Appendix C

Example Scripts

C.1 Introduction

This appendix provides several example scripts to illustrate the use of scripting with HEC-DSSVue. Documentation for the calls in these scripts is provided in Chapter 8 of the HEC-DSSSVue User Manual. The intent of this appendix is to show variations in scripts and how to implement them.

Usually it is easiest to write a script by taking an example and modifying it to meet your needs. The scripts in this appendix are provided with the program in the directory "Samples". The data for the scripts is provided in the *sample.dss* file. Most scripts access this file in the "*C:\temp*" directory. To execute the scripts, you will need to copy sample.dss to the C:\temp directory, and the sample scripts to "*HecDssVue\Scripts*" directory, or you may import them with the Script Editor.

This section covers math scripts, export/import scripts, simple graphics, "headless" operation (graphics in a background mode), complex graphics and calling scripts from scripts.

C.2 Sample Math Scripts

Math scripts use the HecMath package, so no direct access to HEC-DSSVue or ListSelection is needed. You can also get a DataContainer and access the data directly. The two approaches are shown in the following two scripts, which both add the number "10" to each value.

AddTenTsc.py

```
#  Add 10 to each value using TimeSeriesContainer
#
from hec.script import *
from hec.heclib.dss import *
from hec.dataTable import *
from java import *

#  Open the file and get the data
try:
```

```
  dssFile = HecDss.open("C:/temp/sample.dss", "10MAR2006 2400,
09APR2006 2400")
  outflow = dssFile.get("/AMERICAN/FOLSOM/FLOW-RES
OUT/01JAN2006/1DAY/OBS/")
  i = 0
  for value in outflow.values :
    outflow.values[i] += 10.
    i += 1

  path = DSSPathname(outflow.fullName)
  fPart = path.fPart() + " + 10"
  path.setFPart(fPart)
  outflow.fullName = path.getPathname()

  dssFile.put(outflow)

except java.lang.Exception, e :
  #  Take care of any missing data or errors
MessageBox.showError(e.getMessage(), "Error reading data")
```

AddTenHecMath.py

```
#  Add 10 to each value using HecMath
#
from hec.script import *
from hec.hecmath import *
from java import *

#  Open the file and get the data
try:
  dssFile = DSS.open("C:/temp/sample.dss", "10MAR2006 2400,
09APR2006 2400")
  outflow = dssFile.read("/AMERICAN/FOLSOM/FLOW-RES
OUT/01JAN2006/1DAY/OBS/")
  newOutflow = outflow.add(10.)

  path = DSSPathname(newOutflow.getPath())
  fPart = path.fPart() + " + 10 HecMath"
  path.setFPart(fPart)
  newOutflow.setPathname(path.getPathname())

  dssFile.write(newOutflow)

except java.lang.Exception, e :
  #  Take care of any missing data or errors
   MessageBox.showError(e.getMessage(), "Error reading data")
```

> Unless the function you are using is not something not supported by HecMath, the HecMath functions are easier to use than manipulating the data directly in a DataContainer.

A routine functions suitable for scripting is estimating missing values. This is done with the HecMath function *estimateForMissingValues()*. An example is show for **EstimateMissing.py**.

```
#  Fill in missing values using HecMath
#
from hec.script import *
from hec.hecmath import *
from java import *
```

```
#  Open the file and get the data
try:
  dssFile = DSS.open("C:/temp/sample.dss", "10MAR2006 2400,
09APR2006 2400")
  flow = dssFile.read("/AMERICAN/FOLSOM/FLOW-RES
IN/01JAN2006/1DAY/OBS/")
  newflow = flow.estimateForMissingValues(10)
  path = DSSPathname(newflow.getPath())
  fPart = path.fPart() + " + FILLED"
  path.setFPart(fPart)
  newflow.setPathname(path.getPathname())
  dssFile.write( newflow)
except java.lang.Exception, e :
  #  Take care of any missing data or errors
   MessageBox.showError(e.getMessage(), "Error reading data")
```

Another useful function is smoothing, typically used for reservoir data. Of course this could have been just an additional line added to the EstimateMissing.py script.

Smoothing.py

```
#  Smooth values using HecMath
from hec.script import *
from hec.hecmath import *
from hec.heclib.dss import *
from java import *

#  Open the file and get the data
try:
  dssFile = DSS.open("C:/temp/sample.dss", "10MAR2006 2400,
09APR2006 2400")
  flow = dssFile.read("/AMERICAN/FOLSOM/FLOW-RES
IN/01JAN2006/1DAY/OBS/")
  newflow = flow.centeredMovingAverage(7, 1, 0)
  path = DSSPathname(newflow.getPath())
  fPart = path.fPart() + " + SMOOTH"
  path.setFPart(fPart)
  newflow.setPathname(path.getPathname())
  dssFile.write( newflow)
except java.lang.Exception, e :
  #  Take care of any missing data or errors
MessageBox.showError(e.getMessage(), "Error reading data")
```

C.3 Sample Import and Export Scripts

You can write scripts to import or export data with various formats. You can also write scripts to display a data entry table that can be used to enter data manually.

C.3.1 Manually Entry Scripts

Both the manual time series data entry screen and paired data entry screen can be brought up from a script. They both can be passed a pathname to preset the dialog, and the time series dialog can be given a start time. For

example, to display an initialized time series data entry screen (Figure C.1), the following **DataEntry.py** script can be used:

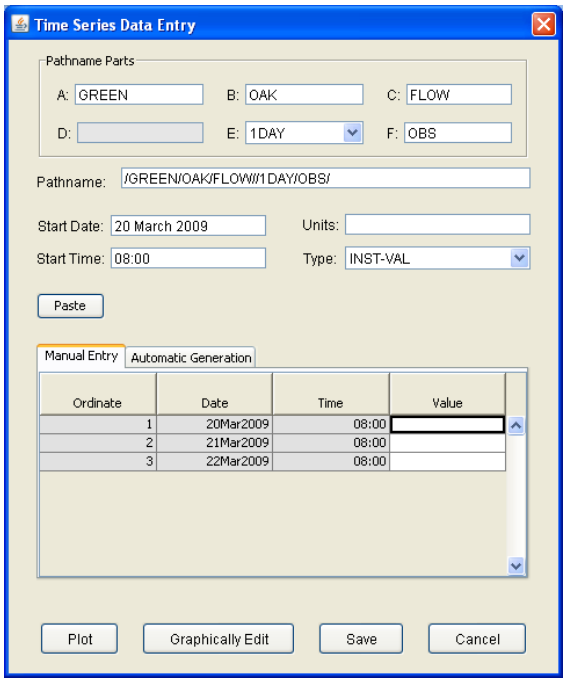


Figure C.1 Time Series Data Entry Screen

```
from hec.script import *
from hec.dssgui import *
from hec.heclib.util import *
import java

ls = ListSelection.getMainWindow()
ls.setIsInteractive(1,0)  # Turn off popups
ls.open("C:/temp/mydb.dss")

time = HecTime()
time.setCurrent()
time.setTime("0800")
time.addDays(-5)

ls.timeSeriesDataEntry("/GREEN/OAK/FLOW//1DAY/OBS",
time.dateAndTime(4))
# ls.finish()  # Batch mode only
```

