

# Chapter 7

## Scripting

Scripting provides a way to control the operation of HEC-DSSVue in a non-interactive way. The user can build and save scripts to be executed later – possibly on different data sets.

This chapter provides an introduction to scripting, describing the components of the user interface, scripting language and application program interface (API), and offering examples illustrating how to use the API.

### 7.1 Executing Scripts

HEC-DSSVue allows the execution of scripts in interactive and batch modes. Scripts are executed interactively by starting the HEC-DSSVue program and selecting the desired script from the Toolbar or the Script Selector. Scripts are executed in batch mode by starting the HEC-DSSVue program with a script file name as a parameter (e.g. `HecDssVue.bat c:\test\myScript.py`).

Interactive scripts are not passed any parameters upon script execution. In a script executed interactively the variable `sys.argv` is a list of length 1, with the only element set to the empty string (e.g. `sys.argv = [""]`).

Scripts executed in batch mode may take parameters from the command line (e.g. `HecDssVue.bat c:\test\myScript.py a b c`). In a script executed in batch mode the variable `sys.argv` is a list whose length is one greater than the number of parameters passed on the command line, with the first element set to the file name of the executing script and the remaining elements set to the parameters (e.g. `sys.argv = ["c:\\test\\myScript.py", "a", "b", "c"]`).

#### 7.1.1 Script Selector

The **Script Selector** (Figure 7.1) displays buttons for all the available scripts which have the “Display Script on Toolbar” box checked (see Section 7.2.2, “Editor Panel”). Buttons are displayed in alphabetical order.

To access the **Script Selector**, select the **Script Selector** command from the **Utilities** menu of HEC-DSSVue. Once the **Script Selector** is open, it will remain open until you close it.

When you press a button, the Jython script engine will execute the associated script.



Figure 7.1 Script Selector

## 7.2 Script Browser

The **Script Browser**, shown in Figure 7.2, allows you to add, delete, and modify scripts.

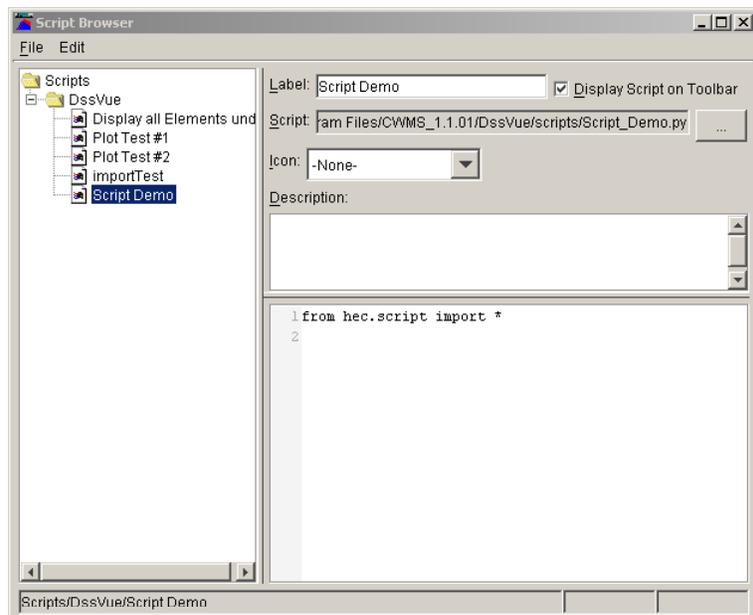


Figure 7.2 Script Browser

You can access the **Script Browser** from HEC-DSSVue's **Utilities** menu by clicking **Script Browser**. Alternatively, from the shortcut menu of the **Script Selector**. In the Script Selector, right click on a button to access the shortcut menu, then select **Edit**. The Script Browser will open with that script selected and ready for editing.

Components of the **Script Browser** include the Menu Bar, the Editor Panel, and the Tree Hierarchy. The following sections describe these components.

## 7.2.1 Menu Bar

The **Menu Bar** (Figure 7.3) contains two primary menu items, **File** and **Edit**.



