PyPost
Version 4.0.3

A Python Postprocessor for Analysis of Code Results and Experimental Data

User's Manual

February 2024

PyPost:
A Python Postprocessor for Analysis of Code Results and Experimental Data

Licensing

Contents

1. Introduction

PyPost consists of a set of Python modules and stand-alone Java application designed to provide advanced post-processing capability for engineering analysis codes and experimental data results.

PyPost can be used to:

- Query and extract time-dependent plot data from several nuclear engineering analysis codes including: RELAP5/RELAP5-3D, TRACE, MELCOR, etc…
- Read experimental data stored in NRC Databank format.
- Read and write data to and from Microsoft Excel and Open Office spreadsheets.
- Read and write data to and from ASCII files.
- Perform a wide range of mathematical operations on time-dependent vector data.
- Interact directly with AptPlot to generate presentation quality plots in a wide range of formats.

All of these features are available to Python scripts running inside the PyPost script editor or outside using Jython, Python 2.7+, or 3.6+.

2. Getting Started

The PyPost stand-alone application can be used as a graphical interface for editing and executing Python scripts. It can also be used as a Python launcher without the graphical interface. In either case, the PyPost modules are automatically included in the "Python Path" so that they are available to Python scripts.

The syntax for the PyPost application is:

```
> pypost [-g] [-h] [-p <executable>] [script]
```

Argument	Description
-g	Start the PyPost Graphical User Interface (GUI).
-h	Print a help message and exit.
-p [executable]	Run PyPost with the given python executable.
script	The python script to be executed if running in batch mode. If running in GUI mode, this file will be loaded into the application but not executed.

Starting PyPost without specifying a Python script file will display the help message.

PyPost includes Jython which is an implementation of the Python language for the Java platform. The Jython website (http://www.jython.org/) includes a link to an online Jython book. An internet search will reveal several additional books on Jython which provide in depth instruction on developing and running Python modules.

PyPost can also use versions 2.7+ and 3.6+ of the command line CPython interpreter instead of Jython by using the -p command line argument when running PyPost from the command line, or by using the Edit > Set Executable menu item when running PyPost with its GUI.

2.1 Calling PyPost from Python Applications

The PyPost modules can be imported into any Python script running outside of PyPost to give that script access to plot data and other PyPost features. To do so, simply add the following directory to your "Python Path" either by environment variable (PYTHONPATH) or some other means.

```
[SNAP Install]/python
```

Here is an example of setting the PYTHONPATH environment variable to include the SNAP PyPost modules under using a Windows command terminal:

```
set PYTHONPATH=%PYTHONPATH%:C:\SNAP\python
```

And the equivalent using a Linux terminal:

```
export PYTHONPATH=$PYTHONPATH:/home/user/SNAP/python
```

These modules can also be added to your favorite Python IDE (PyCharm, Spyder, etc.) to enable code-completion and pop-up help for the PyPost modules. Like the PyPost application, these modules are compatible with versions 2.7+ and 3.6+ of the CPython interpreter.

3. Working with Analysis Codes and Experimental Data

PyPost can query and extract data from the graphics (aka plot) files created by various engineering analysis codes along with files containing experimental results.

Generated API documentation for the Python modules that support these features can be found in the following directory:

```
[SNAP Install]/pypost/doc/pypost/index.html
```

The list of currently supported analysis codes and experimental data file formats is shown in the table below. The interface instance used to interact with each file format is also listed.

Table 1. Engineering Analysis Code Interfaces

Analysis Code	Interface Instance
COBRA	pypost.codes.COBRA
CONTAIN	pypost.codes.CONTAIN
EXTDATA	pypost.codes.EXTDATA
FAST	pypost.codes.FAST
FRAPCON	pypost.codes.FRAPCON
FRAPTRAN	pypost.codes.FRAPTRAN
GOTHIC	pypost.codes.GOTHIC
MELCOR	pypost.codes.MELCOR
MOOSE	pypost.codes.moose_csv.MooseCsv pypost.codes.moose_exodus.MooseExodus
NRCDB	pypost.codes.NRCDB
PARCS	pypost.codes.PARCS
RELAP5/RELAP5-3D	pypost.codes.RELAP
TRAC-B	pypost.codes.TRACB
TRACE	pypost.codes.TRACE

Refer to the Generated API documentation and the following sections for more information on these interfaces.

3.1 A RELAP5 Example

The following example illustrates the general process of reading plot data with a RELAP5-3D plot file. This process is nearly identical when working with other codes.

The first step is to import the required modules. In this case, PyPost and the RELAP5 code support module.

```
import pypost
from pypost.codes.relap import *
```

The PyPost and RELAP5 interfaces are created and initialized the first time they're imported.

```
RELAP.openPlotFile("sample_data/relap/typpwr.rst", 1)
```

The RELAP interface manages the open files using a "file index". The file index can be specified explicitly when opening a file. If it's not specified then one will be assigned to it and returned from the open method (openPlotFile in this case). If there's already a file with the given index then it will be closed and replaced with the new file.

Some analysis codes can produce plot files in different formats. In addition, some plot files can be demultiplexed to improve performance. The interfaces typically provide different methods to open the different plot file formats. The RELAP5 demultiplexed format, for example.

```
RELAP.openDmxFile("sample_data/relap/relap.dmx", 3)
```

If the file format can be determined by examining the file contents then there is usually a single "open" method. In this case, RELAP5 requires a separate method for its demultiplexed format.

Most plot files include engineering units for each of their data channels. The data written to these files may be in SI or British (Imperial) units. By default, all data read from the plot files are converted, if necessary, to SI units. The PyPost library includes two methods setUseBritishUnits() and setUseSIUnits() that can be used to alter this read behavior. Specifically, the following command will ensure all data is read into British units:

```
# Change default units to British
setUseBritishUnits()
```

Please note that these commands will not affect data that has already been read from the files.

The getData method is used to read data from an open plot file into a Python variable. The method takes two arguments, the file ID, and a list of data channels. For example, the following code reads two data channels from the demultiplexed plot file opened above:

```
# Read in a couple temperatures
httemp1 = RELAP.getData(3, 'httemp-100100101')
httemp2 = RELAP.getData(3, 'httemp-100100201')
```

A shorthand notation for the getData() function is available for each interface. For RELAP5 files, RELAP.getData() can be replaced by simply R5() so the last commands can also be written:

```
# Read in a couple temperatures
httemp1 = R5(3,'httemp-100100101')
httemp2 = R5(3,'httemp-100100201')
```

As shown later, this notation makes it easier to use plot data directly in an equation:

```
# Read in a couple temperatures
deltaT = R5(3,'httemp-100100101')- R5(3,'httemp-100100201')
```

In the earlier case, two heat structure temperatures are read into two variables, httemp1 and httemp2. The getData method will also accept a list of data channel names in which case it will return a list of data channels as in the following example:

```
# Read a list of channels
plotvars = ['httemp-100100101',
            'httemp-100100201',
            'httemp-100100301',
            'httemp-100100401',
            'httemp-100100501',
            'httemp-100100601']
httemps = R5(3, plotvars)
```

Just like a front door, it is always good practice to close a file when you are done using it. To close a single file use the closeFile method:

```
RELAP.closeFile(3)
```

The closeAll() method can be used to close all open files:

```
RELAP.closeAll()
```

3.2 Engineering Units

These functions are available in the ‘pypost’ module. They specify the engineering units that will be used to read data channels from analysis code plot files and experimental results files.

Refer to the Generated API documentation for more information.

`[SNAP Install]/pypost/doc/pypost/index.html`

pypost.setUseSIUnits()

Description:

Sets the plot file engineering units type flag to SI. All subsequent data channels read from plot files will be stored in SI units. (Default)

pypost.setUseBritishUnits()

Description:

Sets the plot file engineering units type flag to British (Imperial). All subsequent data channels read from plot files will be stored in British units.

pypost.isUsingSIUnits()

Description:

Returns True if the plot file engineering units type flag is currently set to SI, otherwise returns False.

Example:

```
# Toggle the engineering units type flag
print("EU Flag="+repr(isUsingSIUnits()))
setUseBritishUnits()
print("EU Flag="+repr(isUsingSIUnits()))
setUseSIUnits()
print("EU Flag="+repr(isUsingSIUnits()))
```

Output:

```
EU Flag=True
EU Flag=False
EU Flag=True
```

3.3 PlotFileIntf Interface Class (Abstract)

All analysis code plot file interfaces extend this abstract class. Each of the interface methods are described below. Examples below utilize the RELAP5/RELAP5-3D implementation of this interface.

Refer to the Generated API documentation for more information.

```
[SNAP Install]/pypost/doc/pypost/index.html
```

Interface Methods:

closeAll()

Close all of the open plot files.

closeFile(fileIndex)

Arguments:

fileIndex [int] The file index of the open file.

Description:

Closes a specific file.

getData(fileIndex, channelNames)

Arguments:

fileIndex [int] The file index of the open file.

channelNames [String | String[]] Either a string containing the channel name or String array containing a list of channels.

Description:

Returns either a single channel vector if a single channel is requested or a list containing all requested channel vectors. Each code interface includes a global function to provide shorthand notation for this routine. The last two methods in the following example produce identical results.

Example:

```
# Read a single channel
httemp1 = RELAP.getData(0, 'httemp-100100101')
#
# Read a list of channels
plotvars = ['httemp-100100101',
            'httemp-100100201',
            'httemp-100100301',
            'httemp-100100401']
```

```
Httemps1 = RELAP.getData(0, plotvars)
httemps2 = R5(0, plotvars)
```

getFileList()

Description:

Outputs a list of the currently open files.

Example:

```
# List the open files.
RELAP.openDmxFile("./sample_data/relap/relap.dmx")
RELAP.openRstpltFile("./sample_data/relap/typpwr.rst")
RELAP.getFileList()
```

Output:

```
OPEN RELAP5 FILES:

./sample_data/relap/typpwr.rst        RSTPLT
./sample_data/relap/relap.dmx         DEMUX
```

getFileInfo()

Description:

Returns an array of **FileData** objects containing information on all open files.

Example:

```
# Print file information for the first file.
RELAP.openDmxFile("./sample_data/relap/relap.dmx")
RELAP.openRstpltFile("./sample_data/relap/typpwr.rst")
info = RELAP.getFileInfo()
print("File Info:")
print(info[0])
```

Output:

```
File Info:
File ID [0]
File Type [RELAP5 DEMUX]
Filename [./sample_data/relap/relap.dmx]
Number of Channels [21429]
Number of Time Steps [278]
```

getNumPlotVars (fileIndex)

Arguments:

fileIndex [int] The file index of the open file.

Description:

Returns the number of plot variables available in the file.

Example:

```
# Print the number of plot variables in a file.
RELAP.openDmxFile("./sample_data/relap/relap.dmx")
numvars=RELAP.getNumPlotVars(0)
print("numvars= "+repr(numvars))
```

Output:

```
numvars= 21429
```

getNumTimeSteps (fileIndex)

Arguments:

fileIndex [int] The file index of the open file.

Description:

Returns the number of time steps available in the file.

Example:

```
# Print the number of time steps in a file.
RELAP = RELAP()
RELAP.openDmxFile("./sample_data/relap/relap.dmx")
numSlices=RELAP.getNumTimeSteps(0)
print("numSlices= "+repr(numSlices))
```

Output:

```
numSlices= 278
```

getPlotVarType (fileIndex, variableName)

Arguments:

fileIndex [int] The file index of the open file.

variableName [string] The name of the requested variable.

Description:

Returns the plot variable's engineering unit type.

Example:

```
# Print the channel units type.
RELAP.openDmxFile("./sample_data/relap/relap.dmx")
vType=RELAP.getPlotVarType (0, 'httemp-100100101')
print("vType= "+vType)
```

Output:

```
        vType= Mesh Point Temperature
```

getPlotVarUnits (fileIndex, variableName)

Arguments:

fileIndex [int] The file index of the open file.

variableName [string] The name of the requested variable.

Description:

Returns the plot variable's engineering units.

Example:

```
# Print the channel units type.
RELAP.openDmxFile("./sample_data/relap/relap.dmx")
eunits=RELAP.getPlotVarUnits (0, 'httemp-100100101')
print("eunits= "+eunits)
```

Output:

```
        eunits= K
```

getPlotVars (fileIndex)

Arguments:

fileIndex [int] The file index of the open file.

Description:

Returns a list of all plot variables for the specified file.

Example:

```
# List the first 10 channels.
RELAP.openDmxFile("./sample_data/relap/relap.dmx")
chans=RELAP.getPlotVars (0)
for i in range(0, 10):
    print chans[i]
```

Output:

```
        acqtank-702
        acqtank-703
        acqtank-704
        acqtank-705
        acrhon-702
        acrhon-703
        acrhon-704
        acrhon-705
        acttank-702
        acttank-703
```

hasPlotVar(fileIndex, variableName)

Arguments:

fileIndex [int] The file index of the open file.

variableName [string] The name of the requested variable.