C.3.2 SHEF Import/Export Scripts

If the SHEF parameter files are setup correctly, a simple script can be written to import or export SHEF data. Essentially, the same steps are followed in the script as would be done interactively through the ListSelection class. For example, **FolsomToShef.py**:

```
from hec.script import *
from hec.heclib.dss import *
from hec.dssgui import *
import java
```

```
#  Open the file and get the data
try:
  dssFile = HecDss.open("C:/temp/sample.dss", "10MAR2006 2400, 09APR2006
2400")
  precip = dssFile.get("/AMERICAN/FOLSOM/PRECIP-
BASIN/01JAN2006/1DAY/OBS/")
  stor = dssFile.get("/AMERICAN/FOLSOM/ STOR-RES EOP/01JAN2006/1DAY/OBS/")
  topcon = dssFile.get("/AMERICAN/FOLSOM/TOP CON
STOR/01JAN2006/1DAY/OBS/")
  sagca = dssFile.get("/AMERICAN/FOLSOM-SAGCA/TOP CON
STOR/01JAN2006/1DAY/OBS/")
  inflow = dssFile.get("/AMERICAN/FOLSOM/FLOW-RES IN/01JAN2006/1DAY/OBS/")
  outflow = dssFile.get("/AMERICAN/FOLSOM/FLOW-RES
OUT/01JAN2006/1DAY/OBS/")
except java.lang.Exception, e :
  #  Take care of any missing data or errors
   MessageBox.showError(e.getMessage(), "Error reading data")

#  Add Data
datasets = java.util.Vector()
datasets.add(precip)
datasets.add(stor)
datasets.add(topcon)
datasets.add(sagca)
datasets.add(inflow)
datasets.add(outflow)

ls = ListSelection.getMainWindow()
ls.setIsInteractive(1,0)  # Turn off popups
ls.exportShef("C:/temp/FolsomShefData.shef", datasets)

#ls.finish()  # Batch mode only
```

The file *FolsomShefData.shef* will contain:

```
  .E FOLSOM 200906 Z DY0603102400/PR/DID1/0.79000002/0.31999999/0.25999999
  .E FOLSOM 200906 Z DY0603132400/PR/DID1/0.20999999/1.08/0.050000000/0.30000001
  .E FOLSOM 200906 Z DY0603172400/PR/DID1/0.76999998/0.039999990/0.020000000
  .E FOLSOM 200906 Z DY0603202400/PR/DID1/0.38999999/0.14/0.020000000/0.020000000
```

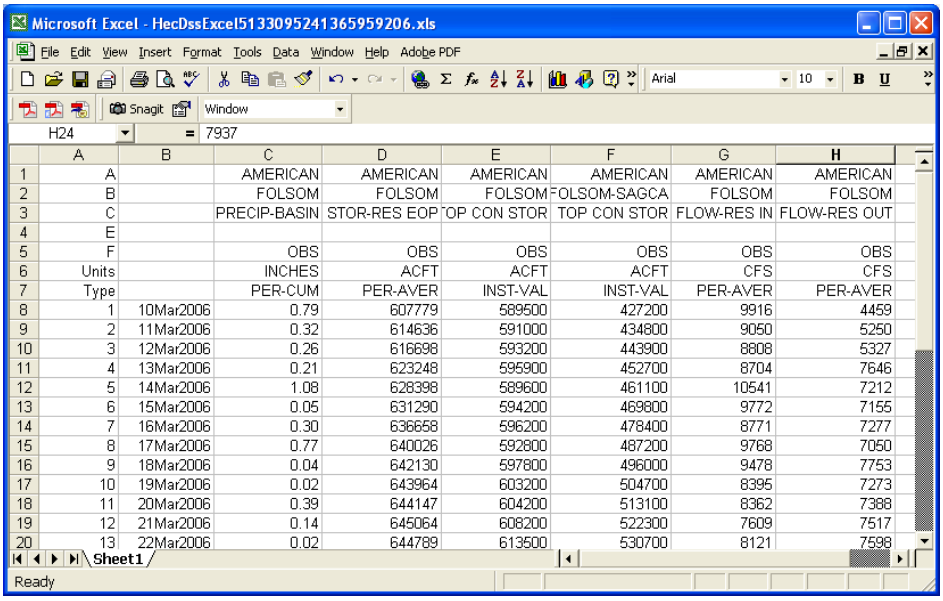
Instead of reading the data and passing to exportShef(), the time window could have been set and the pathnames set selected in ListSelection just as well.

Importing SHEF data is easy; you essentially call the function *importShef(String fileName)*, passing in the name of the file to import.

```
from hec.script import *
from hec.dssgui import *
import java

ls = ListSelection.getMainWindow()
ls.setIsInteractive(1,0)  # Turn off popups
ls.open("C:/temp/myDb.dss")
ls.importShef("C:/temp/FolsomShefData.shef")
#ls.finish()  # Batch mode only
```

C.3.3 Exporting to Excel

The export to Excel function uses code from HecDataTable; writing to Excel is similar to creating a table. The script **FolsomExcel.py** not only puts the data in an Excel file, but runs Excel too (Figure C.2):