Figure 7.3 Menu Bar

### File Menu Commands

- |                    |  |
|--------------------|--|
| <b>New</b>         | Creates a new script stored at the currently selected position. Only available when a folder node is the selected node in the scripts tree.  |
| <b>Open Script</b> | Edits the currently displayed script. Double clicking on the script also edits the currently displayed script. Only available when a script node is the selected node in the scripts tree. |
| <b>Import</b>      | Imports a file into the script browser. If the import is successful the browser is placed in edit mode. Only available when a folder node is the selected node in the scripts tree.        |
| <b>Save</b>        | Saves the current script. Only available when a script is being edited.  |
| <b>Delete</b>      | Deletes the currently opened script. Prompts user for confirmation. Only available when a script node is the selected node in the scripts tree.  |
| <b>Test</b>        | Executes the currently selected script.  |
| <b>Close</b>       | Closes the Script Browser Window.  |

### Edit Menu Commands

- Cut** Cuts the currently selected text in the script window or the currently selected tree node to the system clipboard.
- Copy** Copies the currently selected text in the script window or the currently selected tree node to the system clipboard.
- Paste** Pastes the contents of the system clipboard into the script window at the current cursor location or if a tree node was cut/copied pastes at the currently selected folder node.

## 7.2.2 Editor Panel

You can select and edit scripts in the **Editor Panel** of the Script Browser (Figure 7.4).

The **Label** field allows you to specify the label displayed on a script's button in the Script Selector.

**Script** displays the name of the file in which the script is stored.

 is the **Select Script** button, which allows you to select a previously-created script file.

**Display Script on Toolbar**, when checked, enables the script to display in the **Script Selector** and the **Toolbar**.

When you uncheck this option, the script will not display on the Script Selector or **Toolbar**.

The **Icon** field allows you to choose the Icon to display for the script's button. If you do not select an icon, the script name displays in the script's button.

The **Description** field allows you to add a description of the script. The first line of your description serves as a tooltip for the corresponding button on the Script Selector and **Toolbar**.

The **Script Text** field contains the script text itself and serves as an editing window for creating new scripts.

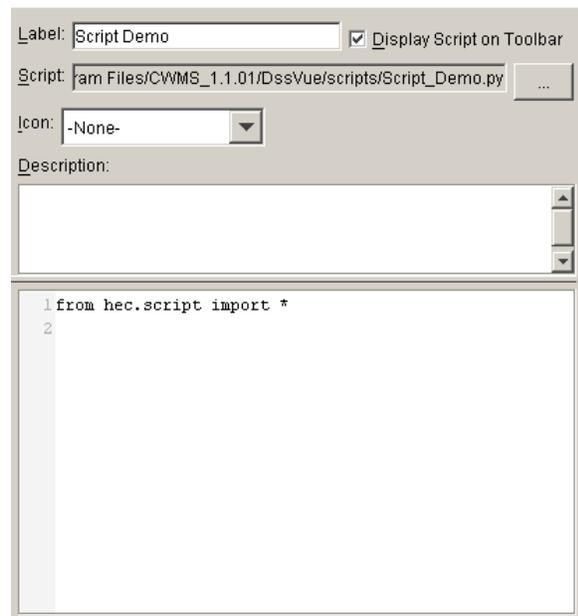


Figure 7.4 Editor Panel

### 7.2.3 Tree Hierarchy

The **Tree Hierarchy** (Figure 7.5) uses a Windows Explorer-style tree structure to allow you to navigate folders in your directory structure and access scripts. By default, the scripts are stored in a “scripts” directory under the directory where HEC-DSSVue was installed.

The Tree Hierarchy also has a shortcut menu that displays **Cut**, **Copy**, and **Edit** commands for script nodes and **New**, **Import**, and **Paste** for folder nodes.

To access the shortcut menu, point to a node in the “tree” and right-click with your mouse.



Figure 7.5 Tree Hierarchy

## 7.3 Scripting Basics

Scripting in HEC-DSSVue is accomplished using Jython, an implementation of the Python programming language designed specifically for integration with the Java programming language. More information about Jython can be found at the official Jython website – [www.jython.org](http://www.jython.org).

Python (of which Jython is an implementation) is an interpreted language with simple syntax and high-level data types. This section is not a comprehensive Python tutorial, but rather a simple primer to allow the creation of simple scripts. This primer does not cover defining classes in Python.

The official Python website - [www.python.org](http://www.python.org) - has links to online Python tutorials as well as programming books.

### 7.3.1 Outputting Text

Text information can be displayed in the console window using the print statement which has the syntax:

```
print [item[, item...]]
```

The *items* are comma-separated and do not need to be of the same type. The print statement will insert a space between every specified item in the output.