Description:

Returns true if the specified file contains the specified variable, false otherwise.

Example:

```
# Test hasPlotVar function
RELAP.openDmxFile("./sample_data/relap/relap.dmx")
hasVar=RELAP.hasPlotVar(0,'httemp-100100101')
print("hasVar= "+repr(hasVar))
hasVar=RELAP.hasPlotVar(0,'httemp-999100101')
print("hasVar= "+repr(hasVar))
```

Output:

```
hasVar= True
hasVar= False
```

supportsAltIndependentUnits()

Description:

Returns true if the plot file interface allows for independent units other than time. Unit types other than time may be changed via the setIndependentUnits function, or retrieved via the getIndependentUnits function.

setIndependentUnits (fileIndex, unitCode)

Arguments:

fileIndex [int] The file index of the open file.

unitCode [int] the enumerated integer value of the unit type

Description:

Sets the independent units used for the file at the given file index. Note that this does nothing unless supportsAltIndependentUnits() returns true. Plot file interfaces supporting this function have a more specific description (included the allowed unitCodes) in their own section.

Example:

```
# Sets the independent units to burnup.
FRAPCON.setIndependentUnits(0, FRAPCON.BURNUP_GWD_MTU)
```

getIndependentUnits (fileIndex)

Arguments:

fileIndex [int] The file index of the open file.

Description:

Returns the enumerated integer value for the independent units used by the file at the given file index. This function returns -1 unless the supportsAltIndependentUnits() function returns true.

Example:

```
# Print the unit type used for the first file.
unitType = FRAPCON.getIndependentUnits(0)
print(unitType)
```

demux (muxPath, demuxPath, String additionalArguments)

Arguments:

muxPath [string] The path of the file to demultiplex, anchored by PyPost's current working directory.

demuxPath [string] The desired path of the demultiplexed file, anchored by PyPost's current working directory.

additionalArguments [string](optional) A string containing the additional arguments to pass to the demuxing utility.

Description:

Demultiplexes the mux file with the given path, creating a demux file with the desired path in the process. Additional arguments can be found in the documentation for the respective plot file.

Example:

```
# Demux the trace.xtv file, creating a trace.dmx file
TRACE.demux("trace.xtv", "trace.dmx")
```

3.4 COBRA Plot File Interface Class

This interface provides access to COBRA plot files. This interface inherits all methods from PlotFileIntf. The additional global and interface methods are described below. The interface can manage any number of open plot files, organized using an integer file ID. Refer to the Generated API documentation for more information.

```
[SNAP Install]/pypost/doc/pypost/codes/cobra.html
```

The first step to reading and working with these files is to import the required modules. In this case, PyPost and the COBRA code support module.

```
import pypost
from pypost.codes.cobra import *
```

Global Instance:

COBRA Singleton instance of the COBRA Plot File interface class.

Global Data Access Function:

CO(fileIndex, channelNames)

Arguments:

fileIndex [int] The file index of the open file.

channelNames [String | String[]] Either a string containing the channel name or String array containing a list of channels.

Description:

Returns either a single channel vector if a single channel is requested or a list containing all requested channel vectors.

Interface Methods:

openPlotFile(fileName, fileIndex)

Arguments:

fileName [string] The full path to the file to be opened.

fileIndex [int] (Optional) An ID used to identify this file for subsequent calls. The file index can be specified explicitly when opening a file. If left unspecified, the file index will be initialized to the next available number. If the file index is specified explicitly and a file is currently opened at that index, the open file will be replaced with the new file.

Description:

Opens a COBRA plot file. This file can be demultiplexed.

Example:

```
# Open two COBRA plot files, Multiplexed and Demultiplexed.
setUseBritishUnits()
COBRA.openPlotFile("./sample_data/cobra/cobra.grf")
COBRA.openPlotFile("./sample_data/cobra/cobra.dmx")
COBRA.getFileList()
pDome = CO(0,'p-001001')
```

3.5 CONTAIN Interface Class

This interface provides access CONTAIN plot files. This interface inherits all methods from PlotFileIntf. Additional global and interface methods are described below. The interface can manage any number of open plot files, organized using an integer file ID. Refer to the Generated API documentation for more information.

```
[SNAP Install]/pypost/doc/pypost/codes/contain.html
```

The first step to reading and working with these files is to import the required modules. In this case, PyPost and the CONTAIN code support module.

```
import pypost
from pypost.codes.contain import *
```

Global Instance:

CONTAIN Singleton instance of CONTAIN Plot File interface class.

Global Data Access Function:

CN(fileIndex, channelNames)

Arguments:

fileIndex [int] The file index of the open file.

channelNames [String | String[]] Either a string containing the channel name or String array containing a list of channels.

Description:

Returns either a single channel vector if a single channel is requested or a list containing all requested channel vectors.

Interface Methods:

openPlotFile(fileName, fileIndex)

Arguments:

fileName [string] The full path to the file to be opened.

fileIndex [int] (Optional) An ID used to identify this file for subsequent calls. The file index can be specified explicitly when opening a file. If left unspecified, the file index will be initialized to the next available number. If the file index is specified explicitly and a file is currently opened at that index, the open file will be replaced with the new file.

Description:

Opens a CONTAIN plot file.

Example:

```
# Open a CONTAIN plot file
setUseBritishUnits()
CONTAIN.openPlotFile("./sample_data/contain/V44.pibplot")
coolantMass = CN(0,'F103_C003_COOLMASS')
```

3.6 EXTDATA Interface Class

This interface provides access to EXTDATA plot files. This interface inherits all methods from PlotFileIntf. Additional global and interface methods are described below. The interface can manage any number of open plot files, organized using an integer file ID. Refer to the Generated API documentation for more information.

```
[SNAP Install]/pypost/doc/pypost/codes/contain.html
```

The first step to reading and working with these files is to import the required modules. In this case, PyPost and the EXTDATA support module.

```
import pypost
from pypost.codes.extdata import *
```

Global Instance:

EXTDATA Singleton instance of the EXTDATA Experimental Data File interface class.

Global Data Access Function:

EX(fileIndex, channelNames)

Arguments:

fileIndex [int] The file index of the open file.

channelNames [String | String[]] Either a string containing the channel name or String array containing a list of channels.

Description:

Returns either a single channel vector if a single channel is requested or a list containing all requested channel vectors.

Interface Methods:

openPlotFile(fileName, fileIndex)

Arguments:

fileName [string] The full path to the file to be opened.

fileIndex [int] (Optional) An ID used to identify this file for subsequent calls. The file index can be specified explicitly when opening a file. If left unspecified, the file index will be initialized to the next available number. If the file index is specified explicitly and a file is currently opened at that index, the open file will be replaced with the new file.

Description:

Opens an EXTDATA plot file.

Example:

```
# Open an EXTDATA plot file
EXTDATA.openPlotFile("./sample_data/extdata/plot-si")
```

3.7 FAST Interface Class

This interface provides access to FAST plot files. This interface inherits all methods from PlotFileIntf. Additional global and interface methods are described below. The interface can manage any number of open plot files, organized using an integer file ID.

Note that this interface supports Rod Average Burnup (both GWd/MTU and MWd/MTU) as an independent unit in addition to time. The independent units may be changed via the setIndependentUnits function described in this section.

Refer to the Generated API documentation for more information.

```
[SNAP Install]/pypost/doc/pypost/codes/fast.html
```

The first step to reading and working with these files is to import the required modules. In this case, PyPost and the FAST code support module.

```
import pypost
from pypost.codes.fast import *
```

Global Instance:

FAST Singleton instance of FAST Plot File interface class.

Global Data Access Function:

FA (fileIndex, channelNames)

Arguments:

fileIndex [int] The file index of the open file.

channelNames [String | String[]] Either a string containing the channel name or String array containing a list of channels.

Description:

Returns either a single channel vector if a single channel is requested or a list containing all requested channel vectors.

Interface Methods:

openPlotFile(fileName, fileIndex)

Arguments:

fileName [string] The full path to the file to be opened.

fileIndex [int] (Optional) An ID used to identify this file for subsequent calls. The file index can be specified explicitly when opening a file. If left unspecified, the file index will be initialized to the next available number. If the file index is specified explicitly and a file is currently opened at that index, the open file will be replaced with the new file.

Description:

Opens FAST plot file.

Example:

```
# Open a FAST plot file
FAST.openPlotFile("./sample_data/FAST/fast.pib", 3)
data = FC(3, 'temp-01R03')
```

getAxialData(fileIndex, channelNames, time, offset)

Arguments:

fileIndex [int] The file index of the open file.

channelNames [String | String[]] Either a string containing the channel name or String array containing a list of channels.

time [double] (days, burnup) The analysis time or burnup in which the axial data will be extracted.

offset [double] (m / ft, optional, Default=0.0) The axial off set.

Description:

Returns a set of ChannelVectors containing the axial data for the indicated data channels at a specified time or burnup.

Example:

```
# Create an axial plot using the burnup data channel
data = FAST.getAxialData(1,'burnup-A01',1000)
APTPLOT.plotAxialChannels(data)
```

getRadialData(fileIndex, channelNames, occurringAt, offset)

Arguments:

fileIndex [int] The file index of the open file.

channelNames [String | String[]] Either a string containing the channel name or String array containing a list of channels.

time [double] (days, burnup) The analysis time or burnup in which the axial data will be extracted.

offset [double] (m / ft, optional, Default=0.0) The axial off set.

Description:

Returns a set of ChannelVectors containing the radial data for the indicated data channels at a specified time or burnup.

Example:

```
# Create a radial plot using the temp data channel
data = FAST.getRadialData(1,'temp-A12R01',14000)
APTPLOT.plotRadialChannels(data)

# Create a radial plot using the fggrain data channel
data = FAST.getRadialData(1,'fggrain-A01R01',1000)
APTPLOT.plotRadialChannels(data)
```

setIndependentUnits (fileIndex, unitCode)

Arguments:

fileIndex [int] The file index of the open file.

unitCode [int] One of the following values:

- FAST.TIME (specifies time in seconds)
- FAST.BURNUP_GWD_MTU (specifies rod average burnup as GWd/MTU)
- FAST.BURNUP_MWD_MTU (specifies rod average burnup as MWd/MTU)

Description:

Sets the independent units used for the file at the given file index. The supported units are time or rod average burnup as GWd/MTU or MWd/MTU. Note that the occurringAt parameter of the getAxialData and getRadialData functions are specified in the units set by this function.

Example:

```
# Sets the independent units to burnup.
FAST.setIndependentUnits(0, FAST.BURNUP_GWD_MTU)
```

3.8 FRAPCON Interface Class

This interface provides access to FRAPCON plot files. This interface inherits all methods from PlotFileIntf. Additional global and interface methods are described below. The interface can manage any number of open plot files, organized using an integer file ID.

Note that this interface supports Rod Average Burnup (both GWd/MTU and MWd/MTU) as an independent unit in addition to time. The independent units may be changed via the setIndependentUnits function described in this section.

Refer to the Generated API documentation for more information.

```
[SNAP Install]/pypost/doc/pypost/codes/frapcon.html
```

The first step to reading and working with these files is to import the required modules. In this case, PyPost and the FRAPCON code support module.

```
import pypost
from pypost.codes.frapcon import *
```

Global Instance:

FRAPCON Singleton instance of FRAPCON Plot File interface class.

Global Data Access Function:

FC(fileIndex, channelNames)

Arguments:

fileIndex [int] The file index of the open file.

channelNames [String | String[]] Either a string containing the channel name or String array containing a list of channels.

Description:

Returns either a single channel vector if a single channel is requested or a list containing all requested channel vectors.

Interface Methods:

openPlotFile(fileName, fileIndex)

Arguments:

fileName [string] The full path to the file to be opened.

fileIndex [int] (Optional) An ID used to identify this file for subsequent calls. The file index can be specified explicitly when opening a file. If left unspecified, the file index will be initialized to the next available number. If the file index is specified explicitly and a file is currently opened at that index, the open file will be replaced with the new file.

Description:

Opens FRAPCON plot file.

Example:

```
# Open a FRAPCON plot file
FRAPCON.openPlotFile("./sample_data/frapcon/BOL_Therm.pib", 3)
data = FC(3, 'temp-01R03')
```

getAxialData(fileIndex, channelNames, time, offset)

Arguments:

fileIndex [int] The file index of the open file.

channelNames [String | String[]] Either a string containing the channel name or String array containing a list of channels.

time [double] (days, burnup) The analysis time or burnup in which the axial data will be extracted.

offset [double] (m / ft, optional, Default=0.0) The axial off set.

Description:

Returns a set of ChannelVectors containing the axial data for the indicated data channels at a specified time or burnup.

Example:

```
# Create an axial plot using the burnup data channel
data = FRAPCON.getAxialData(1,'burnup-A01',1000)
APTPLOT.plotAxialChannels(data)
```

getRadialData(fileIndex, channelNames, occurringAt, offset)

Arguments:

fileIndex [int] The file index of the open file.

channelNames [String | String[]] Either a string containing the channel name or String array containing a list of channels.

time [double] (days, burnup) The analysis time or burnup in which the axial data will be extracted.

offset [double] (m / ft, optional, Default=0.0) The axial off set.