	A	B	C	D	E	F	G	H
1	A		AMERICAN	AMERICAN	AMERICAN	AMERICAN	AMERICAN	AMERICAN
2	B		FOLSOM	FOLSOM	FOLSOM	FOLSOM-SAGCA	FOLSOM	FOLSOM
3	C		PRECIP-BASIN	STOR-RES EOP	TOP CON STOR	TOP CON STOR	FLOW-RES IN	FLOW-RES OUT
4	E							
5	F		OBS	OBS	OBS	OBS	OBS	OBS
6	Units		INCHES	ACFT	ACFT	ACFT	CFS	CFS
7	Type		PER-CUM	PER-AVER	INST-VAL	INST-VAL	PER-AVER	PER-AVER
8	1	10Mar2006	0.79	607779	589500	427200	9916	4459
9	2	11Mar2006	0.32	614636	591000	434800	9050	5250
10	3	12Mar2006	0.26	616698	593200	443900	8808	5327
11	4	13Mar2006	0.21	623248	595900	452700	8704	7646
12	5	14Mar2006	1.08	628398	589600	461100	10541	7212
13	6	15Mar2006	0.05	631290	594200	469800	9772	7155
14	7	16Mar2006	0.30	636658	596200	478400	8771	7277
15	8	17Mar2006	0.77	640026	592800	487200	9768	7050
16	9	18Mar2006	0.04	642130	597800	496000	9478	7753
17	10	19Mar2006	0.02	643964	603200	504700	8395	7273
18	11	20Mar2006	0.39	644147	604200	513100	8362	7388
19	12	21Mar2006	0.14	645064	608200	522300	7609	7517
20	13	22Mar2006	0.02	644789	613500	530700	8121	7598

Figure C.2 Export to Excel

If you would rather specify the Excel file name, you can do that by giving the name of the file following the datasets. You can also just create an Excel file, without running it, by using the function createExcelFile. This is shown in the script **FolsomSaveExcel.py**:

```
from hec.script import *
from hec.heclib.dss import *
from hec.dataTable import *
import java


#  Open the file and get the data
try:
  dssFile = HecDss.open("C:/temp/sample.dss", "10MAR2006 2400, 09APR2006
2400")
  precip = dssFile.get("/AMERICAN/FOLSOM/PRECIP-
BASIN/01JAN2006/1DAY/OBS/")
  stor = dssFile.get("/AMERICAN/FOLSOM/ STOR-RES EOP/01JAN2006/1DAY/OBS/")
  topcon = dssFile.get("/AMERICAN/FOLSOM/TOP CON
STOR/01JAN2006/1DAY/OBS/")
  sagca = dssFile.get("/AMERICAN/FOLSOM-SAGCA/TOP CON
STOR/01JAN2006/1DAY/OBS/")
  inflow = dssFile.get("/AMERICAN/FOLSOM/FLOW-RES IN/01JAN2006/1DAY/OBS/")
  outflow = dssFile.get("/AMERICAN/FOLSOM/FLOW-RES
OUT/01JAN2006/1DAY/OBS/")
except java.lang.Exception, e :
  #  Take care of any missing data or errors
   MessageBox.showError(e.getMessage(), "Error reading data")

#  Add Data
datasets = java.util.Vector()
datasets.add(precip)
datasets.add(stor)
datasets.add(topcon)
datasets.add(sagca)
datasets.add(inflow)
datasets.add(outflow)

#  For this code, jython sees a List before a Vector
#list = java.awt.List()
list = []
list.append(datasets)

table = HecDataTableToExcel.newTable()
```

```
#  If you want to run Excel with a specific name and not a temp name:
table.runExcel(list, "myWorkbook.xls")
#  Or, if you would just rather create an Excel file, but not run it:
#table.createExcelFile(datasets, "myWorkbook.xls")
```

C.3.4 Importing Other Formats

If you can parse a format, you can write a script to enter your data into HEC-DSS by using a DataContainer. DataContainers are generic database independent classes that hold data used by the various functions in HEC-DSSVue.

The following example, WriteDss.py, shows the use of a TimeSeriesContainer with made-up data. In this example, HecTime is used to generate the times needed and the TimeSeriesContainer is imported from hec.io:

```
from hec.script import *
from hec.heclib.dss import *
from hec.heclib.util import *
from hec.io import *
import java

try :
  try :
    myDss = HecDss.open("C:/temp/test.dss")
    tsc = TimeSeriesContainer()
    tsc.fullName = "/BASIN/LOC/FLOW//1HOUR/OBS/"
    start = HecTime("04Sep1996", "1330")
    tsc.interval = 60
    flows = [0.0,2.0,1.0,4.0,3.0,6.0,5.0,8.0,7.0,9.0]
    times = []
    for value in flows :
      times.append(start.value())
      start.add(tsc.interval)
    tsc.times = times
    tsc.values = flows
    tsc.numberValues = len(flows)
    tsc.units = "CFS"
    tsc.type = "PER-AVER"
    myDss.put(tsc)

  except Exception, e :
    MessageBox.showError(' '.join(e.args), "Python Error")
  except java.lang.Exception, e :
    MessageBox.showError(e.getMessage(), "Error")
finally :
  myDss.close()
```

C.3.5 Sample Graphics Scripts

Generally, graphics scripts may be broken into five segments; 1) Retrieve the data; 2) Initialize the plot; 3) Bring the plot into existence with *showPlot()*; 4) Change the plot; and 5) Save the plot to file and close.

Most people are tripped up with the function *showPlot()*. This function puts all the parts together and creates the plot. One would think that this would be called at the end of the script, but to the contrary, it needs to be called near the beginning. For example, you can not change or set a curve's color until the curve exists, and the showPlot() function is what creates curves. Scripting is emulating the steps that you would do interactively; it is not a *command* language.

HecDss uses *exceptions* for error processing, such as indicating missing data. You need to use *try: except* loops to catch errors, otherwise an exception message will be written to the output and that exception may not be very clear.

A simple plot (Figure C.3) example that illustrates these points is one for Oakville:

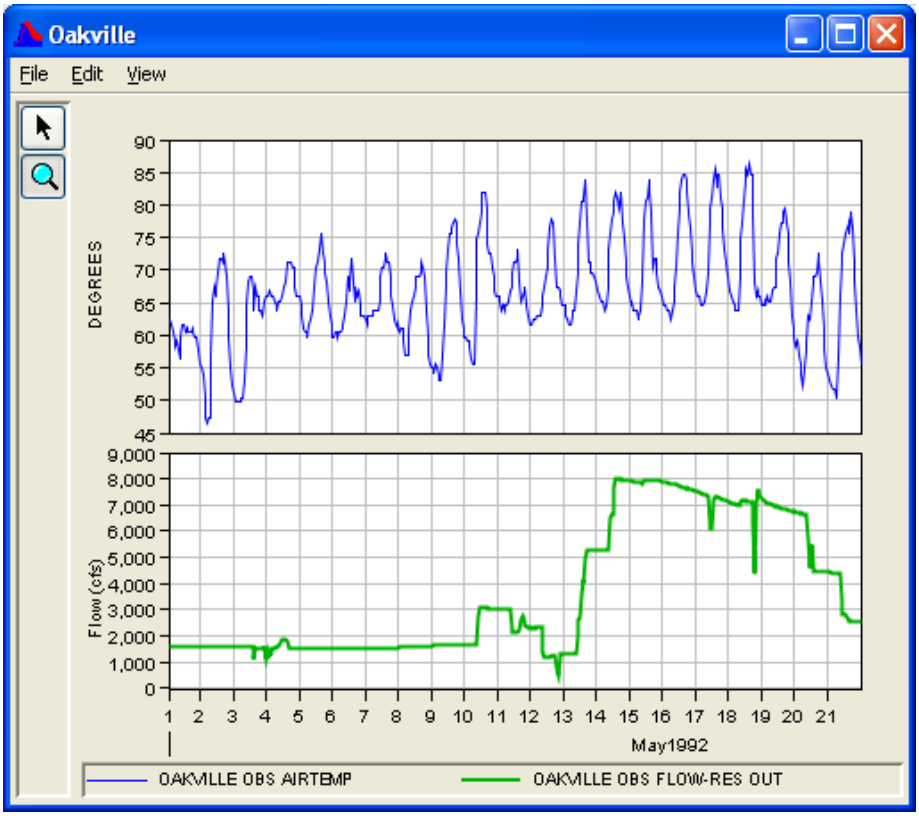


Figure C.3 Oakville Output - Plot

```
from hec.script import *      # always needed
from hec.heclib.dss import *  # Used to access HEC-DSS
import java                   #  Provide access to java

#  Open the file and get the data
try :
  dssFile = HecDss.open("C:/temp/sample.dss")
  airTemp = dssFile.get("/GREEN
RIVER/OAKVILLE/AIRTEMP/01MAY1992/1HOUR/OBS/")
  outflow = dssFile.get("/GREEN RIVER/OAKVILLE/FLOW-RES
OUT/01MAY1992/1HOUR/OBS/")
```

```
  dssFile.done()
except java.lang.Exception, e :
  #  Take care of any missing data or errors
   MessageBox.showError(e.getMessage(), "Error reading data")

#  Initialize the plot and add data
plot = Plot.newPlot("Oakville")
plot.addData(airTemp)
plot.addData(outflow)

#  Create plot
plot.setSize(600,600)
plot.setLocation(100,100)  #  Move off screen if you don't want
to see it
plot.showPlot()

#  Change the plot
outCurve = plot.getCurve(outflow)
outCurve.setLineColor("darkgreen")

#  Save the plot and close
plot.saveToPng("C:/temp/Oakville.png")
#plot.close()  #  Do this if you only want the plot
#                as a .png and not on screen
```

This example shows a more complex plot where we break the rule of *showPlot()* of creating the plot prior to changing it. In this case, *configurePlotLayout()* will also create the plot. In this plot of Folsom Lake (Figure C.4), we create three separate viewports and size them according to the different data types. Without too much work, we produce a pretty fancy plot.