**Example 1: Outputting Text**

```
print "Testing myFunction, i =", i, ", x =", x
```

### 7.3.2 Data Types

Python has integer, long integer, floating-point, imaginary number, and sequence and dictionary data types. Sequences are divided into mutable (or changeable) sequences called lists, immutable sequences called tuples. Strings are special tuples of characters that have their own syntax. Dictionaries are like sequences but are indexed by non-numeric values. In addition, Python also has a special type called None, which is used to indicate the absence of any value.

Python does not have a specific type for boolean (logical, or “true / false”) data. Tests, such as conditional expressions, which must evaluate to true or false are conducted such that the result is false if the expression evaluates to None, integer or floating-point zero, or an empty sequence. Any other result is true. Python statements that generate Boolean information (such as the **if** statement) generate integer 0 for false and integer 1 for true. This becomes an issue in Jython which allows calling Java functions and methods which expect a Java boolean for input or generate a Java boolean for output. Jython maps these boolean values to integer 0 or 1. Documentation for the HEC-DSSVue API uses the term Constants.TRUE (1), or Constants.FALSE (0), or sometimes the shorthand “0/1”, for arguments (these are constants defined to 1 and 0, respectively, in `hec.script`), and “0/1” to specify that the return type is a Python integer, but its value is restricted to 0 or 1, corresponding to a Java boolean. The `hec.script` module supplies constants to use in these situations.

There are also situations regarding the HEC-DSSVue API where it is necessary or desirable to set a time-series value to “missing” or to test whether a time-series value is missing. The `hec.script` module also supplies a constant to use in these situations.

The currently-defined constants in the hec.script module are:

Constant	Type	Represents
Constants.TRUE	integer	true
Constants.FALSE	integer	false
Constants.UNDEFINED	floating-point	missing data value

It is recommended that these defined constants be used where applicable for portability and clarity.

### Example 2: Variable Types

```
# set some integer values
i = 0
j = 1
k = -10998
m = Constants.TRUE

# set a long integer
n = 79228162514264337593543950336L

# set some floating-point values
x = 9.375
y = 6.023e23
z = -7.2e-3
t = Constants.UNDEFINED

# set some strings
string_1 = "abc"
string_2 = 'xyz'
string_3 = "he said \"I won't!\""
string_4 = 'he said "I will not!"'
string_5 = """this is a
multi-line string"""

# set a tuple – tuples are contained within ()
tuple_1 = (1, 2, "abc", x, None)

# set a list – lists are contained within []
list_1 [1, 2, "abc", x, tuple_1]

# set a dictionary, using key : value syntax
# dictionaries are contained within {}
dict_1 = {"color" : "red", "size" : 10, "list" : [1, 5, 8]}
```

Indexing into sequence types is done using `[i]` where `i` starts at 0 for the first element. Subsets of sequence types (called slices) are accessed using `[i:j]` where `i` is the first element in the subset and `j` is the element *after* the last element in the subset. If negative numbers are used to specify and index or slice, the index is applied to the *end* of the sequence, where `[-1]` specifies the last element, `[-2]` the next-to last and so on. If `i` is omitted in slice syntax it defaults to 0. If `j` is omitted in slice syntax it defaults to the length of the sequence, so `list_1[0:len(list_1)]` is the same as `list_1[:]`. Indexing into dictionaries is done using `[x]` where `x` is the key.

The number of elements in a sequence type or dictionary is returned by the `len()` function.

### Example 3: Sequence Indexing and Slicing

```
string_4[3]           # 4th element
string_4[3:5]        # 4th & 5th elements
list_1[-1]           # last element
list_1[2:-1]         # 3rd through next-to-last element
list_1[2:len(list_1)] # 3rd through last element (also list_1[2:])
dict_1["size"]       # value associated with "size" key
i = len(list_1)      # length of list_1
```

## 7.3.3 Variables

Python variable names consist of an upper- or lower-case letter or the “`_`” (underscore) character followed by an unlimited number of upper- or lower-case characters, digits or underscore characters.

Variables are assigned values by use of the “`=`” (equals) character. A sequence may be assigned to multiple variables using a single equals character. Variable names are case sensitive, so the name “`startdate`” is not the same name as “`startDate`”.

