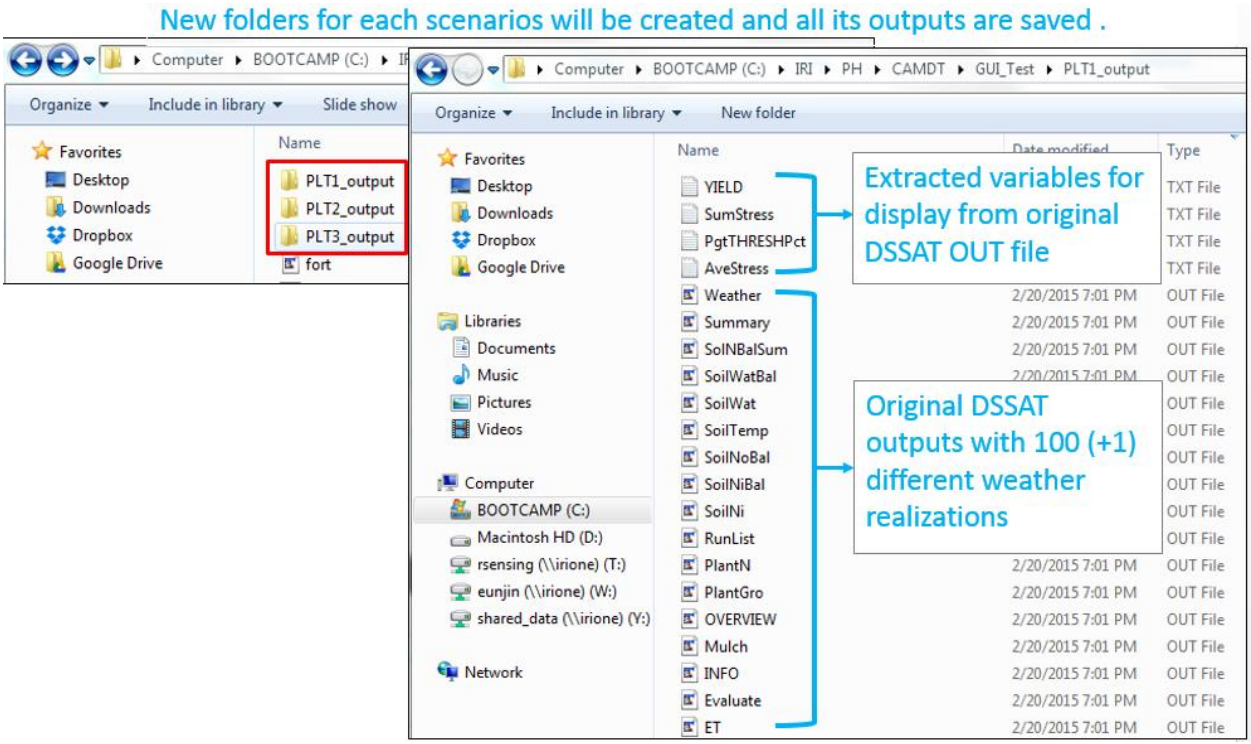
→ This subprocess will run “CAMDT_exe.exe” with an argument “param*.txt”. This is same as running the executable on MS-Dos window by typing “CAMDT_exe param*.txt”
The executable, “CAMDT_exe.exe” was created from Fortran compiler. The Fortran code developed by Eunjin will run temporal downscaling algorithm and then run DSSAT using the user-provided input written in param*.txt (simulation period, management practices, soil, irrigation etc.)