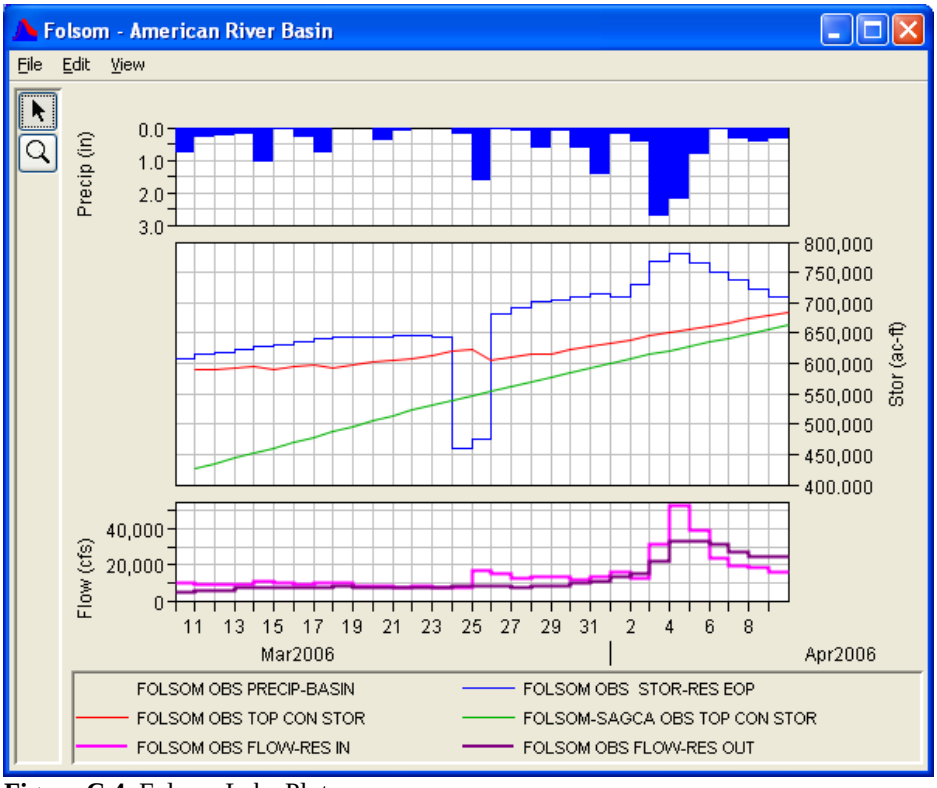


Figure C.4 Folsom Lake Plot

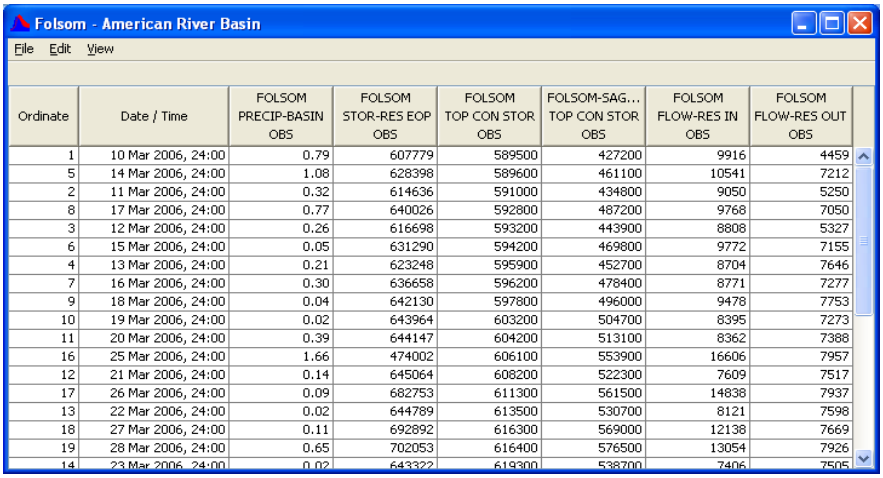
Folsom.py

```
from hec.script import *
from hec.script.Constants import TRUE, FALSE
from hec.heclib.dss import *
import java

#  Open the file and get the data
try:
  dssFile = HecDss.open("C:/temp/sample.dss", "10MAR2006 2400,
09APR2006 2400")
  precip = dssFile.get("/AMERICAN/FOLSOM/PRECIP-
BASIN/01JAN2006/1DAY/OBS/")
  stor = dssFile.get("/AMERICAN/FOLSOM/ STOR-RES
EOP/01JAN2006/1DAY/OBS/")
  topcon = dssFile.get("/AMERICAN/FOLSOM/TOP CON
STOR/01JAN2006/1DAY/OBS/")
  sagca = dssFile.get("/AMERICAN/FOLSOM-SAGCA/TOP CON
STOR/01JAN2006/1DAY/OBS/")
  inflow = dssFile.get("/AMERICAN/FOLSOM/FLOW-RES
IN/01JAN2006/1DAY/OBS/")
  outflow = dssFile.get("/AMERICAN/FOLSOM/FLOW-RES
OUT/01JAN2006/1DAY/OBS/")
except java.lang.Exception, e :
  #  Take care of any missing data or errors
   MessageBox.showError(e.getMessage(), "Error reading data")

#  Initialize the plot and set viewport size in precent
plot = Plot.newPlot("Folsom - American River Basin")
layout = Plot.newPlotLayout()
topView = layout.addViewport(10.)
middleView = layout.addViewport(60.)
bottomView = layout.addViewport(30.)

#  Add Data in specific viewports
topView.addCurve("Y1", precip)
middleView.addCurve("Y2", stor)
middleView.addCurve("Y2", topcon)
middleView.addCurve("Y2", sagca)
bottomView.addCurve("Y1", inflow)
bottomView.addCurve("Y1", outflow)

panel = plot.getPlotpanel()
prop = panel.getProperties()
prop.setViewportSpaceSize(0)

#  Break our first rule - actually this creates the plot to
change
plot.configurePlotLayout(layout)

panel = plot.getPlotpanel()
prop = panel.getProperties()
prop.setViewportSpaceSize(0)

#  Invert the precip and make pretty
view0 = plot.getViewport(0)
yaxis = view0.getAxis("Y1")
yaxis.setReversed(FALSE)
precipCurve = plot.getCurve(precip)
precipCurve.setFillColor("blue")
precipCurve.setFillType("Above")
precipCurve.setLineVisible(FALSE)
```

```
#  Set the inflow and outflow colors
inflowCurve = plot.getCurve(inflow)
inflowCurve.setLineColor("magenta")
outflowCurve = plot.getCurve(outflow)
outflowCurve.setLineColor("purple")

plot.showPlot()
```

C.4 Sample Table Script

Creating a table from a script is easier than plots as there is less to do. Like plots, you retrieve the data, add the data to a *Vector*, create a table and set the data, and then show the table. The same data shown in the plot is easily displayed in tabular form (Figure C.5) by the following **FolsomTable.py** script:



Ordinate	Date / Time	FOLSOM PRECIP-BASIN OBS	FOLSOM STOR-RES EOP OBS	FOLSOM TOP CON STOR OBS	FOLSOM-SAG... TOP CON STOR OBS	FOLSOM FLOW-RES IN OBS	FOLSOM FLOW-RES OUT OBS
1	10 Mar 2006, 24:00	0.79	607779	589500	427200	9916	4459
5	14 Mar 2006, 24:00	1.08	628398	589600	461100	10541	7212
2	11 Mar 2006, 24:00	0.32	614636	591000	434800	9050	5250
8	17 Mar 2006, 24:00	0.77	640026	592800	487200	9768	7050
3	12 Mar 2006, 24:00	0.26	616698	593200	443900	8808	5327
6	15 Mar 2006, 24:00	0.05	631290	594200	469800	9772	7155
4	13 Mar 2006, 24:00	0.21	623248	595900	452700	8704	7646
7	16 Mar 2006, 24:00	0.30	636658	596200	478400	8771	7277
9	18 Mar 2006, 24:00	0.04	642130	597800	496000	9478	7753
10	19 Mar 2006, 24:00	0.02	643964	603200	504700	8395	7273
11	20 Mar 2006, 24:00	0.39	644147	604200	513100	8362	7388
16	25 Mar 2006, 24:00	1.66	474002	606100	553900	16606	7957
12	21 Mar 2006, 24:00	0.14	645064	608200	522300	7609	7517
17	26 Mar 2006, 24:00	0.09	682753	611300	561500	14838	7937
13	22 Mar 2006, 24:00	0.02	644789	613500	530700	8121	7598
18	27 Mar 2006, 24:00	0.11	692892	616300	569000	12138	7669
19	28 Mar 2006, 24:00	0.65	702053	616400	576500	13054	7926
14	23 Mar 2006, 24:00	0.02	643322	619300	538700	7406	7505