Description:

Returns a set of ChannelVectors containing the radial data for the indicated data channels at a specified time or burnup.

Example:

```
# Create a radial plot using the temp data channel
data = FRAPCON.getRadialData(1,'temp-A12R01',14000)
APTPLOT.plotRadialChannels(data)

# Create a radial plot using the fggrain data channel
data = FRAPCON.getRadialData(1,'fggrain-A01R01',1000)
APTPLOT.plotRadialChannels(data)
```

setIndependentUnits (fileIndex, unitCode)

Arguments:

fileIndex [int] The file index of the open file.

unitCode [int] One of the following values:

- FRAPCON.TIME (specifies time in seconds)
- FRAPCON.BURNUP_GWD_MTU (specifies rod average burnup as GWd/MTU)
- FRAPCON.BURNUP_MWD_MTU (specifies rod average burnup as MWd/MTU)

Description:

Sets the independent units used for the file at the given file index. The supported units are time or rod average burnup as GWd/MTU or MWd/MTU. Note that the occurringAt parameter of the getAxialData and getRadialData functions are specified in the units set by this function.

Example:

```
# Sets the independent units to burnup.
FRAPCON.setIndependentUnits(0, FRAPCON.BURNUP_GWD_MTU)
```

3.9 FRAPTRAN Interface Class

This interface provides access to FRAPTRAN plot files. This interface inherits all methods from PlotFileIntf. Additional global and interface methods are described below. The interface can manage any number of open plot files, organized using an integer file ID.

Refer to the Generated API documentation for more information.

```
[SNAP Install]/pypost/doc/pypost/codes/fraptran.html
```

The first step to reading and working with these files is to import the required modules. In this case, PyPost and the FRAPTRAN code support module.

```
import pypost
from pypost.codes.fraptran import *
```

Global Instance:

FRAPTRAN Singleton instance of FRAPTRAN Plot File interface class.

Global Data Access Function:

FT(fileIndex, channelNames)

Arguments:

fileIndex [int] The file index of the open file.

channelNames [String | String[]] Either a string containing the channel name or String array containing a list of channels.

Description:

Returns either a single channel vector if a single channel is requested or a list containing all requested channel vectors.

Interface Methods:

openPlotFile(fileName, fileIndex)

Arguments:

fileName [string] The full path to the file to be opened.

fileIndex [int] (Optional) An ID used to identify this file for subsequent calls. The file index can be specified explicitly when opening a file. If left unspecified, the file index will be initialized to the next available number. If the file index is specified explicitly and a file is currently opened at that index, the open file will be replaced with the new file.

Description:

Opens a FRAPTRAN plot file.

Example:

```
# Open a FRAPTRAN plot file
FRAPTRAN.openPlotFile("./sample_data/fraptran/fraptran.pib")
```

3.10 GOTHIC Interface Class

This interface provides access GOTHIC plot files. This interface inherits all methods from PlotFileIntf. Additional global and interface methods are described below. The interface can manage any number of open plot files, organized using an integer file ID.

Refer to the Generated API documentation for more information.

```
[SNAP Install]/pypost/doc/pypost/codes/gothic.html
```

The first step to reading and working with these files is to import the required modules. In this case, PyPost and the GOTHIC code support module.

```
import pypost
from pypost.codes.gothic import *
```

Global Instance:

GOTHIC Singleton instance of GOTHIC Plot File interface class.

Global Data Access Function:

GO(fileIndex, channelNames)

Arguments:

fileIndex [int] The file index of the open file.

channelNames [String | String[]] Either a string containing the channel name or String array containing a list of channels.

Description:

Returns either a single channel vector if a single channel is requested or a list containing all requested channel vectors.

Interface Methods:

openPlotFile(fileName, fileIndex)

Arguments:

fileName [string] The full path to the file to be opened.

fileIndex [int] (Optional) An ID used to identify this file for subsequent calls. The file index can be specified explicitly when opening a file. If left unspecified, the file index will be initialized to the next available number. If the file index is specified explicitly and a file is currently opened at that index, the open file will be replaced with the new file.

Description:

Opens a GOTHIC plot file (PIB format).

Example:

```
# Open GOTHIC plot file and demultiplexed plot file
GOTHIC.openPlotFile("./sample_data/gothic/gothic.SGR")
GOTHIC.openPlotFile("./sample_data/gothic/gothic.dmx")
```

3.11 MELCOR Interface Class

This interface provides access to MELCOR plot files. This interface inherits all methods from PlotFileIntf. Additional global and interface methods are described below. The interface can manage any number of open plot files, organized using an integer file ID.

Refer to the Generated API documentation for more information.

```
[SNAP Install]/pypost/doc/pypost/codes/melcor.html
```

The first step to reading and working with these files is to import the required modules. In this case, PyPost and the MELCOR code support module.

```
import pypost
from pypost.codes.melcor import *
```

<u>Global Instance:</u>

MELCOR Singleton instance of MELCOR Plot File interface class.

<u>Global Data Access Function:</u>

MC(fileIndex, channelNames)

Arguments:

fileIndex [int] The file index of the open file.

channelNames [String | String[]] Either a string containing the channel name or String array containing a list of channels.

Description:

Returns either a single channel vector if a single channel is requested or a list containing all requested channel vectors.

<u>Interface Methods:</u>

openPlotFile(fileName, fileIndex)

Arguments:

fileName [string] The full path to the file to be opened.

fileIndex [int] (Optional) An ID used to identify this file for subsequent calls. The file index can be specified explicitly when opening a file. If left unspecified, the file index will be initialized to the next available number. If the file index is specified explicitly and a file is currently opened at that index, the open file will be replaced with the new file.

Description:

Opens a MELCOR plot file (PTF format).

Example:

```
# Open a MELCOR plot file
MELCOR.openPlotFile("./sample_data/melcor/Fukushima.ptf")
```

openDmxFile(fileName, fileIndex)

Arguments:

fileName [string] The full path to the file to be opened.

fileIndex [int] (Optional) An ID used to identify this file for subsequent calls. The file index can be specified explicitly when opening a file. If left unspecified, the file index will be initialized to the next available number. If the file index is specified explicitly and a file is currently opened at that index, the open file will be replaced with the new file.

Description:

Opens a demultiplexed MELCOR plot file.

Example:

```
# Open demultiplexed MELCOR plot file
MELCOR.openDmxFile("./sample_data/melcor/Fukushima.dmx")
```

demux (muxPath, demuxPath, String additionalArguments)

Additional arguments:

-cq : Perform a quick run length compression.

3.12 MooseCsv Interface Class

This interface provides access to MOOSE CSV plot files. This interface inherits all methods from PlotFileIntf. Additional global methods are described below. The interface can manage any number of open plot files, organized using an integer file ID.

Refer to the Generated API documentation for more information.

```
[SNAP Install]/pypost/doc/pypost/codes/moose_csv.html
```

The first step to reading and working with these files is to import the required modules. In this case, PyPost and the MooseCsv support module.

```
import pypost
from pypost.codes.moose_csv import *
```

<u>Global Instance:</u>

MooseCsv Singleton instance of MOOSE CSV data file interface class.

<u>Global Data Access Function:</u>

MO_C(file_number, channels)

Arguments:

file_number [int] The file index of the open file.

channelNames [String | String[]] Either a string containing the channel name or String array containing a list of channels.

Description:

Reads the requested channel(s) from the file as channel vectors. This method is a shortcut to 'MooseCsvIntf.getData'.

3.13 Moose_Exodus Interface Class

This interface provides access to MOOSE Exodus plot files. This interface inherits all methods from PlotFileIntf. Additional global methods are described below. The interface can manage any number of open plot files, organized using an integer file ID.

Refer to the Generated API documentation for more information.

```
[SNAP Install]/pypost/doc/pypost/codes/moose_exodus.html
```

The first step to reading and working with these files is to import the required modules. In this case, PyPost and the MooseExodus support module.

```
import pypost
```

```
from pypost.codes.moose_exodus import *
```

Global Instance:

MooseExodus Singleton instance of MOOSE CSV data file interface class.

Global Data Access Function:

MO_E(file_number, channels)

Arguments:

file_number [int] The file index of the open file.

channelNames [String | String[]] Either a string containing the channel name or String array containing a list of channels.

Description:

Reads the requested channel(s) from the file as channel vectors. This method is a shortcut to 'MooseExodusIntf.getData'.

3.14 NRCDB Interface Class

This interface provides access to NRC Databank experimental data files. This interface inherits all methods from PlotFileIntf. Additional global and interface methods are described below. The interface can manage any number of open plot files, organized using an integer file ID.

Refer to the Generated API documentation for more information.

```
[SNAP Install]/pypost/doc/pypost/codes/nrcdb.html
```

The first step to reading and working with these files is to import the required modules. In this case, PyPost and the NRCDB support module.

```
import pypost
from pypost.codes.nrcdb import *
```

Global Instance:

NRCDB Singleton instance of NRC Databank experimental data file interface class.

Global Data Access Function:

DB(fileIndex, channelNames)

Arguments:

fileIndex [int] The file index of the open file.

channelNames [String | String[]] Either a string containing the channel name or String array containing a list of channels.

Description:

Returns either a single channel vector if a single channel is requested or a list containing all requested channel vectors.

Interface Methods:

openPlotFile(fileName, fileIndex)

Arguments:

fileName [string] The full path to the file to be opened.

fileIndex [int] (Optional) An ID used to identify this file for subsequent calls. The file index can be specified explicitly when opening a file. If left unspecified, the file index will be initialized to the next available number. If the file index is specified explicitly and a file is currently opened at that index, the open file will be replaced with the new file.

Description:

Opens an NRC Databank plot file.

Example:

```
# Open an NRC Databank plot file
NRCDB.openPlotFile("./sample_data/nrcdb/1_Pump_Trip.bin")
```

getNumTimeSteps (fileIndex, variableName)

Arguments:

fileIndex [int] The file index of the open file.

variableName [string] The name of the requested variable.

Description:

Returns the number of time steps available for the specified data channel. Different channels in Databank files may have a differing number of timesteps.

Example:

```
# Print the number of time steps for different channels.
NRCDB.openPlotFile("./sample_data/nrcdb/1_Pump_Trip.bin")
print("-----Number of time steps for COREFLO")
print(NRCDB.getNumTimeSteps(0, COREFLO))
print("-----Number of time steps for COREWTRLVLCNG")
print(NRCDB.getNumTimeSteps(0, 'COREWTRLVLCNG'))
```

Output:

```
-----Number of time steps for COREFLO
20
-----Number of time steps for COREWTRLVLCNG
13
```

3.15 PARCS Interface Class

This interface provides access to PARCS plot files. This interface inherits all methods from PlotFileIntf. Additional global and interface methods are described below. The interface can manage any number of open plot files, organized using an integer file ID.

Refer to the Generated API documentation for more information.

```
[SNAP Install]/pypost/doc/pypost/codes/parcs.html
```

The first step to reading and working with these files is to import the required modules. In this case, PyPost and the PARCS code support module.

```
import pypost
from pypost.codes.parcs import *
```

Global Instance:

PARCS Singleton instance of PARCS Plot File interface class.

Global Data Access Function:

PC(fileIndex, channelNames)

Arguments:

fileIndex [int] The file index of the open file.

channelNames [String | String[]] Either a string containing the channel name or String array containing a list of channels.

Description:

Returns either a single channel vector if a single channel is requested or a list containing all requested channel vectors.

Interface Methods:

openPlotFile(fileName, fileIndex)

Arguments:

fileName [string] The full path to the file to be opened.

fileIndex [int] (Optional) An ID used to identify this file for subsequent calls. The file index can be specified explicitly when opening a file. If left unspecified, the file index will be initialized to the next available number. If the file index is specified explicitly and a file is currently opened at that index, the open file will be replaced with the new file.

Description:

Opens a PARCS plot file (BPF format).

Example:

```
# Open a PARCS plot file
PARCS.openPlotFile("./sample_data/parcs/neacrp.bpf")
```

3.16 RELAP5/RELAP5-3D Interface Class

This interface provides access to RELAP5 and RELAP5-3D plot files. This interface inherits all methods from PlotFileIntf. Additional global and interface methods are described below. The interface can manage any number of open plot files, organized using an integer file ID.

Refer to the Generated API documentation for more information.

```
[SNAP Install]/pypost/doc/pypost/codes/relap.html
```

The first step to reading and working with these files is to import the required modules. In this case, PyPost and the RELAP5 code support module.

```
import pypost
from pypost.codes.relap import *
```

<u>Global Instance:</u>

RELAP Singleton instance of RELAP5/RELAP5-3D Plot File interface class.

<u>Global Data Access Function:</u>

R5(fileIndex, channelNames)

Arguments:

fileIndex [int] The file index of the open file.

channelNames [String | String[]] Either a string containing the channel name or String array containing a list of channels.