While CAMDT is running, Dos window shows progress



RUN		TRT	FLO dap	MAT dap	TOPWT kg/ha	HARWT kg/ha	RAIN mm	TIRR mm	CET mm	PESW mm	TNUP kg/ha	TNLF kg/ha	TSON kg/ha	TSOC t/ha
1	RI	1	122	149	4956	2469	420	0	376	89	81	118	12956	198
2	RI	2	113	141	5147	2439	385	0	370	69	85	105	12963	198
3	RI	3	110	139	5017	2411	332	0	350	46	83	95	12974	198
4	RI	4	125	153	3851	1618	302	0	303	63	62	116	12972	198
5	RI	5	119	148	3919	1685	290	0	319	33	59	114	12975	198
6	RI	6	130	158	3705	1798	397	0	315	141	64	123	12964	198
7	RI	7	130	158	3558	1643	240	0	293	11	52	118	12974	198
8	RI	8	126	154	3805	1631	310	0	307	60	70	109	12969	198
9	RI	9	122	151	3731	1670	402	0	332	133	59	116	12970	198
10	RI	10	121	148	3972	1721	225	0	293	0	64	101	12979	198
11	RI	11	135	163	4812	1997	321	0	320	64	82	103	12970	198
12	RI	12	148	170	5245	1912	363	0	362	63	89	93	12970	198
13	RI	13	124	153	3938	1702	406	0	349	120	76	113	12963	198
14	RI	14	129	153	4304	1855	310	0	343	29	70	111	12974	198
15	RI	15	113	138	5027	2286	279	0	341	2	82	89	12980	198
16	RI	16	129	157	3578	1555	503	0	324	163	60	103	12957	198
17	RI	17	110	138	5720	2507	310	0	369	5	91	96	12980	198

New folders for each scenarios will be created and all its outputs are saved .



(b) Run "Yield_Analysis"

```
1800    def Yield_Analysis(self):
1801        #Get output YIELD.txt from all scenario
1802        fname=[]
1803        scename=[]   #scenario name
1804        temp_entries=[[self.label54.cget("text"),self.name1.getvalue()],
1805                      [self.label55.cget("text"),self.name2.getvalue()],
1806                      [self.label56.cget("text"),self.name3.getvalue()],
1807                      [self.label57.cget("text"),self.name4.getvalue()],
1808                      [self.label58.cget("text"),self.name5.getvalue()]]
1809        count=0
1810        for entry in temp_entries:
1811            if entry[0] != "Not added":
1812                print 'yield.txt is ', Wdir_path + "\\" + entry[1] +"_
1813                fname.append(Wdir_path + "\\" + entry[1] +"_output"+"
1814                scename.append(entry[1])
1815                count=count+1
1816
1817        #Read YIELD.txt from all scenario output
1818        yield_data=[]
1819        obs=[]
1820        for x in range(0, count):
1821            temp_data = np.loadtxt(fname[x])
1822            n_col=temp_data.__len__() #n real
1823            net_data=temp_data[0:n_col-1]
            obs.append(temp_data[n_col-1])
            print 'yield file name is ', fname[x],scename[x]
            if x == 0:
                yield_data=net_data
            else:
                yield_data=np.vstack((yield_data,net_data))
        transpose array before plotting
        ield_data=np.transpose(yield_data)

1833        #X data for plot
1834       # myXList=[i+1 for i in range(3)]
1835        myXList=[i+1 for i in range(count)]
1836
1837        #Plotting
1838        fig = plt.figure()
1839        fig.suptitle('Yield Forecast', fontsize=14, fontweight='bold')
1840
1841        ax = fig.add_subplot(111)
1842        #fig.subplots_adjust(top=0.85)
1843        #ax.set_title('Yield Forecast')
1844        ax.set_xlabel('Scenario',fontsize=14)
1845        ax.set_ylabel('Yield [kg/ha]',fontsiz
1846
1847        # Plot a line between the means of each dataset
1848        plt.plot(myXList, obs, 'go-')
1849        ax.boxplot(yield_data,labels=scename, showmeans=True, meanline=True)
```

Get contents of labels in Frame 55

Count number of scenarios (by counting non-"Not added" labels). Full path of each output file (YIELD.txt) is added to a list "fname".

1) Use "loadtxt" from NumPy to read "YIELD.txt"

Last value of the output is from the observed weather data (in this case, hindcast mode)

Concatenate matrics read from loadtxt using "vstack" (Stack arrays in sequence vertically (row wise).)

2) Create a line plot using "obs" data ('go-' is properties of the plot-> 'g' means green color and 'o' means marker etc.