Figure C.5 Folsom Lake Tabular Display

```
from from hec.script import *
from hec.heclib.dss import *
from hec.dataTable import *
import java

#  Open the file and get the data
try:
  dssFile = HecDss.open("C:/temp/sample.dss", "10MAR2006 2400,
09APR2006 2400")
  precip = dssFile.get("/AMERICAN/FOLSOM/PRECIP-
BASIN/01JAN2006/1DAY/OBS/")
  stor = dssFile.get("/AMERICAN/FOLSOM/ STOR-RES
EOP/01JAN2006/1DAY/OBS/")
  topcon = dssFile.get("/AMERICAN/FOLSOM/TOP CON
STOR/01JAN2006/1DAY/OBS/")
  sagca = dssFile.get("/AMERICAN/FOLSOM-SAGCA/TOP CON
STOR/01JAN2006/1DAY/OBS/")
  inflow = dssFile.get("/AMERICAN/FOLSOM/FLOW-RES
IN/01JAN2006/1DAY/OBS/")
  outflow = dssFile.get("/AMERICAN/FOLSOM/FLOW-RES
OUT/01JAN2006/1DAY/OBS/")
except java.lang.Exception, e :
  #  Take care of any missing data or errors
   MessageBox.showError(e.getMessage(), "Error reading data")
```

```
      #  Add Data
datasets = java.util.Vector()
datasets.add(precip)
datasets.add(stor)
datasets.add(topcon)
datasets.add(sagca)
datasets.add(inflow)
datasets.add(outflow)

table = HecDataTableFrame.newTable("Folsom - American River Basin")
table.setData(datasets)
table.showTable()
```

C.5 Complex Graphics Scripts

A report-ready script for displaying monthly reservoir data and a script that uses arguments to define what is to be plotted are presented in the following sections.

Note: *An important detail to keep in mind is that you need to call showPlot() early on. Scripting emulates what you would do interactively; it is not a "command driven" procedure. You cannot change features on a plot until they exist and the showPlot() function causes the plot object to come into existence.*

C.5.1 Coyote Valley Dam Reservoir Plot

This is a plot showing reservoir data for the month of March (Figure C.6). The plot shows precipitation, storage, top of conservation, inflow, outflow and downstream flows.