Description:

Returns either a single channel vector if a single channel is requested or a list containing all requested channel vectors.

<u>Interface Methods:</u>

openPlotFile(fileName, fileIndex)

Arguments:

fileName [string] The full path to the file to be opened.

fileIndex [int] (Optional) An ID used to identify this file for subsequent calls. The file index can be specified explicitly when opening a file. If left unspecified, the file index will be initialized to the next available number. If the file index is specified explicitly and a file is currently opened at that index, the open file will be replaced with the new file.

Description:

Opens a legacy RELAP5/ RELAP5-3D rstplt file or a RELAP5-3D plot file (PIB format).

Example:

```
# Open a RELAP5 plot file
RELAP.openPlotFile("./sample_data/relap/relap.plt")
RELAP.openPlotFile("./sample_data/relap/relap.rst")
```

openDmxFile(fileName, fileIndex)

Arguments:

fileName [string] The full path to the file to be opened.

fileIndex [int] (Optional) An ID used to identify this file for subsequent calls. The file index can be specified explicitly when opening a file. If left unspecified, the file index will be initialized to the next available number. If the file index is specified explicitly and a file is currently opened at that index, the open file will be replaced with the new file.

Description:

Opens a demultiplexed legacy RELAP5/ RELAP5-3D rstplt file.

Example:

```
# Open RELAP5 rstplt file
RELAP.openPlotFile("./sample_data/relap/relap.rst")
```

demux (muxPath, demuxPath, String additionalArguments)

Additional arguments:

-cq : Perform a quick run length compression.

3.17 RETRAN-3D Interface Class

This interface provides access RETRAN-3D ASCII, binary, and demultiplexed, plot files. This interface inherits all methods from PlotFileIntf. Additional global and interface methods are described below. The interface can manage any number of open plot files, organized using an integer file ID.

Refer to the Generated API documentation for more information.

```
[SNAP Install]/pypost/doc/pypost/codes/retran3d.html
```

The first step to reading and working with these files is to import the required modules. In this case, PyPost and the RETRAN-3D code support module.

```
import pypost
from pypost.codes.retran3d import *
```

Global Instance:

RETRAN3D Singleton instance of RETRAN3D Plot File interface class.

Global Data Access Function:

RN(fileIndex, channelNames)

Arguments:

fileIndex [int] The file index of the open file.

channelNames [String | String[]] Either a string containing the channel name or String array containing a list of channels.

Description:

Returns either a single channel vector, if a single channel is requested, or a list containing all requested channel vectors.

Interface Methods:

openPlotFile(fileName, fileIndex)

Arguments:

fileName [string] The full path to the file to be opened.

fileIndex [int] (Optional) An ID used to identify this file for subsequent calls. The file index can be specified explicitly when opening a file. If left unspecified, the file index will be initialized to the next available number. If the file index is specified explicitly and a file is currently opened at that index, the open file will be replaced with the new file.

Description:

Opens a RETRAN3D ASCII, binary, or demultiplexed, plot file.

Example:

```
# Open each supported RETRAN3D plot file format
RETRAN3D.openPlotFile("./sample_data/retran3d/retran3d.ascii")
RETRAN3D.openPlotFile("./sample_data/retran3d/retran3d.dmx")
RETRAN3D.openPlotFile("./sample_data/retran3d/retran3d.plt")
```

3.18 TRACE Interface Class

This interface provides access to TRACE plot files. This interface inherits all methods from PlotFileIntf. Additional global and interface methods are described below. The interface can manage any number of open plot files, organized using an integer file ID.

Refer to the Generated API documentation for more information.

```
[SNAP Install]/pypost/doc/pypost/codes/trace.html
```

The first step to reading and working with these files is to import the required modules. In this case, PyPost and the TRACE code support module.

```
import pypost
from pypost.codes.trace import *
```

Global Instance:

TRACE Singleton instance of TRACE Plot File interface class.

Global Data Access Function:

TR(fileIndex, channelNames)

Arguments:

fileIndex [int] The file index of the open file.

channelNames [String | String[]] Either a string containing the channel name or String array containing a list of channels.

Description:

Returns either a single channel vector if a single channel is requested or a list containing all requested channel vectors.

Interface Methods:

openPlotFile(fileName, fileIndex)

Arguments:

fileName [string] The full path to the file to be opened.

fileIndex [int] (Optional) An ID used to identify this file for subsequent calls. The file index can be specified explicitly when opening a file. If left unspecified, the file index will be initialized to the next available number. If the file index is specified explicitly and a file is currently opened at that index, the open file will be replaced with the new file.

Description:

Opens a TRACE XTV plot file. The file may or may not be demultiplexed.

Example:

```
# Open a TRACE XTV plot file and a demultiplexed
# XTV plot file
TRACE.openPlotFile("./sample_data/trace/trace.xtv")
TRACE.openPlotFile("./sample_data/trace/trace_demux.xtv")
```

getAxialData(fileIndex, channelNames, time, offset)

Arguments:

fileIndex [int] The file index of the open file.

channelNames [String | String[]] Either a string containing the channel name or String array containing a list of channels. Each channel name must represent the data channel at the first axial node (i.e. vol-7A01 rather than vol-7A03).

time [double] (s) The analysis time that the axial data will be extracted.

offset [double] (m / ft, optional, Default=0.0) The axial off set.

Description:

Returns a set of ChannelVectors containing the axial data for the indicated data channels at a specified transient time. Each channel vector's independent values are the elevation data, while the dependent values are the requested channel data (i.e. (lencc-7A01, vol-7A01), (lencc-7A02, vol-7A02) and so on).

Example:

```
# Retrieve vol-7A01, vol-7A02 and the
# corresponding elevation data as a channel vector
# then create a plot in AptPlot.
data = TRACE.getAxialData(2,'vol-7A01',20.0)
```

```
APTPLOT.plotAxialChannels(data)
```

getValueAt(fileIndex, channelName, time)

Arguments:

fileIndex [int] The file index of the open file.

channelName [String] The string containing the channel name

time [double] (s) The analysis time in which the data will be extracted.

Description:

Returns a double containing the plot data for the indicated data channel at a specified transient time.

Example:

```
# Read liquid temperature at 100 seconds.
TRACE.getDataAt(1, ‘tln-21A01’, 100.0)
```

getValuesAt(fileIndex, channelNames, time)

Arguments:

fileIndex [int] The file index of the open file.

channelNames [String[]] The strings containing the channel names

time [double] (s) The analysis time in which the data will be extracted.

Description:

Returns a list of doubles containing the plot data for the indicated data channels at a specified transient time.

Example:

```
# Read liquid temperature at 100 seconds.
TRACE.getDataAt(1, [‘tln-21A01’, ‘tln-21A20’], 100.0)
```

demux (muxPath, demuxPath, String additionalArguments)

Additional arguments:

-debug : Flag that turns off all debug printing.

3.19 TRACB Interface Class

This interface provides access to TRAC-B plot files. This interface inherits all methods from PlotFileIntf. Additional global and interface methods are described below. The interface can manage any number of open plot files, organized using an integer file ID.

Refer to the Generated API documentation for more information.

```
[SNAP Install]/pypost/doc/pypost/codes/trace.html
```

The first step to reading and working with these files is to import the required modules. In this case, PyPost and the TRACE code support module.

```
import pypost
from pypost.codes.trace import *
```

Global Instance:

TRACB Singleton instance of TRAC-B Plot File interface class.

Global Data Access Function:

TB(fileIndex, channelNames)

Arguments:

fileIndex [int] The file index of the open file.

channelNames [String | String[]] Either a string containing the channel name or String array containing a list of channels.

Description:

Returns either a single channel vector if a single channel is requested or a list containing all requested channel vectors.

Interface Methods:

openPlotFile(fileName, fileIndex)

Arguments:

fileName [string] The full path to the file to be opened.

fileIndex [int] (Optional) An ID used to identify this file for subsequent calls. The file index can be specified explicitly when opening a file. If left unspecified, the file index will be initialized to the next available number. If the file index is specified explicitly and a file is currently opened at that index, the open file will be replaced with the new file.

Description:

Opens a TRAC-B plot file. If the file must not be demultiplexed

Example:

```
# Open a TRAC-B TRCGRF plot file
TRACB.openPlotFile("./sample_data/trace/VsslSolidNoLevSS.grf")
```

openDmxFile(fileName, fileIndex)

Arguments:

fileName [string] The full path to the file to be opened.

fileIndex [int] (Optional) An ID used to identify this file for subsequent calls. The file index can be specified explicitly when opening a file. If left unspecified, the file index will be initialized to the next available number. If the file index is specified explicitly and a file is currently opened at that index, the open file will be replaced with the new file.

Description:

Opens a TRAC-B demultiplexed plot file.

Example:

```
# Open a demultiplexed TRAC-B TRCGRF plot file
TRACB.openDmxFile("./sample_data/trace/VsslSolidNoLevSS.dmx")
```

getValueAt(fileIndex, channelName, time)

Arguments:

fileIndex [int] The file index of the open file.

channelName [String] The string containing the channel name

time [double] (s) The analysis time in which the data will be extracted.

Description:

Returns a double containing the plot data for the indicated data channel at a specified transient time.

Example:

```
# Read liquid temperature at 100 seconds.
TRACB.getDataAt(1, ‘FRICWAV-050001’, 100.0)
```

getValuesAt(fileIndex, channelNames, time)

Arguments:

fileIndex [int] The file index of the open file.

channelNames [String[]] The strings containing the channel names

time [double] (s) The analysis time in which the data will be extracted.

Description:

Returns a list of doubles containing the plot data for the indicated data channels at a specified transient time.

Example:

```
# Read liquid temperature at 100 seconds.
TRACB.getDataAt(1, [‘FRICWAV-050001’, ‘FRICWAV-050002’], 100.0)
```

3.20 FileData Class

An array of FileData objects are returned by the getFileInfo() method. The class serves as a holder for information on the files that are currently opened by an interface.

Class Methods:

[string] getFilename()

Description:

Returns the fully qualified name of the open file.

[string] getFileType()

Description:

Returns a string identifying the type of the open file.

[int] getNumChnls()

Description:

Returns the number of data channels contained in the file.

[int] getNumTimeSteps()

Description:

Returns the number of time steps contained in the file.

Example Usage:

```
# Exercise FileData methods.
RELAP.openDmxFile("./sample_data/relap/relap.dmx")
RELAP.openPlotFile("./sample_data/relap/typpwr.rst")
info = RELAP.getFileInfo()
print("getFileInfo returned "+repr(len(info))+\
      " FileData instances.\n")
print("File 0 Info:")
print(info[0])
print("File 1 Info:")
print("info[1].getFileID()= "+repr(info[1].getFileID()));
print("info[1].getFileType()= "+info[1].getFileType());
print("info[1].getFilename()= "+info[1].getFilename());
print("info[1].getNumChnls()= "+repr(info[1].getNumChnls()));
print("info[1].getNumTimeSteps()= "\
      +repr(info[1].getNumTimeSteps()));
```

Output:

```
getFileInfo returned 2 FileData instances.
```

```
File 0 Info:
File ID [0]
File Type [RELAP5 DEMUX]
Filename [./sample_data/relap/relap.dmx]
Number of Channels [21429]
Number of Time Steps [278]

File 1 Info:
info[1].getFileID()= 1
info[1].getFileType()= RELAP5 RSTPLT
info[1].getFilename()= ./sample_data/relap/typpwr.rst
info[1].getNumChnls()= 8024
info[1].getNumTimeSteps()=
```

3.21 ChannelVector Class

The individual data channels read from the various plot files, text files and spreadsheets and stored as ChannelVector instances. As described in Section 4, mathematical operations performed on ChannelVectors may also produce new ChannelVector instances.

In addition to the set of (x,y) data pairs the ChannelVector includes a name, short description, engineering units and labels for the independent (x), and dependant (y) values and a flag indicating whether the data was read in SI or British units. These fields are populated when the ChannelVectors are read from the plot or experimental data files. The independent data is normally time in seconds when then the data is read from these files. In some cases, such as reading an axial temperature profile from a plot file at a specified time will result in other units for the independent data, in this case 'Length (ft)' may be returned from the read operation.

ChannelVectors read from a spreadsheet or text file may or may not include the name, description, labels or engineering unit data. ChannelVectors generated from mathematical operations do not typically populate these fields.

ChannelVectors may also be created directly in python and can subsequently be used for mathematical or plotting purposes.