### Example 4: Assigning Values to Variables

```
i = 0
j = 1
k = -10998
string_1 = "abc"
i, j, k, string_1 = 0, 1, -10998, "abc"
```

### 7.3.4 Operators

Each of the following operators can be used in the form  $a = b \text{ operator } c$ . Each can also be used as an assignment operator in the form  $a \text{ operator} = b$  (e.g.  $a += 1$ ,  $x **= 2$ ).

+	arithmetic addition
-	negation or arithmetic subtraction
*	arithmetic multiplication
/	arithmetic division
**	arithmetic power
%	arithmetic modulo
&	bit-wise and
	bit-wise or
~	bit-wise not
^	bit-wise xor (exclusive or)
<<	bit-wise left shift
>>	bit-wise right shift

Each of the following operators returns 0 (false) or 1 (true) and can be used in conditional expressions as discussed in Section 7.3.7.

>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to
!=	not equal to
==	equal to

### 7.3.5 Comments

Python uses the “#” (hash) character to indicate comments. Everything from the “#” character to the end of the line is ignored by the interpreter. Comments may not be placed after a program line continuation (“\”) character on the same input line.

### 7.3.6 Program Lines

Unless otherwise indicated, every input line corresponds to one logical program statement. Two or more statements can be combined on line input by inserting the “;” (semicolon) character between adjacent statements. A single statement may be continued across multiple input lines by ending each line with the “\” (back slash) character. Comments may not be placed after a program line continuation (“\”) character on the same input line.

**Example 5: Input vs. Program Lines**

```
# multiple statements per line
r = 1; pi = 3.1415927; a = pi * r ** 2

# multiple lines per statement
a = \
    pi * \
    r ** 2
```

Input lines are grouped according to their function. Input lines forming the body of a conditional, loop, exception handler, or function or class definition must be grouped together. Input lines not in any of the construct comprise their own group. In Python, grouping of input lines is indicated by indentation. All lines of a group must be indented the same number of spaces. A horizontal tab character counts as 8 spaces on most systems. In some Python documentation, a group of input lines is called a *suite*.

**Example 6: Input Line Grouping**

```
# this is the main script group
dist = x2 - x1
if dist > 100.:
    # this is the "if" conditional group
    y = dist / 2.
    z = y ** 2.
else :
    # this is the "else" conditional group
    y = dist.
    z = y ** 2. / 1.5
# back to main script group
q = y * z
```

### 7.3.7 Conditional Expressions

Conditional expressions have the form:

```
if [not] condition :  
    if-group  
[elif [not] condition :  
    elif-group]  
[else :  
    else-group]
```

The “:” (colon) character must be placed after each condition.

The condition in each test is an expression built from one or more simple conditions using the form:

```
simple-condition ( and | or ) [not] simple-condition
```

Parentheses can be used to group conditions.

The simple-condition in each expression is either an expression using one of the conditional operators mentioned in Section 7.3.4 or is of the form:

```
item [not] in sequence
```

#### Example 7: Conditional Expressions

```
if (x < y or y >= z) and string_1.index("debug") != -1 :  
    # do something  
...  
elif z not in value_list or (x < z * 2.5) :  
    # do something different  
...  
else :  
    # do something else
```

If the statement group to be processed upon a condition is a single statement, that statement may follow the condition on the same line (after the colon character).

#### Example 8: Simple Conditional Expressions

```
if x1 < x2 : xMax = x2  
else      : xMax = x1
```

## 7.3.8 Looping

Python supports conditional looping and iterative looping. For each type, the body of the loop (the loop-group) can contain **break** and/or **continue** statements.

The **break** statement immediately halts execution of the loop-group and transfers control to the statement immediately following the loop.

The **continue** statement skips the remainder of the current iteration of the loop-group and continues with the next iteration of the loop-group.

### 7.3.8.1 Conditional Looping

Python supports conditional looping with the *while* statement, which has the form:

```
while condition :  
    loop-group
```

Conditional looping executes the body of the loop (the loop-group) as long as the condition evaluates to true.

#### Example 9: Conditional Looping

```
# print the first 10 characters  
string_1 = "this is a test string"  
i = 0  
while i < 10 :  
    print string_1[i]  
    i += 1
```

### 7.3.8.2 Iterative Looping

Python supports iterative looping with the *for* statement, which has the form:

```
for item in sequence :  
    loop-group  
[else :  
    else-group]
```

Iterative looping executes the body of the loop (the loop-group) once for each element in *sequence*, first setting *item* to be that element. If the iteration proceeds to completion without being interrupted by a break statement the else-group will be executed, if specified.