Create a boxplot using data saved in "yield_data". Show mean and meanline

1) Numpy

NumPy is the fundamental package for scientific computing with Python. It contains among other things:

- a powerful N-dimensional array object
- sophisticated (broadcasting) functions
- tools for integrating C/C++ and Fortran code
- useful linear algebra, Fourier transform, and random number capabilities

Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional

container of generic data. Arbitrary data-types can be defined. This allows NumPy to seamlessly and speedily integrate with a wide variety of databases.
(source: http://www.numpy.org/)

Original "YIELD.txt" output → After loading using "loadtxt"

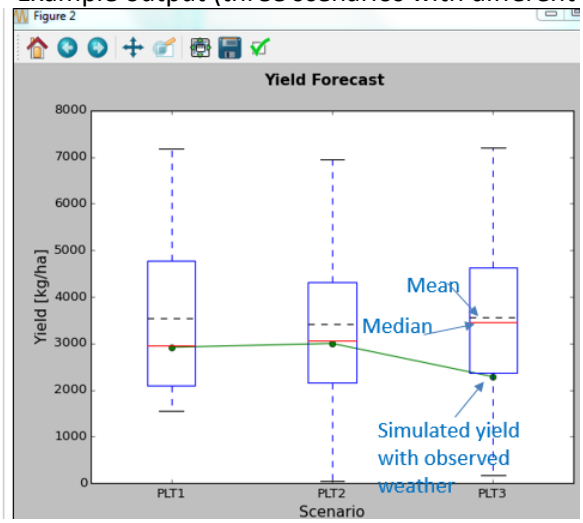
YIELD.txt

```
1613
1942
1555
1765
1868
1683
2777
2777
1105
1775
1976
3503
2106
2539
2135
2548
2279
2094
1652
1942
1664
2001
4999
1243
5781
2972
3960
3903
```

```
>>> temp_data = np.loadtxt('C:\\IRI\\PH\\CAMDT\\GUI_Test\\PLT1_output\\YIELD.txt')
>>> temp_data
array([ 1613.,  1942.,  1555.,  1765.,  1868.,  1683.,  2777.,  2777.,
        1105.,  1775.,  1976.,  3503.,  2106.,  2539.,  2135.,  2548.,
        2279.,  2094.,  1652.,  1942.,  1664.,  2001.,  4999.,  1243.,
        5781.,  2972.,  3960.,  3903.,  1989.,  3235.,  3797.,  3094.,
        4476.,  2525.,  2981.,  2394.,  2628.,  3503.,  6247.,  1855.,
         849.,  1705.,  5495.,  5685.,  5984.,  5058.,  4753.,  5887.,
        5013.,  6091.,  5799.,  5673.,  5665.,  5311.,  5278.,  4211.,
        5973.,  5189.,  2699.,  4099.,  5734.,  5906.,  2572.,  5092.,
        3412.,  5945.,  5922.,  3660.,  5620.,  4753.,  4815.,  4218.,
        4215.,  4517.,  5945.,  4649.,  3730.,  5160.,  4169.,  4659.,
        3753.,  3689.,  4119.,  3427.,  3510.,  3430.,  3295.,  4088.,
        4049.,  3888.,  3678.,  4219.,  3477.,  3118.,  3992.,  3943.,
        3383.,  3496.,  3744.,  3799.,  2203.])
>>> temp_data.__len__()
101
```

→ Original data from YIELD.txt is loaded as one long row as shown above. Therefore, the loaded data are vertically stacked (np.vstack) to create a matrix which will be an input to create multiple boxplots.

*Example output (three scenarios with different planting dates)



2) Pyplot/plot

matplotlib.pyplot is a collection of command style functions that make matplotlib work like MATLAB. Each pyplot function makes some change to a figure: e.g., create a figure, create a plotting area in a figure, plot some lines in a plotting area, decorate the plot with labels, etc.
plot() is a versatile command, and will take an arbitrary number of arguments. For example, to plot x versus y, you can issue the commanplt.

plot([1,2,3,4], [1,4,9,16])d:

(source: http://matplotlib.org/users/pyplot_tutorial.html)

**** Script for "WSI_Analysis" and "Risk_Analsys" functions are similar to "Yield_Analysis". Therefore detailed explanations are skipped.**