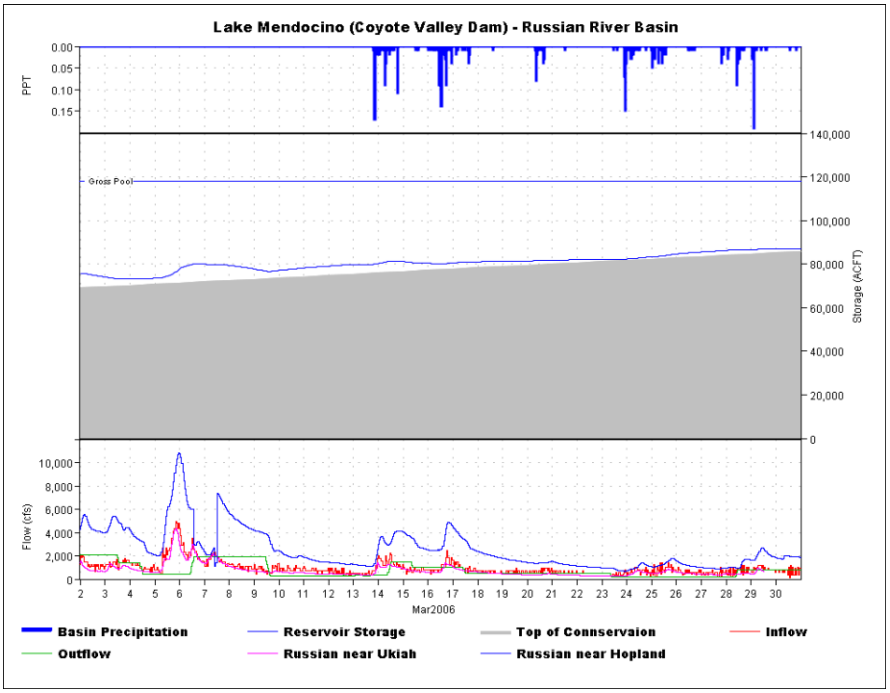


Figure C.6 Mendocino Reservoir Data for March

The script for Coyote Valley is:

```
from hec.script import *
from hec.script.Constants import TRUE, FALSE
from hec.heclib.dss import *
import java

#  grid pattern - 2 solid pixels followed by 8 blank ones
dotPat = [2., 8.]
datasets = []

#  Get the data
Try :
  dssFile = HecDss.open("C:/temp/sample.dss", "01MAR2006 2400,
30MAR2006 2400")
  precip = dssFile.get("/EF RUSSIAN/COYOTE/PRECIP-
INC/01MAR2006/1HOUR/TB/")
  datasets.append(precip)
  stor = dssFile.get("/EF RUSSIAN/COYOTE/STOR-RES
EOP/01MAR2006/1HOUR//")
  datasets.append(stor)
  topcon = dssFile.get("/EF RUSSIAN/COYOTE/TOP CON
STOR/01JAN2006/1DAY//")
  datasets.append(topcon)
  inflow =dssFile.get("/EF RUSSIAN/COYOTE/FLOW-RES
IN/01MAR2006/1HOUR/SMOOTH/")
  datasets.append(inflow)
  outflow = dssFile.get("/EF RUSSIAN/COYOTE/FLOW-RES
OUT/01MAR2006/1HOUR//")
  datasets.append(outflow)
  ukiah = dssFile.get("/RUSSIAN/NR UKIAH/FLOW/01MAR2006/1HOUR//")
  datasets.append(ukiah)
  hopland = dssFile.get("/RUSSIAN/NR HOPLAND/FLOW/01MAR2006/1HOUR//")
  datasets.append(hopland)
       except java.lang.Exception, e :
        MessageBox.showError(e.getMessage(), "Error reading
data")

#  Create the view ports
plot = Plot.newPlot("Lake Mendocino")
layout = Plot.newPlotLayout()
topView = layout.addViewport(10.)
middleView = layout.addViewport(60.)
bottomView = layout.addViewport(30.)

#  Add the data
topView.addCurve("Y1", precip)
middleView.addCurve("Y2", stor)
middleView.addCurve("Y2", topcon)
bottomView.addCurve("Y1", inflow)
bottomView.addCurve("Y1", outflow)
bottomView.addCurve("Y1", ukiah)
bottomView.addCurve("Y1", hopland)
plot.configurePlotLayout(layout)

#  For headless operation (batch mode), draw the plot off the screen
plot.setLocation(-10000, -10000)
plot.setSize(1000, 800)

####### Important – showPlot() creates the plot objects ######
# (You cannot set or change things that do not exist yet)
plot.showPlot()
```

```
#  Make the legend labels look nice
for dataset in datasets:
label = plot.getLegendLabel(dataset)
label.setFontSize(16)
label.setFont("Arial Black,Plain,14")
label.setForeground("black")

# Set the label text
label = plot.getLegendLabel(precip)
label.setText( "Basin Precipitation")
label = plot.getLegendLabel(stor)
label.setText("Reservoir Storage")
label = plot.getLegendLabel(topcon)
label.setText("Top of Connservaion")
label = plot.getLegendLabel(inflow)
label.setText("Inflow")
label = plot.getLegendLabel(outflow)
label.setText("Outflow")
label = plot.getLegendLabel(ukiah)
label.setText("Russian near Ukiah")
label = plot.getLegendLabel(hopland)
label.setText("Russian near Hopland")

#  Set the plot title
plot.setPlotTitleText("Lake Mendocino (Coyote Valley Dam) - Russian
River Basin")
tit = plot.getPlotTitle()
tit.setFont("Arial Black")
tit.setFontSize(18)
plot.setPlotTitleVisible(TRUE)

#  Make the viewports right next to each other
panel = plot.getPlotpanel()
panel.setHorizontalViewportSpacing(0)

#  Set the Y axis labels
view0 = plot.getViewport(0)
yaxis = view0.getAxis("Y1")
yaxis.setReversed(FALSE)
yaxis.setLabel("PPT")
view1 = plot.getViewport(1)
yaxis1 = view1.getAxis("Y2")
yaxis1.setLabel("Storage (ACFT)")
yaxis1.setScaleLimits(0., 140000.)
yaxis1.setViewLimits(0., 140000.)

#  Mark the gross pool level
marker = AxisMarker()
marker.axis = "Y"
marker.value = "118000"
marker.labelText = "Gross Pool"
marker.lineColor = "Blue"
marker.labelPosition = "center"
view1.addAxisMarker(marker)

#  Set the curve colors and fill
precipCurve = plot.getCurve(precip)
precipCurve.setFillColor("blue")
precipCurve.setFillType("Above")
precipCurve.setLineColor("blue")

conCurve = plot.getCurve(topcon)
conCurve.setFillColor("lightGray")
conCurve.setFillType("Below")
```

```
conCurve.setLineColor("lightGray")

storCurve = plot.getCurve(stor)
storCurve.setLineColor("blue")
storCurve.setLineWidth(1.)


outCurve = plot.getCurve(outflow)
       outCurve.setLineColor("darkgreen")
       outCurve.setLineWidth(1.)

inCurve = plot.getCurve(inflow)
inCurve.setLineColor("red")
inCurve.setLineWidth(1.)

ukiahCurve = plot.getCurve(ukiah)
ukiahCurve.setLineColor("magenta")
ukiahCurve.setLineWidth(1.)

hoplandCurve = plot.getCurve(hopland)
hoplandCurve.setLineColor("blue")
hoplandCurve.setLineWidth(1.)

#  Set grid style
view0 = plot.getViewport(0)
prop = view0.getProperties()
prop.setMajorXGridStyle(dotPat)
prop.setMajorYGridStyle(dotPat)
view1 = plot.getViewport(1)
prop = view1.getProperties()
prop.setMajorXGridStyle(dotPat)
prop.setMajorYGridStyle(dotPat)
view2 = plot.getViewport(2)
prop = view2.getProperties()
prop.setMajorXGridStyle(dotPat)
prop.setMajorYGridStyle(dotPat)

#  Now that it is complete, save to a png and close it
plot.saveToPng("C:/temp/Coyote.png")
plot.close()
```

C.5.2 Scripts with Arguments

Often you will want to setup a plot or other function and apply it to different locations, variables or times. Instead of duplicating the script for each instance, which leads to maintenance and manageability issues, it is usually advantageous to write one script and pass in arguments that may vary.

To access arguments passed into the script you must include the following: import sys (sm = globals())

Arguments are passed into a python script using a "name-space dictionary", which gives the keyword followed by a colon and then the parameter. Inside the script, the keyword will be replaced with the parameter. For example, in the GagePlot script, the name-space dictionary could be:

```
"location" : "Glenfir", "version" : "OBS"
```

Wherever the string location or version is in the script will be substituted with GlenFir (Figure C.7) and Obs, respectively.

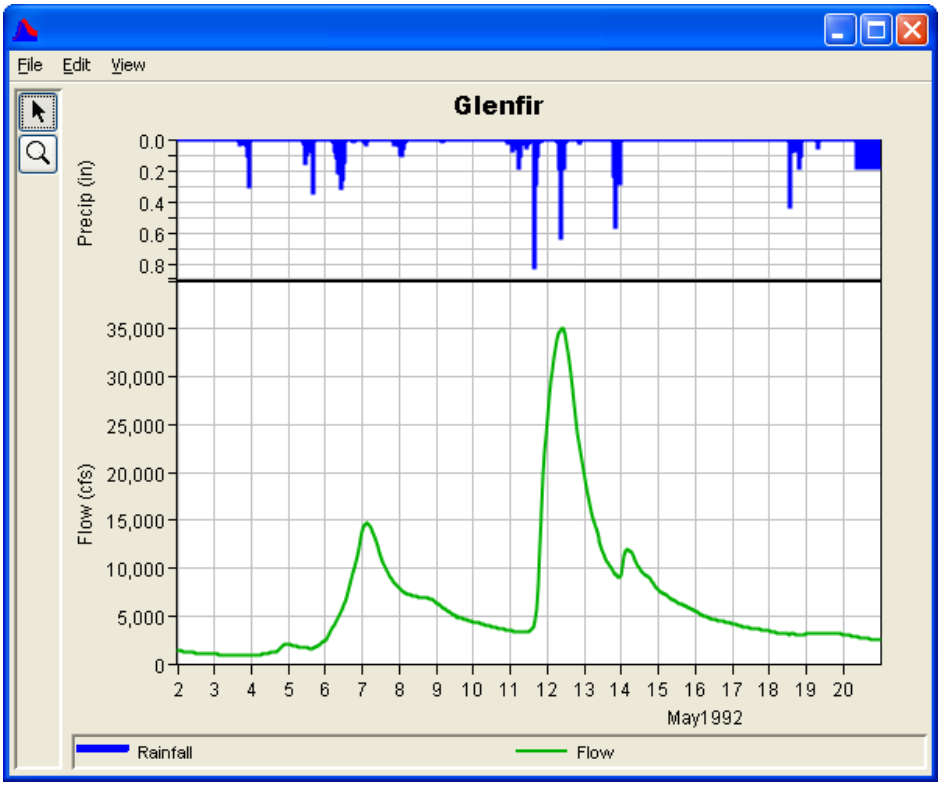


Figure C.7 Glenfir Plot

The GagePlot script is designed to produce a standard plot for a gage where there is both a stream flow gage and a precipitation gage. Here you can have one script that you call from another script multiple times, with a different argument for the location. The individual scripts may be run using the function "exec" for in-line scripts or "execFile" for scripts on disk. For example, a RunGages script can be written to run GagePlot for several locations:

Script RunGages

```
from hec.script import *

glenFirArgs  = {"location" : "Glenfir", "version" : "OBS"}
execfile("C:/HecDSSVueDev/HecDssVue/scripts/GagePlot.py", {},
glenFirArgs)

madronArgs   = {"location" : "Madron", "version" : "OBS"}
execfile("C:/HecDSSVueDev/HecDssVue/scripts/GagePlot.py", {},
madronArgs)

oakTreeArgs   = {"location" : "Madron", "version" : "OBS"}
execfile("C:/HecDSSVueDev/HecDssVue/scripts/GagePlot.py", {},
oakTreeArgs)
```

The **GagePlot** Script is:

```
# Both location=name version=ver must be defined as arguments
# To run:  GagePlot location=Glenfir version=OBS

from hec.script import *
from hec.heclib.dss import *
import sys
import java
from hec.script.Constants import TRUE, FALSE

# Access arguments
sm = globals()

#  "location" and "version" will be replaced with the arguments
passed in

#  Retrieve data
try :
  #  If you wanted to use a relative time window, you could do
something like:
  #  dssFile = HecDss.open("C:/temp/sample.dss", "T-30D, T")
  dssFile = HecDss.open("C:/temp/sample.dss","01MAY1992
2400",”20MAY1992 2400")
  flowPath = "/GREEN RIVER/" + location + "/FLOW//1HOUR/OBS/"
  precipPath = "/GREEN RIVER/" + location + "/PRECIP-
INC//1HOUR/OBS/"
  precip = dssFile.get(precipPath)
  flow = dssFile.get(flowPath)
except java.lang.Exception, e :
    MessageBox.showError(e.getMessage(), "Error reading data")

# Create plot and viewports
plot = Plot.newPlot()
layout = Plot.newPlotLayout()
topView = layout.addViewport(15.)
bottomView = layout.addViewport(85.)

#  Add data
topView.addCurve("Y1", precip)
bottomView.addCurve("Y1", flow)
plot.configurePlotLayout(layout)
plot.setSize(600, 500)

#  Add title
plot.setPlotTitleText(location)
tit = plot.getPlotTitle()
tit.setFont("Arial Black")
tit.setFontSize(18)
plot.setPlotTitleVisible(TRUE)

#  This actually creates the plot - cannot access any components
until done
plot.showPlot()

#  Now we can change components
panel = plot.getPlotpanel()
#  Remove space between viewports
panel.setHorizontalViewportSpacing(0)

#  Invert the precip
view0 = plot.getViewport(0)
yaxis = view0.getAxis("Y1")
yaxis.setReversed(FALSE)
```

```
#  Make the precip pretty
precipCurve = plot.getCurve(precip)
precipCurve.setFillColor("blue")
precipCurve.setFillType("Above")
precipCurve.setLineColor("blue")

#  Make the flow pretty
flowCurve = plot.getCurve(flow)
flowCurve.setLineColor("darkgreen")
flowCurve.setLineWidth(2.)

#  Set the legend labels
plot.setLegendLabelText(flow, "Flow")
plot.setLegendLabelText(precip, "Rainfall")

# Done (should look like we want now)

#  If you wanted this to run in the background, you could add
the following
#  plot.setLocation(-10000, -10000)
#  plot.saveToPng("C:/temp/GagePlot.png")
#  plot.close()
```

Another example of calling scripts from scripts is the **script tester.py**:

```
from hec.script import *
import time
import java

#--------------------#
# some code to eval() #
#--------------------#
getCurrentTimeSnippet = "time.ctime(time.time())"

#-----------------#
# a simple script #
#-----------------#
scriptText = '''
def myFunc() :
     global startTime, endTime
     startTime = time.time()
     for i in range(3) :
            print("Sleeping for 1 second at %f" % time.time())
            time.sleep(1)
     endTime = time.time()

myFunc()
print("Start time = %s" % time.ctime(startTime))
print("End time   = %s" % time.ctime(endTime))
'''

#-------------------------------#
# write the script out to a file #
#-------------------------------#
scriptFilename = "myFunc.py"
scriptFile = open(scriptFilename, "w")
scriptFile.write(scriptText)
scriptFile.close()

#---------------------------------#
# Use eval() to evaluate some code #
#---------------------------------#
```

```
print("\nCalling eval()")
print("The local time is %s" % eval(getCurrentTimeSnippet))

#---------------------------------------#
# set up a separate scope for execution #
#---------------------------------------#
otherGlobals = {"time" : time, "startTime" : None, "endTime" :
None}
otherLocals  = {}

#-----------------------------------#
# exec some text in a separate scope #
#-----------------------------------#
print("\nCalling exec in a separate scope")
exec scriptText in otherGlobals, otherLocals
try    : print("Global variable startTime = %f" % startTime)
except : print("Global variable  startTime is not defined.")
try    : print("Global variable endTime   = %f" % endTime)
except : print("Global variable  endTime is not defined.")

#-------------------------------------------#
#execfile the same script in a separate scope #
#-------------------------------------------#
print("\nCalling execfile in a separate scope")
execfile(scriptFilename, otherGlobals, otherLocals)
try    : print("Global variable startTime = %f" % startTime)
except : print("Global variable  startTime is not defined.")
try    : print("Global variable endTime   = %f" % endTime)
except : print("Global variable  endTime is not defined.")

#-----------------------------------#
# exec the same text in our own scope #
#-----------------------------------#
print("\nCalling exec in the current scope")
exec scriptText
try    : print("Global variable startTime = %f" % startTime)
except : print("Global variable  startTime is not defined.")
try    : print("Global variable endTime   = %f" % endTime)
except : print("Global variable  endTime is not defined.")

#---------------------------------------#
#execfile the same script in our own scope #
#---------------------------------------#
print("\nUndefining global variables startTime and endTime")
del globals()['startTime']
del globals()['endTime']
print("\nCalling execfile in the current scope")
execfile(scriptFilename)
try    : print("Global variable startTime = %f" % startTime)
except : print("Global variable  startTime is not defined.")
try    : print("Global variable endTime   = %f" % endTime)
except : print("Global variable  endTime is not defined.")
```