The **range([start,] stop[, increment])** helper function generates a sequence of numbers from *start* (default = 0) to *stop*, incrementing by *increment* (default = 1). Thus `range(4)` generates the sequence (0, 1, 2, 3).

#### Example 10: Iterative Looping

```
# print the first 10 characters
string_1 = "this is a test string"
for i in range(10) :
    print string_1[i]

# print all the characters
string_1 = "this is a test string"
for i in range(len(string_1)) :
    print string_1[i]

# print all the characters (more Python-y)
string_1 = "this is a test string"
for c in string_1 :
    print c
```

### 7.3.9 Defining and Using Functions

In Python, functions are defined with the syntax:

```
def functionName([arguments]) :
    function-body
```

Function names follow the same naming convention as variable names specified in Section 7.3.2. The arguments are specified as a comma-delimited list of variable names that will be used within the function-body. These variables will be positionally assigned the values used in the function call. More complex methods of specifying function arguments are specified in Python tutorials and references listed at the official Python website ([www.python.org](http://www.python.org)).

A function must be defined in a Python program before it can be called. Therefore, function definitions must occur earlier in the program than calls to those functions.

A function may optionally return a value or sequence of values.

**Example 11: Defining And Using Functions**

```
def printString(stringToPrint) :  
    "Prints a tag plus the supplied string"  
    tag = "function printString : "  
    print tag + stringToPrint  
  
def addString(string_1, string_2) :  
    "Concatenates 2 strings and returns the result"  
    concatenatedString = string_1 + string_2  
    return concatenatedString  
  
testString = "this is a test"  
printString(testString)  
wholeString = addString("part1:", "part2")  
printString(wholeString)  
printString(addString("this is ", "another test"))
```

### 7.3.10 Modules, Functions and Object Methods

A function is a procedure which takes zero or more parameters, performs some action, optionally modifies one or more of the parameters and optionally returns a value or sequence of values.

A class is the definition of a type of object. All objects of that type (class) have the same definition and thus have the same attributes and behavior. Classes may define functions that apply to individual objects of that type. These functions are called methods.

An object is an instance of a class, which behaves in the way defined by the class, and has any methods defined by the class.

Python provides many functions and classes by default. In our examples we have used functions `len()` and `range()` which Python provides by default. We have also used the classes `list` and `string`, which Python also provides by default. We didn't use any object methods of the class `list`, but we used the string method `index()` in the example in Section 7.3.7

(`string_1.index("debug") != -1`). It is important to note that the object method `index()` doesn't apply to the string class in general, but to the specific string object `string_1`.

There are other functions and classes which Python does not provide by default. These functions and classes are grouped into modules according to their common purpose. Examples of modules are "os" for operating system functions and "socket" for socket-based network functions. Before any of the functions or classes in a module can be accessed, the module must be imported with the `import` statement, which has the syntax:

```
from module import *
```

Other methods of using the import statement are specified in Python tutorials and references listed at the official Python website ([www.python.org](http://www.python.org)). In the Jython implementation, Java packages can be imported as if they were Python modules, and the Java package `java.lang` is imported by default.

**Example 12: Using A Function From An Imported Module**

```
# use the getcwd() function in the os module to get
# the current working directory

from os import *
cwd = getcwd()
```

A module *does not* have to be imported in order to work with objects of a class defined in that module *if* that object was returned by a function or method already accessible. For example, the Python module “string” does not have to be imported to call methods of string objects, but *does* have to be imported to access string functions.

### 7.3.11 Handling Exceptions

Certain errors within a Python program can cause Python to raise an exception. An exception that is not handled by the program will cause the program to display error information in the console window and halt the program.

Python provides structured exception handling with the following constructs:

```
try :
    try-group
except :
    except-group
[else :
    else-group]
```

```
try :
    try-group
finally :
    finally-group
```

In the try-except-else construct, if an exception is raised during execution of the try-group control immediately transfers to the first line of the except-group. If no exception is raised during execution of the try-group control transfers to the first line of the else-group, if present. If there is no exception raised and no else-group is specified, the control transfers to the first line after the except-group.