Refer to the Generated API documentation for more information.

```
[SNAP Install]/pypost/doc/pypost/index.html
```

Constructor

ChannelVector(channelName, channelLabel, values, xUnitsType, xUnits, yUnitsType,yUnits, isSI)

Arguments:

channelName [string]:

The name of the channel.

e.g. "p-1000101", "pdome"

channelLabel [string]:

The descriptive label of the channel vector.

e.g. "Lower plenum pressure", "Dome Pressure"

values [(double,double), …]:

A list of (x,y) pairs of data points in the vector with x being the independent variable and y the dependent variable.

yUnitsType**[string]:**

The dependent variable’s engineering units type.

e.g. “Time”, “Length”, “Pressure”

yUnits[string]:

The dependent variable’s engineering units.

e.g. “s”, “ft”, “Pa”

xUnitsType**[string]:**

The independent variable’s engineering units type.

e.g. “Time”, “Length”, “Pressure”

xUnits[string]:

The independent variable’s engineering units.

e.g. “s”, “ft”, “Pa”

isSI[boolean] (Optional):

A Boolean value to determine whether this ChannelVector instance uses British or SI units. This will default to what the environment uses if not specified.

```
#Example of using the constructor
#A list of tuples containing (x,y) pairs
values = [(40.32,325.5),(45.77,270.5),(50.55,406)]
ChannelVector(‘psvt-01’,’Pressure Vs. Temperature’, values,\
                        ’Pressure’,’psi’,\
                        ‘Temperature’,’C’)
#This would use the environment default for isSI
```

Class Methods:

[string] getChannelName()

Description:

Returns the name of the channel which the data in the vector refers to.

[string] getChannelLabel()

Description:

Returns the channel label describing the channel.

[string] getXUnits()

Description:

Returns the unit for the independent values.

[string] getYUnits()

Description:

Returns the unit for the dependent values.

[void] setXUnits(String *type*)

Description:

Sets the independent units to the specified *type*.

[void] setYUnits(String *type*)

Description:

Sets the dependent units to the specified *type*.

[string] getXLabel()

Description:

Returns the label for the independent values.

[string] getYLabel()

Description:

Returns the label for the dependent values.

[void] setXLabel(String *label*)

Description:

Sets the independent description to the specified *label*.

[void] setYLabel(String *label*)

Description:

Sets the dependent description to the specified *label*.

4. Advanced Equation Interpreter

Data extracted from analysis code plot files and experimental data files typically consists of time dependent vector data stored in ChannelVector objects.

These objects can be used directly in Python equations to calculate values. For example, difference in pressure between two hydraulic cells from a RELAP5 calculation can be determined from the following simple Python coding:

```
# Plot the pressure drop across Pipe 106.
RELAP.openDmxFile("./sample_data/relap/relap.dmx")
p106i = R5(0,'p-106010000')
p106o = R5(0,'p-106070000')
deltaP = p106o - p106i

aptIntf = AptPlotIntf()
aptIntf.plotChannels(deltaP)
aptIntf.runCmds(["view 0.35, 0.15, 1.15, 0.85",
                 "title \"RELAP5 TYPPWR\"",
                 "subtitle \"SG Tube dP\"" ])
aptIntf.printFile( "PDF", "./results/deltaP.pdf")
```

Note that the minus operator "-" is overloaded to handle ChannelVector variables directly. The results of the calculation are stored in a newly created ChannelVector, deltaP. This new variable will contain all of the independent data values (time in this case) and the difference between the two dependent data values, 'p-106070000' and 'p-106010000', respectively.

In this simple case, both variables came from the same RELAP5 run so each time dependent vector contains the same time values. The new time dependent vector simply contains the difference between the two pressure values at each time step. However, when a mathematical operation uses variables from different sources such as a RELAP5 run and an NRC Databank experimental data file, the time values in each vector will most likely not be identical. The post-processor handles this situation by interpolating the dependant data from the second vector using the time values from the first vector. The resulting vector contains all of the time values that fall within the bounds of the second vector, with dependent values calculated at these times using interpolated data from the second vector.

4.1 Overloaded Operators

Several Python operators are overloaded to support ChannelVector objects as shown in the following Table:

Table 2. Overloaded Operators

Op	**Function**	**Supported Modes** **S =: Scalar Variable** **V=: Vector Variable**
+	Addition	S+V, V+S, V+V
-	Unary Minus	-V
-	Subtraction	S-V, V-S, V-V
*	Multiplication	S*V, V*S, V*V
/	Division	S*V, V*S, V*V
**	Exponentiation	V^S
=	Assignment	V=V
==	Equivalence	V==V
!= <>	Not Equal	V!=V V<>V
[m:n]	Slicing	V[m:n]

The overloaded addition, subtraction, multiplication and division operators can act on either two vectors, or a combination of a scalar and a vector variable. For example, the following equations are all valid and return vector variables:

```
p106i = R5(0,'p-106010000')
p106o = R5(0,'p-106070000')
ptot = p106o + p106i
p2 = 2.0 * p106o
p3 = p106i * 2.0
pave = (p106o + p106i)/2.0
```

If the operation involves two vector variables, the resulting vector will contain all of the x-values of the first vector that fall within the range of x-values of the second variable. Its y-values will be determined by performing the operation at each x-value using interpolated y-data from the second vector variable. If one of the operands is a scalar, the resulting vector will contain all of the x-values of the vector operand with its y-values determined by performing the operation using the scalar value and each y-value of the vector operand.

The exponentiation operator will raise all dependent values in a vector to a scalar power. The equivalence operator, ==, returns 'True' if all independent and dependent values in two vectors match. Likewise, the not-equal operators return 'False' when the vectors match.

Slicing is also supported for vector variables. Slicing allows you to obtain a new vector variable containing a subset of the points from a vector variable. For example:

```
vnew1 = p106i[2:5] # vnew1 contains points 3 through 6 of p106i
vnew2 = p106o[:5]  # vnew2 contains the first 6 points of p106o
```

4.2 Vector Functions

These functions return a new vector containing the modified data from the calling vector.

4.2.1 Math Functions

Vector Function	Returns	Description
acos()	ChannelVector	Returns a new vector with the dependent values calculated to the arc cosine value, in radians. Example: arcCosVec = vVar.acos()
asin()	ChannelVector	Returns a new vector with the dependent values calculated to the arc sine value, in radians. Example: arcSinVec = vVar.asin()
atan()	ChannelVector	Returns a new vector with the dependent values calculated to the arc tangent value, in radians. Example: arcTanVec = vVar.atan()
atan2()	ChannelVector	Returns a new vector with the dependent values calculated for the angle θ from the conversion of rectangular coordinates (x,y) to polar coordinates (r, θ). This function treats the dependent values in the vector as the abscissa coordinate in the calculation. Example: arcTanVec = vVar.atan2()
bound(*s1*,*s2*)	ChannelVector	Returns a new vector with the dependent values bounded by *s1* and *s2*. Requires that *s1* < *s2*. ***Arguments:*** *[double] s1 – The low bound.* *[double] s2 – The high bound.* Example: bounded = vVar.bound(1.2,5.5)
ceil()	ChannelVector	Returns a new vector with the dependent values calculated to the next largest integer value. Example: highVal = vVar.ceil()
cos()	ChannelVector	Returns a new vector with the dependent values calculated to the cosine value, in radians. Example: cosVec = vVar.cosh()
cosh()	ChannelVector	Returns a new vector with the dependent values calculated to the cosine hyperbolic value, in radians. Example: hyperCos = vVar.cosh()
deriv()	ChannelVector	Returns a new vector containing the slopes of the line between dependent values. The independent values are set to the midpoints of these lines. Example: slopeVec = vVar.deriv()

Vector Function	Returns	Description
exp()	ChannelVector	Returns a new vector with dependent values as the equation e^x where *x* is the dependent value. Example: expVec = vVar.exp()
fabs()	ChannelVector	Returns a new vector with the dependent values calculated to their absolute value. Example: absVec = vVar.fabs()
floor()	ChannelVector	Returns a new vector with the dependent values calculated to the next smallest integer value. Example: lowVal = vVar.floor()
integrate()	ChannelVector	Returns a new vector with the dependent values calculated to integral values using the trapezoidal method. Example: intVals = vVar.integrate()
ln()	ChannelVector	Returns a new vector with the dependent values calculated to their natural logarithmic value. Example: logVec = vVar.ln()
log()	ChannelVector	Returns a new vector with the dependent values calculated to their natural logarithmic value. Example: logVec = vVar.log()
log(*n*)	ChannelVector	Returns a new vector with the dependent values calculated to their logarithmic base-n value. ***Arguments:*** *[double] n – The selected logarithmic base* Example: logVec = vVar.log(3.3)
log10()	ChannelVector	Returns a new vector with the dependent values calculated to their logarithmic base-10 value. Example: logVec = vVar.log10()
maxXval()	Scalar	Returns the largest independent value contained within the vector. Example: maxTime = vVar. maxXval()
maxYval()	Scalar	Returns the largest dependent value contained within the vector. Example: maxPres = vVar. maxYval()
meanval()	Scalar	Returns the average mean value for the dependent variables in the vector. Example: mean = vVar.meanval()

Vector Function	Returns	Description
minXval()	Scalar	Returns the smallest independent value contained within the vector. Example: minTime = vVar. minXval()
minYval()	Scalar	Returns the smallest dependent value contained within the vector. Example: minPres = vVar. minYval()
sin()	ChannelVector	Returns a new vector with the dependent values calculated to the arc sine value, in radians. Example: sinVec = vVar.sin()
sinh()	ChannelVector	Returns a new vector with the dependent values calculated to the arc sine value, in radians. Example: hyperSinVec = vVar.sinh()
sqrt()	ChannelVector	Returns a new vector with the dependent values calculated to their square root values. Example: squareVec = vVar.sqrt()
stddev()	Scalar	Returns the standard deviation of the dependent values of the vector. Example: stdDev = vVar.stdDev()
tan()	ChannelVector	Returns a new vector with the dependent values calculated to the tangent value, in radians. Example: tanVec = vVar.tan()
tanh()	ChannelVector	Returns a new vector with the dependent values calculated to the tangent hyperbolic value, in radians. Example: hyperTanVec = vVar.tanh()

4.2.2 Utility Functions

Vector Function	Returns	Description
append(*other*)	ChannelVector	Returns a new vector with another vector's points appended to the current one. Any points in the first vector that fall within the range of the second vector are ignored. ***Arguments:*** *[ChannelVector] other – The vector to be appended to the calling vector.* Example: newVec = vVar.append(otherVVar)

Vector Function	Returns	Description
copySegment(*low*,*high*)	ChannelVector	Returns a copy of the segment from index *low* to index *high*. Note that this is the same as a slice operation. This function is maintained for legacy conversion. ***Arguments:*** *[int] low – The starting index* *[int] high – The ending index* Example: vecSeg = vVar.copySegment(0,30)
dropPoints(*low*,*high*)	ChannelVector	Returns a new vector with the points that have an independent value between low and high removed from the calling vector. ***Arguments:*** *[double] low – The lower end of the independent values to drop from the vector.* *[double] high – The higher end of the independent values to drop from the vector.* Example: dropVec = vVar.dropPoints(1.2, 3.7)
newEmptyVector()	ChannelVector	Creates a new empty vector. ***Arguments:*** Example: # Create a new vector containing: # 0.2 5.50 # 0.7 5.60 # 2.7 5.67 myVec = newEmptyVector () myVec = myVec.putPt(0.2,5.5) myVec = myVec.putPt(0.7,5.6) .putPt(2.7,5.67)
flipXY()	ChannelVector	Returns a new vector with the independent and dependent values swapped. Example: flippedVec = vVar.flipXY()
merge(*other*)	ChannelVector	Returns a new vector with another vector's points added to the current one. If any independent values are the same in both vectors, the paired dependent value in the second vector will take precedence in the new vector. ***Arguments:*** *[ChannelVector] other – The other vector to merge into the calling vector.* Example: newVec = vVar.append(otherVVar)

Vector Function	Returns	Description
newSet(*x1,x2,dx,y1,y2*)	ChannelVector	Creates a new linear vector. ***Arguments:*** *[double] x1 – The beginning of the independent values* *[double] x2 – The end of the independent values* *[double] dx –The slope of the vector* *[double] y1 – The beginning of the dependent values* *[double] y2 – The end of the dependent values.* Example: # Create a new vector containing: # 0.0 40.0 # 20.0 30.0 # 40.0 20.0 # 60.0 10.0 newVec = newSet(0,60,20,40,10)
putPt*(x,y*)	ChannelVector	Returns a new vector with a new point added to it. If the independent value of the provided point is currently contained in the vector, it will overwrite the paired dependent value. ***Arguments:*** *[double] x – The independent value* *[double] y – The dependent value* Example: ptVec = vVar.putPt(3.0,5.5)

shiftx(*offset*)	ChannelVector	Returns a new vector with a scalar offset added to each of the independent values.
		Arguments:
		[double] offset – The scalar amount to be added to the independent variables.
		Example:
		# Shift a time dependant vector by 6.3sec earlier
		shiftVec = vVar.shiftx(-6.3)

yvalAt(x)	Scalar	Returns the dependent value at the specified independent value. An exception is thrown if the argument falls outside of the range of the vector. The value is interpolated if the independent value falls between two points. ***Arguments:*** *[double] x – The independent value* Example: myY = vVar. yvalAt(7.2)

4.2.3 Units Conversion Functions

later

5. Working with External Applications

The PyPost library includes interfaces to external applications which facilitate post processing data analysis. These interfaces allow data to be read from and written to spreadsheets, as well as automated generation of presentation quality plots and reports.