In the try-finally construct, control is transferred to the first line of the finally-group when either an exception is raised during the execution of the try-group or the try-group completes without an exception.

The two constructs cannot be combined into a try-except-finally construct, but the same effect can be obtained by making a try-except-else construct the try-group of a try-finally construct.

### Example 13: Exception Handling

```
try :
  try :
    string_1.find(substring) # may raise an exception
  except :
    print substring + " is not in " + string_1
    # do some stuff that might raise another exception
    ...
  else :
    print substring + " is in " + string_1
    # do some stuff that might raise another exception
    ...
finally :
  print "No matter what, we get here!"
```

More exception handling information, including filtering on specific types of exceptions, exception handler chains, and raising exceptions, is provided in Python tutorials and references listed at the official Python website ([www.python.org](http://www.python.org)).

## 7.4 Displaying Messages

It is often useful to display messages to inform the user that something has occurred, to have the user answer a Yes/No question, or offer debugging information to help determine why a script isn't working as expected. Text information can be displayed in the console window as described in Section 7.3.1, "Outputting Text."

### 7.4.1 Displaying a Message Dialog

#### MessageBox ()

The MessageBox class in the hec.script module has several functions used to display messages in message box dialogs. The message box can be one of four different types: Error, Warning, Informational or Plain.

**Note:** Do not use the MessageBox functions in a script that is to run unattended since these functions cause scripts to pause for user interaction.

Table 7.1 describes MessageBox functions.

**Table 7.1 - MessageBox Functions**

showError(string message, string title)	None	Display an error dialog to you with the message and title
showWarning(string message, string title)	None	Display a warning dialog to you with the message and title
showInformation(string message, string title)	None	Display a Informational dialog to you with the message and title
showPlain(string message, string title)	None	Display a plain dialog to you with the message and title
showYesNo(string message, string title)	string	Display a Yes/No dialog to you with the message and title
showYesNoCancel(string message, string title)	string	Display a Yes/No/Cancel dialog to you with the message and title
showOKCancel(string message, string title)	string	Display a Ok/Cancel dialog to you with the message and title

**Example 14: Display Error Dialog**

```
from hec.script import *
# display error dialog to user
MessageBox.showError("An Error Occurred", "Error")
```

**Example 15: Display OK/Cancel Dialog**

```
from hec.script import *
ok=MessageBox.showOkCancel("Continue with Operation", "Confirm")
```

## 7.5 Reading and Writing to HEC-DSS Files

Reading or writing a data set from a DSS file involves using functions and methods from 3 classes in the `hec.hecmath` module: `DSS`, `DSSFile` and `HecMath`. The `DSS` class is used to get a `DSSFile` object which represents a DSS File. The `DSSFile` object is then used to get individual data sets out of the DSS File by returning a `HecMath` object.

### 7.5.1 DSS Class

```
DSS.open(string filename)
DSS.open(string filename, string startTime, string endTime)
```

The `DSS` class is used to gain access to a HEC-DSS File, as illustrated in Example 16.

**Example 16: Opening a DSS File**

```
theFile = DSS.open("MyFile.dss")
or
theFile = DSS.open("MyFile.dss", "01Jan2002,1300", "02Jan2002,1300")
```

## 7.5.2 DSSFile Objects

`DSSFile` objects are used to read and write data sets in a DSS file. Table 7.2 describes `DSSFile` object methods.

**Table 7.2 - DSSFile Methods**

Function	Returns	Description
<code>read(string pathname)</code>	HecMath	Return an HecMath object that holds the data set specified by pathname.
<code>read(string pathname, string startTime, string endTime)</code>	HecMath	Return an HecMath object that holds the data set specified by pathname with the specified time window.
<code>setTimeWindow(string startTime, string endTime)</code>	None	The default time window for this DSSFile.
<code>write(HecMath dataset)</code>	integer	Write the data set to the DSS file.
<code>close()</code>	None	Close the DSS file.

### Example 17: Reading a DSS Data Set

```
from hec.hecmath import *
# open myFile.dss and read a data set
theFile = DSS.open("myFile.dss")
flowDataSet = theFile.read("/BASIN/LOC/FLOW/01NOV2002/1HOUR/OBS/")
theFile.close()
```

## 7.6 Display Objects

HecMath objects – data sets read from DSS files - cannot be added to plots and tables directly. To add data from a HecMath object to a plot or table, a DisplayObject must first be created from the HecMath object using the `createDisplayObject` function in the `hec.hecmath.DisplayUtilities` module.

### Example 18: Creating a Display Object

```
from hec.hecmath import *
from hec.hecmath.DisplayUtilities import *
theFile = DSS.open("myFile.dss")
flowDataSet = theFile.read("/BASIN/LOC/FLOW/01NOV2002/1HOUR/OBS/")
theFile.close()
displayObject = createDisplayObject(flowDataSet)
```

## 7.7 Plotting Basics

Figure 7.6 identifies the title, viewport, axis label, axis tics, and legend of a plot, each of which are accessible via scripts.

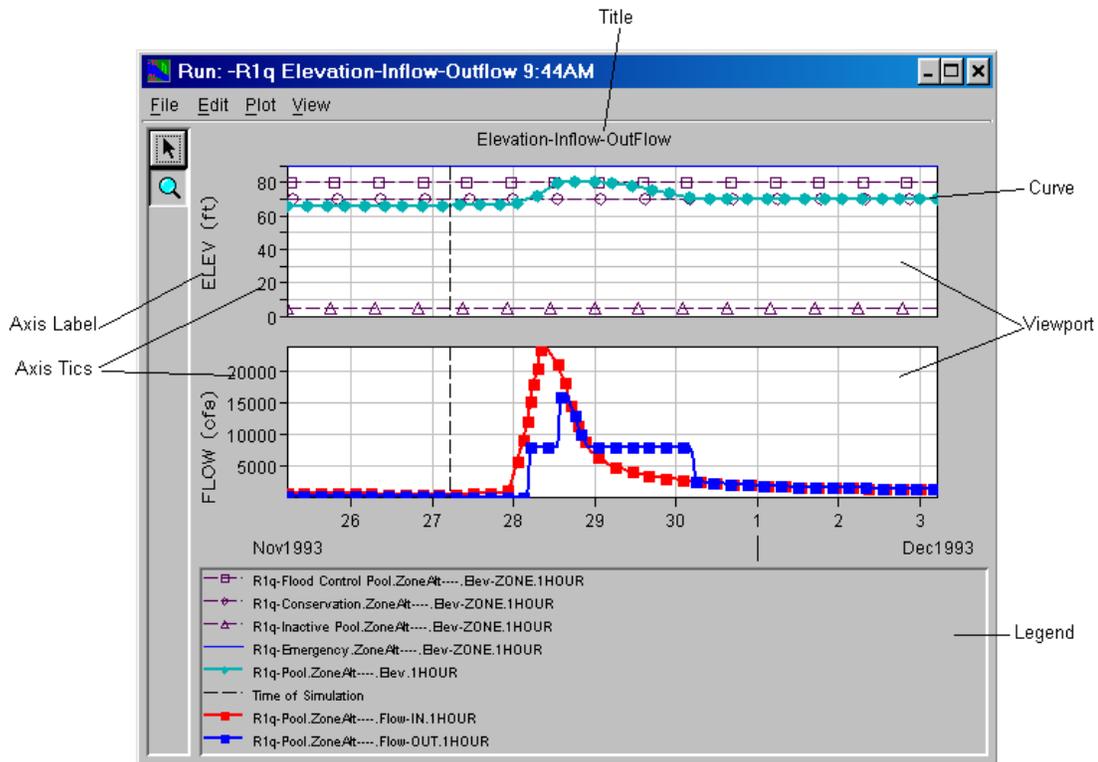


Figure 7.6 Plot Components

### 7.7.1 Plot Class

```
Plot.newPlot()
Plot.newPlot(string title)
```

The Plot class in the hec.script module is used to create a new Plot dialog. It contains two methods to create a Plot dialog, each of which returns a G2dDialog object.

#### Example 19: Creating a Plot

```
myPlot = Plot.newPlot()
or
thePlot = Plot.newPlot("Elevation vs Flow")
```

## 7.7.2 Changing Plot Component Attributes

Not all Plot Component attributes are drawn by default. Just setting the attribute may not make that attribute draw on the plot. Often it is necessary to tell that attribute to draw by calling `setAttributeOn()` method.

Example 20 illustrates reading a flow data set from a DSS file, plotting the data set, setting the minor Y grid color to black and making it display.