5.1 AptPlot Interface

This interface provides direct access the AptPlot (https://www.snaphome.com) application. Please note that this is a preliminary implementation of the interface. Only a small set of functions are currently implemented. Refer to the Generated API documentation for more information.

```
[SNAP Install]/pypost/doc/pypost/codes/aptplot.html
```

The first step to working with this interface is to import the required modules. In this case, PyPost and the AptPlot support module.

```
import pypost
from pypost.codes.aptplot import *
```

Global Instance:

APTPLOT Singleton instance of AptPlot application interface class. This instance is headless which will prevent the user interface classes from loading, allowing batch scripts to be run on platforms without graphics capability.

Interface Methods:

clearData()

Description:

Clears the data sets from the current AptPlot graph.

runCmds(commands)

Arguments:

commands [string | string[]] This can be either a single or an array of batch commands to be run by AptPlot.

Description:

Sends a single or an array of batch commands to AptPlot for execution.

runScript(fileName)

Arguments:

fileName [string] The full path to the AptPlot batch script file to be run.

Description:

Runs an AptPlot batch file script.

plotChannels(channels, graph, set)

Arguments:

channels [string | string[]] This can be either a single channel name or an array of channel names specifying the data to plot.

graph [int] (optional) The AptPlot graph number that will be used to plot the data. If not provided, the data will be plotted in the current graph.

set [int] (optional) The AptPlot set number that will be used to plot the first data channel data. If provided, subsequent data channels will be assigned to consecutive set numbers. If not provided, the data will be plotted in the next available open sets.

Description:

Writes the plot data to a specified AptPlot graph. Engineering units and channel labels will be transmitted if available.

printFile(fileType, fileName)

Arguments:

fileType [string] The graphics file format to print. Currently supported formats include: ‘PDF’ ‘SVG’ ‘JPEG’ ‘TIFF’ ‘PNG’ ‘Postscript’ ‘EPS’ or ‘EMF’

fileName [string] The full path to the output graphics file.

Description:

Prints the current graph to a file in the specified format.

Example:

```
setUseBritishUnits()
TRACE.openPlotFile("./sample_data/trace/trace.xtv")
TRACE.openPlotFile("./sample_data/trace/trace_demux.xtv")
TRACE.getFileList()
trdata = TRACE.getData(0,['pn-1A01'])

COBRA.openPlotFile("./sample_data/cobra/cobra.grf")
COBRA.openPlotFile("./sample_data/cobra/cobra.dmx")
COBRA.getFileList()
codata = CO(0,['p-001001'])

APTPLOT.plotChannels(trdata)
APTPLOT.printFile( "PDF", "./results/TRACE.pdf")
APTPLOT.clearData()
APTPLOT.plotChannels(codata)
APTPLOT.printFile( "PDF", "./results/COBRA.pdf")
APTPLOT.clearData()
APTPLOT.plotChannels(codata/trdata)
APTPLOT.printFile( "SVG", "./results/COVERT.svg")
APTPLOT.clearData()

APTPLOT.plotChannels(trdata)
APTPLOT.runCmds(["view 0.35, 0.15, 1.15, 0.85",\
                 "title \"Sample TRACE Data\"",\
                 "title font 0",\
                 "title size 1.5",\
                 "title color 1",\
                 "subtitle \"TRACE Data\"",\
                 "subtitle font 0",\
                 "subtitle size 1.0",\
                 "subtitle color 1"])
APTPLOT.printFile( "PDF", "./results/TestCommands.pdf")
APTPLOT.runScript("./aptplot.b")
APTPLOT.printFile( "PDF", "./results/TestScript.pdf")
```

5.2 Microsoft Excel™ Interface

This interface provides direct access to tables of data in Excel spreadsheets. These spreadsheets can be saved in either XLS or the newer XLSX file formats.

Refer to the Generated API documentation for more information.

```
[SNAP Install]/pypost/doc/pypost/codes/parcs.html
```

The first step to working with this interface is to import the required modules. In this case, PyPost and the Excel support module.

```
import pypost
from pypost.codes.excel import *
```

Constructor:

MSExcelIntf(filename, mode)

Arguments:

fileName [string] The full path to the MS Excel spreadsheet file script file. The file extension must be either ".xls" to read/write legacy spreadsheet formats through Excel 2003, or ".xlsx" to work with the newer Open XML format available in Excel 2007 and later.

Description:

Constructs a new MSExcelIntf instance to interface with a Microsoft Excel spreadsheet.

Interface Methods:

close()

Description:

Write the new or modified file and close the spreadsheet. This must be the last method called on the interface. If this method is not called all changes will be discarded.

readDouble(sheet, cell)

readDouble (sheet, column, row)

Arguments:

sheet [int] (optional, default=0) The index of sheet that will be used.

cell [String] (optional, default=”A1”) The cell to read the string.

column [int] (optional, default=0) The cell’s column index to read the string.

row [int] (optional, default=0) The cell’s row index to read the string.

Description:

Reads the contents of a single spreadsheet cell into a double. Supply either the cell name (”B25”) or the cell’s column and row indexes. All indexes are zero-based.

readString(sheet, cell)

readString(sheet, column, row)

Arguments:

sheet [int] (optional, default=0) The index of sheet that will be used.

cell [String] (optional, default=”A1”) The cell to read the string.

column [int] (optional, default=0) The cell’s column index to read the string.

row [int] (optional, default=0) The cell’s row index to read the string.

Description:

Reads the contents of a single spreadsheet cell into a String. Supply either the cell name (”B25”) or the cell’s column and row indexes. All indexes are zero-based.

readVectors(numVec, numHdrRows, numDataRows, sheet, cell)

readVectors(numVec, numHdrRows, numDataRows, sheet, scol, srow)

Arguments:

numVec [int] (optional, default=1) The number of vectors that will be read from the spreadsheet.

numHdrRows [int] (optional, default=3) The number of header rows in the data range.

numDataRows [int] (optional, default=-1) The number of data rows in the data range.

sheet [int] (optional, default=0) The index of sheet that will be used.

cell [String] (optional, default="A1") The starting cell to read the data.

scol [int] (optional, default=0) The starting cell's column index to read the data.

srow [int] (optional, default=0) The starting cell's row index to read the data.

Description:

Reads a set of vectors from a spreadsheet starting at a specified location. Either the cell name (e.g. "B25") or the cell's column and row zero-based indexes may be entered. The coordinates of the range of cells that are read is:

(scol, srow) to (scol+(numVec +1), srow+numHdrRows+numDataRows)

Up to three header rows are read corresponding to the vector name, the engineering units type, and the engineering units, respectively. The engineering units may be enclosed in parenthesis.

The header rows are followed by numDataRows rows of data. The first column of data contains the independent data values shared by all vectors. The remaining vecNum columns contain the dependent data for each vector.

For example, assuming the coordinate of the upper left cell in the following table is D12 on the second sheet of the workbook:

	p-1000201	pdome	tsteam
Time	Pressure	Pressure	Temperature
(s)	(Pa)	(Pa)	(K)
0.0	5.5012e7	5.52e7	551.0
3.0	5.6562e7	5.67e7	551.1
6.0	5.9872e7	5.91e7	551.3
9.0	5.8768e7	5.82e7	551.6
12.0	5.8772e7	5.82e7	553.0

The data can be read using the following code:

```
inFile=MSExcelIntf("results/exceltest1.xls")
setUseBritishUnits()
vects = inFile.readVectors(3,3,5,1,"D12")
inFile.close();
```

The vects list would then contain three channel vectors: p-1000201, pdome and tsteam.

writeDouble(val, sheet, cell)

writeDouble (val, sheet, column, row)

Arguments:

val [double] A double to be written to a cell.

sheet [int] (optional, default=0) The index of the sheet that will be used.

cell [String] (optional, default="A1") The cell to write the string.

column [int] (optional, default=0) The cell's column index to write the string.

row [int] (optional, default=0) The cell's row index to write the string.

Description:

Writes a String to a spreadsheet cell. Supply either the cell name ("B25") or the cell's column and row indexes. All indexes are zero-based.

writeString(str, sheet, cell)

writeString(str, sheet, column, row)

Arguments:

str [String] A string to be written to a cell.

sheet [int] (optional, default=0) The index of the sheet that will be used.

cell [String] (optional, default="A1") The cell to write the string.

column [int] (optional, default=0) The cell's column index to write the string.

row [int] (optional, default=0) The cell's row index to write the string.

Description:

Writes a String to a spreadsheet cell. Supply either the cell name ("B25") or the cell's column and row indexes. All indexes are zero-based.

writeVectors(vectors, sheet, cell)

writeVectors(vectors, sheet, column, row)

Arguments:

vectors [ChannelVector | ChannelVector []] This can be either a single vector or an array of vectors that will be written to the spreadsheet.

sheet [int] (optional, default=0) The index of sheet that will be used.

cell [String] (optional, default="A1") The starting cell to write the data.

scol [int] (optional, default=0) The starting cell's column index to write the data.

srow [int] (optional, default=0) The starting cell's row index to write the data.

Description:

Writes a set of vectors to the spreadsheet at a specified location.

Three header rows are written to the spreadsheet containing the vector name, the engineering units type, and the engineering units, for each vector, respectively. This is followed by the floating point vector data. The first column contains the independent data for the first vector (typically time). The second column contains the dependent data for the first vector. The remaining columns are filled with the dependant data of the remaining vectors interpolated to the independent data values contained in the first column.

Example:

```
# Read in 3 vectors the write them to a new location
ss1=MSExcelIntf("C:\Users\kkj\test\relap5Test.xls")
setSIUnits()
vData = readVectors(3,3,5,1,"D12")
ss1.writeVectors(vData, 1,"J12")
ss1.close()
```

5.3 Text File Interface

This interface provides direct access to read and write test files. This interface extends the python *file* class. All methods available to *file*, (read(), seek(), writelines(), close() etc..) can be called on TextFileIntf instances. Files should be closed after all operations on them are completed. Refer to the Generated API documentation for more information.

[SNAP Install]/pypost/doc/pypost/codes/text.html

Constructor:

TextFileIntf(filename, mode)

Arguments:

fileName [string] The full path to the text file.

mode [string] The python file access mode. This should be one of the following:

"r" - Open for reading only.

"w" - Open for writing. If the file exists it will be overwritten.

"a" - Open for writing, append to the end of the file.

"r+" - Open for reading and writing.

"w+" - Open for reading and writing. File contents will be overwritten.

"a+" - Open for reading and writing, append to the end of the file.

Description:

Constructs a new TextFileIntf instance to interface to a text file.

Interface Methods:

readVectors(numVec, numHdrRows, separator)

Arguments:

numVec [int] The number of vectors to be read from the file.

numHdrRows [int] (optional, default=3) The number of header rows in the data range

separator [char] (optional, default=',') The separator character used to parse the data fields.

Description:

Reads a set of vectors from the text file starting at the current location.

Up to three header rows are read containing the vector names, the engineering units types, and the engineering units, respectively. The parsing character is read on the first line, followed by up to numVec entries containing the vector names, one for each vector. Up to (numVec+1) entries are read from the second and third lines. The first entry of the second line contains the Engineering Units Type of the independent variable for all of the vectors (typically "Time"). This is followed by the dependent variable Engineering Units Type for each of the vectors. In a similar manner, the third header line contains the engineering units for the independent variable, followed by the dependent variable engineering units for each vector. The engineering units may be enclosed in parenthesis.

The header rows are followed by rows of data. The first column of data contains the independent data values shared by all vectors. The remaining vecNum columns contain the dependent data for each vector. Data is read until either a blank line or the end of file is reached.

For example, the data read from a spreadsheet in the last section, could also be read from the following a comma separated value (csv) file:

```
,p-1000201,pdome,tsteam
Time,Pressure,Pressure,Temperature
(s),(Pa),(Pa),(K)
0.0,5.5012e7,5.52e7,551.0
3.0,5.6562e7,5.67e7,551.1
6.0,5.9872e7,5.91e7,551.3
9.0,5.8768e7,5.82e7,551.6
12.0,5.8772e7,5.82e7,553.0
```

Using the following code:

```
inFile= TextFileIntf("results/texttest.csv","r")
setUseBritishUnits()
vects = inFile.readVectors(3)
inFile.close();
```

The vects list would then contain three channel vectors: p-1000201, pdome and tsteam.

setXformat(format)

Arguments:

format [String] The format used to write the independent (X) data.

Description:

Sets the formatting string for writing the independent data to the file.

The formatting string follows the convention detailed in the java.util.Formatter class (see: https://docs.oracle.com/javase/7/docs/api/java/util/Formatter.html).

Format strings for numeric types follow the following syntax:

```
 %[flags][width][.precision]conversion
```

The most common flags include:

```
'-'   The result will be left-justified.
'+'   The result will always include a sign
' '   The result will include a leading space for values >0
'0'   The result will be zero-padded
```

Width is the minimum number of characters to be written to the output. For the floating-point conversions 'e', 'E', and 'f' the precision is the number of digits after the decimal separator. If the conversion is 'g' or 'G', then the precision is the total number of digits in the resulting magnitude after rounding.

Typical Examples of formatting strings:

```
%12.2F      1234567890.12
%12.2e              1.23e9
%12.2G              1.23E9
```

setYformat(format)

Arguments:

format [String | String[]] The format used to write the dependent (Y) data. If a single string is provided it will be used for all dependent columns. If an array of strings is provided, the first string will be used for the first dependant column, second string will be used for the second dependant column, and so on. The last formatting string will be used for any remaining columns.   See setXformat(format) above for a description of the formatting strings.

Description:

Sets the formatting strings for writing the dependent data to the file.

writeVectors(vectors, separator)

Arguments:

vectors [ChannelVector | ChannelVector []] This can be either a single vector or an array of vectors that will be written to the file.

separator [char] (optional, default=',') The separator character used to parse the data fields.

Description:

Writes a set of vectors to the current location in the file.

Three header rows are written to the file containing the vector name, the engineering units type, and the engineering units, for each vector, respectively. This is followed by the floating point vector data. The first line starts with the parsing character, followed by numVec entries, one for each vector. (numVec+1) entries are written to the second and third lines. The first entry of the second line contains the Engineering Units Type of the independent variable for all of the vectors (typically “Time”). This is followed by the dependent variable Engineering Units Type for each of the vectors. In a similar manner, the third header line contains the engineering units for the independent variable, followed by the dependent variable engineering units for each vector. The engineering units are enclosed in parenthesis.

The header lines are followed by lines of data. The first entry on each row contains the independent data value shared by all vectors. The remaining vecNum values contain the dependent data for each vector. A line with a single period '.' is written after the last line of data.

Example:

```
# Read in 3 vectors from a spreadsheet and write
# them to a text file.
ss1=MSExcelIntf("C:\Users\kkj\test\relap5Test.xlsx")
tf1=TextFileIntf("C:\Users\kkj\test\test.txt","w+")
setSIUnits()
vData = ss1.readVectors(3,3,5,1,"D12")
tf1.writeVectors(vData,"\t")
ss1.close()
tf1.close()
```

Produces the following file:

```
       p-1000201  pdome     tsteam
Time   Pressure   Pressure  Temperature
(s)    (Pa)       (Pa)      (K)
0.0    5.5012e7   5.52e7    551.0
3.0    5.6562e7   5.67e7    551.1
6.0    5.9872e7   5.91e7    551.3
9.0    5.8768e7   5.82e7    551.6
12.0   5.8772e7   5.82e7    553.0
```

6. ParaView Reader Plug-Ins

PyPost includes a set of ParaView reader plug-ins that utilize various PyPost plot file interfaces to read plot file data. The reader plug-ins will require that PyPost is on the PYTHONPATH variable (see Calling PyPost from Python Applications for more information). The plug-ins can be loaded from ParaView's Tools > Manage Plugins menu, or loaded in a ParaView pvpython compatible script with the LoadPlugin function. An example of loading the Trace reader plug-in is shown below.

```
from paraview.simple import *
LoadPlugin('/home/SNAP/python/pypost/paraview/TraceReader.py', remote =
False, ns=globals())
```

Once a plug-in is loaded, the associated reader can be used to read plot files through the File > Open menu or by instantiating the class in the reader plug-in that is responsible for reading the desired plot file. Each of the reader plug-ins will store selected plot file data in a vtkTable object, with the row data containing each of the selected channels data. The field data arrays keep a parallel set of arrays containing the units for each of the data channels.

The table below shows the reader plug-ins and the classes they use for reading particular plot file types.

Plug-Ins	Classes	Extensions
CobraReader.py	COBRAReader	.dmx, .grf
ContainReader.py	CONTAINReader	.pibplot
ExtdataReader.py	EXTDATAReader	.plt, .dmx
FastReader.py	FASTReader	.pib, .plot
FrapconReader.py	FRAPCONReader	.pib, .plot
FraptranReader.py	FRAPTRANReader	.pib, .plot
GothicReader.py	GOTHICReader	.SGR, .dmx
MelcorReader.py	MELCORDMXReader	.dmx
	MELCORPTFReader	.ptf
MooseExodusReader.py	MOOSEExodusReader	.e
NrcdbReader.py	NRCDBReader	.bin
ParcsReader.py	PARCSBPFReader	.bpf

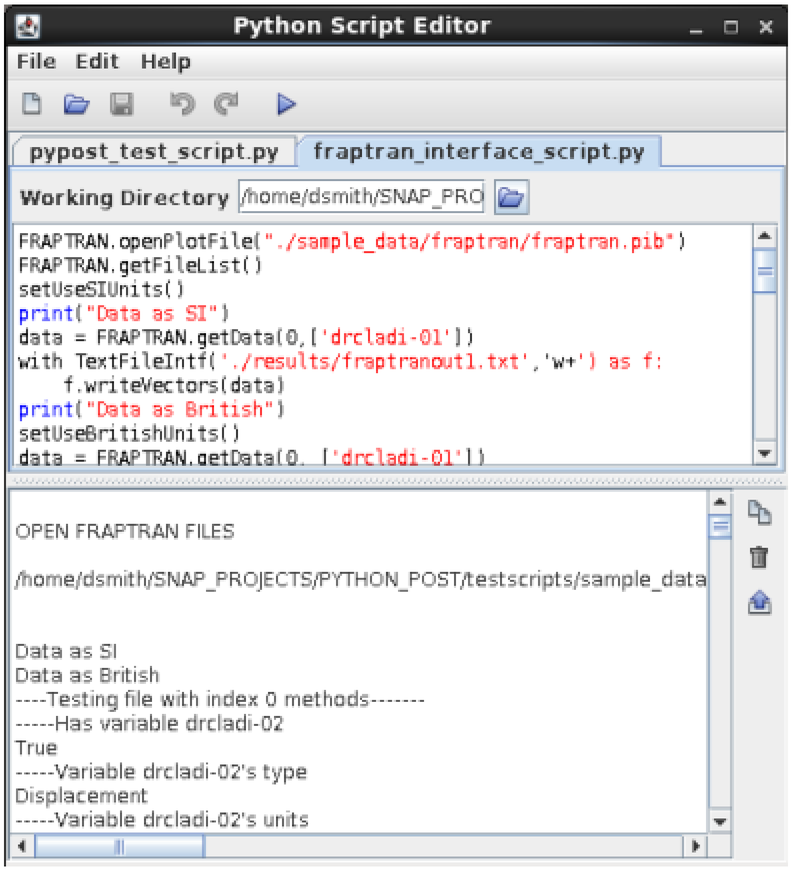
Plug-Ins	Classes	Extensions
RelapReader.py	RELAPReader	.dmx, .plt, rst
Retran3DReader.py	RETRAN3DReader	.ascii, .dmx, .plt
TraceReader.py	TRACEXTVReader: xtv	.xtv
	TRACBGRFReader: grf	.grf
	TRACBDMXReader: dmx	.dmx

An example of opening a TRACE XTV file from within a script is shown below:

```
from paraview.simple import *
traceXtv = TRACEXTVReader(FileName='sample_data/trace/trace.xtv')
```

7. PyPost Graphical User Interface

The Python Post module includes a Graphical User Interface for use with the Python Post library. The graphical interface provides the ability to created, modify and execute python scripts using the PyPost libraries. The user interface contains an editing panel, for modifying python scripts, and an output panel where the standard output is redirected. Figure 1 below displays the PyPost graphical user interface.



```
FRAPTRAN.openPlotFile("./sample_data/fraptran/fraptran.pib")
FRAPTRAN.getFileList()
setUseSIUnits()
print("Data as SI")
data = FRAPTRAN.getData(0,['drcladi-01'])
with TextFileIntf('./results/fraptranout1.txt','w+') as f:
    f.writeVectors(data)
print("Data as British")
setUseBritishUnits()
data = FRAPTRAN.getData(0, ['drcladi-01'])
```

```
OPEN FRAPTRAN FILES

/home/dsmith/SNAP_PROJECTS/PYTHON_POST/testscripts/sample_data


Data as SI
Data as British
----Testing file with index 0 methods-------
-----Has variable drcladi-02
True
-----Variable drcladi-02's type
Displacement
-----Variable drcladi-02's units
```

Figure 1: The PyPost GUI Application

7.1 Main Toolbar

The main toolbar sits at the top of the frame and contains several functions for use with the Python Post script editor. The options include creating new scripts, opening existing scripts, and saving scripts, as well as undo and redo options.

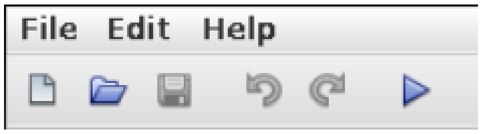


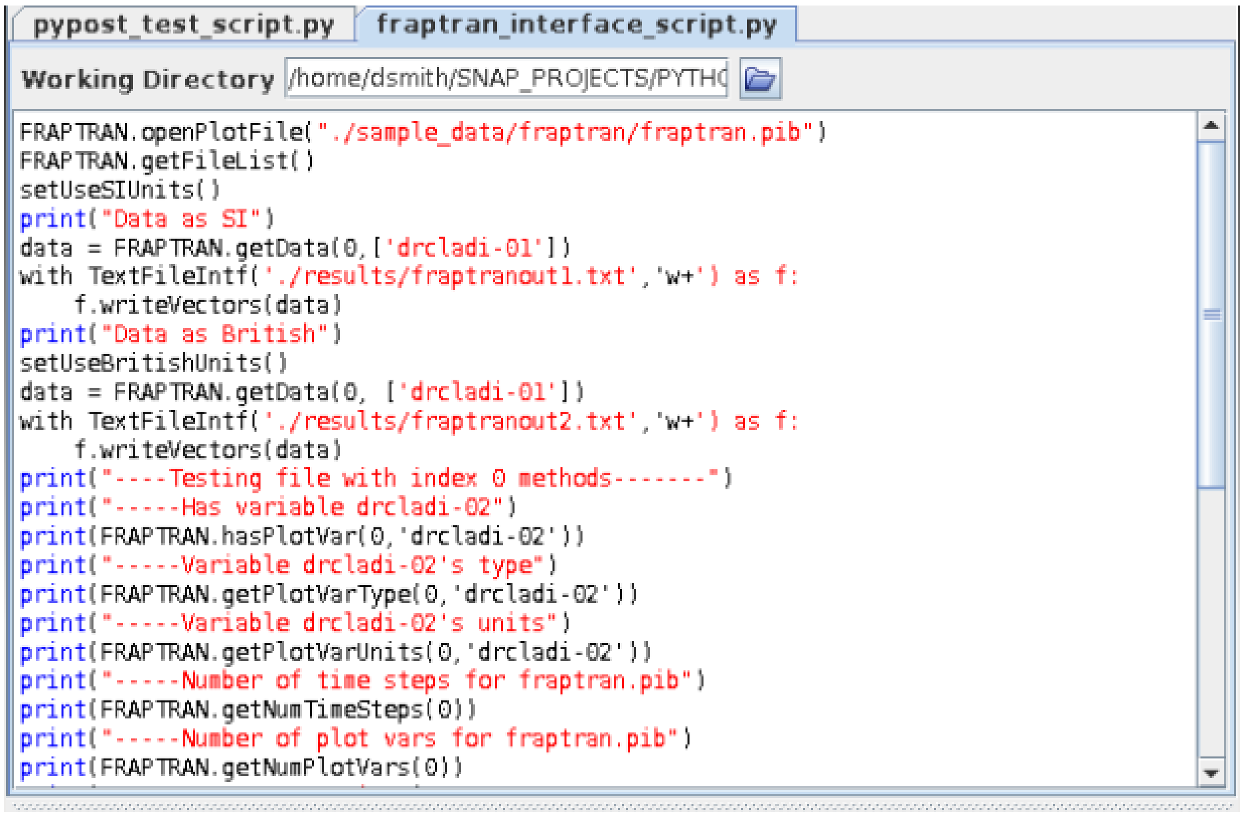
Figure 2: Main Toolbar

7.1.1 Execute Button

The execute button, which appears as a right-ward facing triangle will execute the current python script. The standard output from the script will be displayed in the output panel.

7.2 Editing Panel

This is the primary panel for the PyPost graphical application. This panel contains one or more scripts, open in separate tabs. Each tab displays the current working directory for that script, along with the contents of the script. When a new script is created, or opened a new tab will be added to the editing panel. The current script may be executed by pressing the Execute button from the Main Toolbar. Figure 3 below shows the Editing Panel with two scripts open.



```
FRAPTRAN.openPlotFile("./sample_data/fraptran/fraptran.pib")
FRAPTRAN.getFileList()
setUseSIUnits()
print("Data as SI")
data = FRAPTRAN.getData(0,['drcladi-01'])
with TextFileIntf('./results/fraptranout1.txt','w+') as f:
    f.writeVectors(data)
print("Data as British")
setUseBritishUnits()
data = FRAPTRAN.getData(0, ['drcladi-01'])
with TextFileIntf('./results/fraptranout2.txt','w+') as f:
    f.writeVectors(data)
print("----Testing file with index 0 methods------")
print("-----Has variable drcladi-02")
print(FRAPTRAN.hasPlotVar(0,'drcladi-02'))
print("-----Variable drcladi-02's type")
print(FRAPTRAN.getPlotVarType(0,'drcladi-02'))
print("-----Variable drcladi-02's units")
print(FRAPTRAN.getPlotVarUnits(0,'drcladi-02'))
print("-----Number of time steps for fraptran.pib")
print(FRAPTRAN.getNumTimeSteps(0))
print("-----Number of plot vars for fraptran.pib")
print(FRAPTRAN.getNumPlotVars(0))
```

Figure 3: The Editor Panel

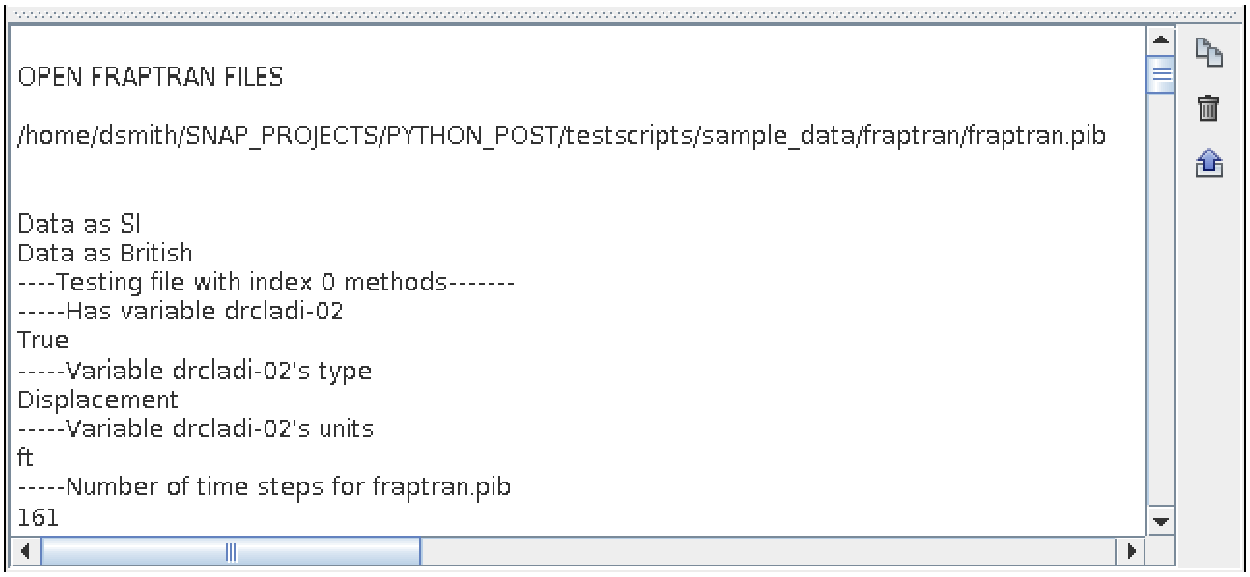
7.2.1 Working Directory

All file interactions defined in a script will be performed relative to the working directory of that script. When a script is opened, the working directory is set to the path where the script was located. Each script's working directory is independent from other open scripts.

7.3 Output Panel

The output panel, which appears on the lower half of the application is the output panel. This panel contains the output from executed scripts. This panel will also display the standard error and standard output text from the python interpreter.

To the right of the output panel are three utility buttons. The first button copies the entire text of the output panel onto the system clipboard. The second button clears the output panel of all text, and the final button exports the output to a local file. Figure 4 below shows an example of the output from executing a script.



```
OPEN FRAPTRAN FILES

/home/dsmith/SNAP_PROJECTS/PYTHON_POST/testscripts/sample_data/fraptran/fraptran.pib


Data as SI
Data as British
----Testing file with index 0 methods-------
-----Has variable drcladi-02
True
-----Variable drcladi-02's type
Displacement
-----Variable drcladi-02's units
ft
-----Number of time steps for fraptran.pib
161
```

Figure 4: The Output Panel

8. PyPost Test Procedures

A comprehensive test suite has been developed to verify proper operation of the software. Tests are organized into four categories, channel vector function tests, plot & experimental data file I/O interface tests, external application interface tests, and miscellaneous tests.

Test suite is installed as a ZIP file (testscripts/testscripts.zip) and should be extracted into a separate directory prior to execution. Each set of tests is located in a separate subdirectory. The entire suite can run from the newly extracted testscripts directory using the following command:

```
> pypost.[exe|sh] pypost_test_script.py
```

where the extension, '.exe' or '.sh', is based on the platform, Widows or Linux, respectively.

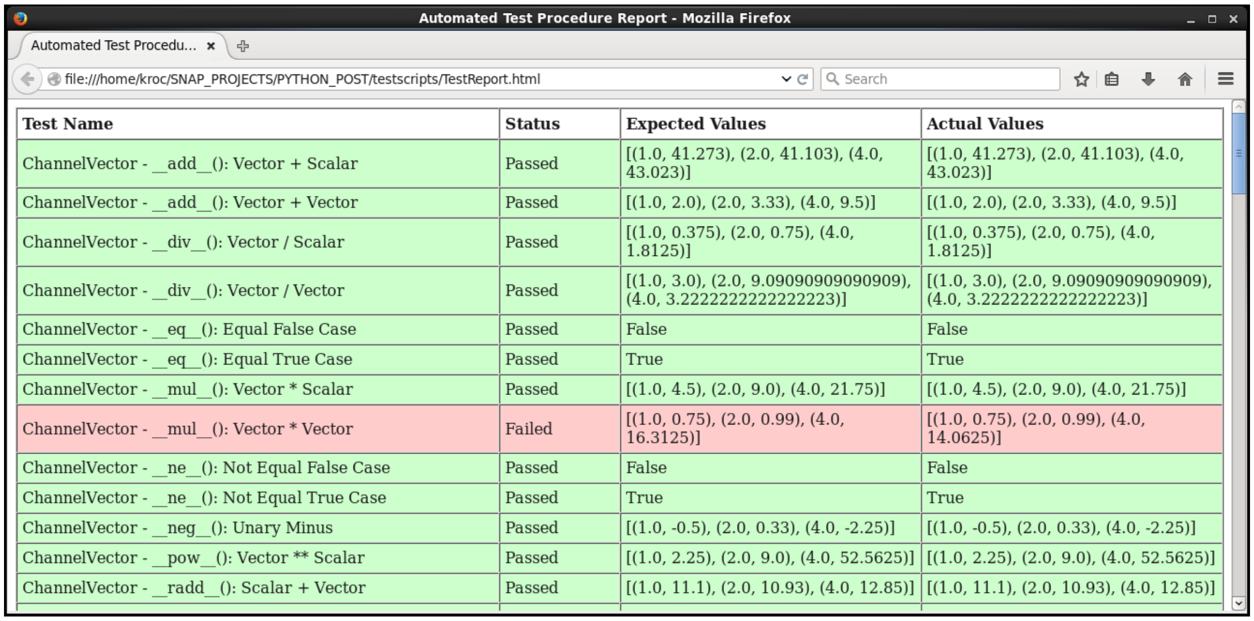
The test script can be executed using an external python application if the appropriate libraries are added to the PYTHONPATH environment variable. The following elements must be added to the python path:

```
[SNAP Install]/pypost/python:
[SNAP Install]/pypost/testscripts:
```

The channel vector function tests are used to verify proper execution of overloaded operators, math functions, error checks, utility functions, I/O routines, and print/read options. These tests exercise each with scalar-vector and vector-vector combinations as appropriate. The results of each test are compared against a set of expected values.

Each of the supported analysis code plot file and experiments data file types are tested to verify that data is properly read from each file format and that each of the various interface methods return the appropriate data. The external application tests verify that each interface function operates as expected. Any additional tests such as performance testing the vector interpolation routines are included in the miscellaneous category.

Any results files generated during test script execution are written to the 'results' subdirectory. After the tests are executed, an html report is generated summarizing each of the tests. This report will indicate the passed/failed status of each verification point. Failed verification points will be highlighted with a red background as shown below.


Automated Test Procedure Report - Mozilla Firefox
Automated Test Procedu...
file:///home/kroc/SNAP_PROJECTS/PYTHON_POST/testscripts/TestReport.html
Search

Test Name	Status	Expected Values	Actual Values
ChannelVector - __add__(): Vector + Scalar	Passed	[(1.0, 41.273), (2.0, 41.103), (4.0, 43.023)]	[(1.0, 41.273), (2.0, 41.103), (4.0, 43.023)]
ChannelVector - __add__(): Vector + Vector	Passed	[(1.0, 2.0), (2.0, 3.33), (4.0, 9.5)]	[(1.0, 2.0), (2.0, 3.33), (4.0, 9.5)]
ChannelVector - __div__(): Vector / Scalar	Passed	[(1.0, 0.375), (2.0, 0.75), (4.0, 1.8125)]	[(1.0, 0.375), (2.0, 0.75), (4.0, 1.8125)]
ChannelVector - __div__(): Vector / Vector	Passed	[(1.0, 3.0), (2.0, 9.09090909090909), (4.0, 3.2222222222222223)]	[(1.0, 3.0), (2.0, 9.09090909090909), (4.0, 3.2222222222222223)]
ChannelVector - __eq__(): Equal False Case	Passed	False	False
ChannelVector - __eq__(): Equal True Case	Passed	True	True
ChannelVector - __mul__(): Vector * Scalar	Passed	[(1.0, 4.5), (2.0, 9.0), (4.0, 21.75)]	[(1.0, 4.5), (2.0, 9.0), (4.0, 21.75)]
ChannelVector - __mul__(): Vector * Vector	Failed	[(1.0, 0.75), (2.0, 0.99), (4.0, 16.3125)]	[(1.0, 0.75), (2.0, 0.99), (4.0, 14.0625)]
ChannelVector - __ne__(): Not Equal False Case	Passed	False	False
ChannelVector - __ne__(): Not Equal True Case	Passed	True	True
ChannelVector - __neg__(): Unary Minus	Passed	[(1.0, -0.5), (2.0, 0.33), (4.0, -2.25)]	[(1.0, -0.5), (2.0, 0.33), (4.0, -2.25)]
ChannelVector - __pow__(): Vector ** Scalar	Passed	[(1.0, 2.25), (2.0, 9.0), (4.0, 52.5625)]	[(1.0, 2.25), (2.0, 9.0), (4.0, 52.5625)]
ChannelVector - __radd__(): Scalar + Vector	Passed	[(1.0, 11.1), (2.0, 10.93), (4.0, 12.85)]	[(1.0, 11.1), (2.0, 10.93), (4.0, 12.85)]

8.1 Extending the Test Procedure

The implementations of the PYPOST test scripts provide a simple method of adding new test cases. To add a ChannelVector function test, place a Python file in the verification directory with the following format:

```
expected=<expected values>
firstParam=<parameter>
secondParam=<parameter>
thirdParam=<parameter>
fourthParam=<parameter>
fifthParam=<parameter>
operation=<function name>
testName=<test description>
```

- ***expected*** can be of any type and is what is to be returned from the function called.
- ***firstParam*** through ***fifthParam*** are the parameters that can be passed to the function. These are in the order that they are expected, but do not need to be declared if unused.
- ***operation*** is the name of the function that is being called with the provided parameters. Note that the overloaded operator functions have two underscores on either side, i.e. '__add__' for the case of the addition operator.
- ***testName*** is the descriptive name of the test that will show up in the generated TestReport.html.

These function tests will automatically be added to the generated report file upon running the pypost_test_script.py file located in the testscripts directory.

The analysis code plot file and experiments data file interfaces are tested using routines to verify proper return data for the interface methods and to compare extracted data to baseline results. The verify method takes three arguments, the expected value, the actual value, and a descriptive label for the report. For example, the following test script segment verified that the hasPlotVar method in the RELAP5 interface returns True when checking for the existence of channel 'httemp-100100101' in the open RELAP5 file with a file ID of 0.

```
verify(True,RELAP.hasPlotVar(0,'httemp-100100101'), \
        "RELAP: Has variable httemp-100100101")
```

The baselineCompare method is used to verify that the data values returned from the plot and experimental data files match a baseline set of values that have been previously checked for accuracy. The method takes three arguments, a descriptive label for the report, the baseline file and the extracted data to be compared against the baseline.

The following example compares the 'VLZN-040101' data channel read from a TRAC-B trcgrf file to the results contained in file tracbBaseline.txt:

```
baselineCompare('TRACB', \
                  "interface_tests/baselines/tracbBaseline.txt", \
                  [TRACB.getData(0,['VLZN-040101'])])
```