### Example 20: Plotting DSS Data

```
from hec.script import *
from hec.hecmath import *
from hec.hecmath.DisplayUtilities import *

theFile = DSS.open("myFile.dss")           # open myFile.dss
thePath = "/BASIN/LOC/FLOW/01NOV2002/1HOUR/OBS/"
flowDataSet = theFile.read(thePath)       # read a path name
thePlot = Plot.newPlot()                  # create the plot
do = createDisplayObject(flowDataSet)     # create a display object
thePlot.addDisplayObject(do)              # add the flow data set to
                                           # the plot
thePlot.showPlot()                        # show the plot
viewport0=thePlot.getViewport(0)          # get the first viewport
viewport0.setMinorGridYColor("black")     # set the viewport's minor Y
                                           # grid to black
viewport0.setDrawMinorYGridOn()           # tell the minor Y grid to
                                           # display
```

### 7.7.3 G2dDialog Objects

**G2dDialog objects** are the dialog that plots display in. Table 7.3 describes **G2dDialog** object methods.

**Table 7.3: G2dDialog Object Methods**

Method	Returns	Description
addDisplayObject(DisplayObject dspObj)	None	Add the DisplayObject specified by dspObj to the plot. Must be called before showPlot()
applyTemplate(string templateFile)	None	Apply the given template to this plot
configurePlotLayout()	None	Display the configure plot layout dialog for this plot
configurePlotTypes()	None	Display the configure plot types dialog
copyToClipboard()	None	Copy the plot to the system clipboard
defaultPlotProperties()	None	Display the default plot properties dialog
exportProperties()	None	Allows you to save the properties of the plot to a disk.
exportProperties(string templateName)	None	Allows you to save the properties of the plot to the file specified by templateName.
getCurve(HecMath dataSet)	G2dLine	Return the G2dLine for the DataSet specified by dataSet
getCurve(string dssPath)	G2dLine	Return the G2dLine for the path specified in dssPath
getLocation()	Point	Return the location of the dialog in screen coordinates
getPlotTitle()	G2dTitle	Return the Title for the G2dDialog
getViewport(HecMath dataSet)	Viewport	Return the Viewport that contains the curve specified by dataSet
getViewport(int viewportIndex)	Viewport	Return the viewport at index specified by viewportIndex

Method	Returns	Description
getViewport(string dataSetPath)	Viewport	Return the Viewport that contains the curve specified by dataSetPath
Hide()	None	Hide the dialog
PlotProperties()	None	Display the plot properties dialog for this plot
print()	None	Display the print dialog for this plot
printMultiple()	None	Display the print multiple dialog for this plot
printPreview()	None	Display the print preview dialog for this plot
printToDefault()	None	Prints using the printer defaults such as page format and printer. This method does not display the printer dialog for user interaction.
saveAs()	None	Display the saveAs dialog for this plot
saveToJpeg(string fileName)	None	Save the plot to the Jpeg file specified by fileName
saveToMetafile(string filename)	None	Save the plot to the Windows Meta file specified by filename
SaveToPng(string fileName)	None	Save the plot to the Portable Network Graphics file specified by filename
saveToPostscript(string fileName)	None	Save the plot to the PostScript file specified by filename
setLocation(int x,int y)	None	Sets the location of the dialog in screen coordinates
showPlot()	None	Show the dialog
tabulate()	TableFrame	Display the table view of this plot

Example 21 shows how to create a new plot with a flow data set, show the plot, and place at location 50,50 on the screen.

### Example 21: Plot Dialog

```

from hec.script import *           # for Plot class
from hec.hecmath import *        # for DSS class
from hec.hecmath.DisplayUtilities import * # for display objects
theFile = DSS.open("myFile.dss") # open myFile.dss
thePath = "/BASIN/LOC/FLOW/01NOV2002/1HOUR/OBS/"
flowDataSet = theFile.read(thePath) # read a path name

do = createDisplayObject(flowDataSet) # create flow display obj
thePlot = Plot.newPlot()           # create a new Plot
thePlot.addDisplayObject(do)       # add the flow display obj
thePlot.showPlot()                 # show the plot
thePlot.setLocation(50,50)         # moves plot to 50,50

```

## 7.7.4 Viewport Objects

Viewport objects hold the data set curves. Table 7.4 describes **Viewport** object methods.

**Table 7.4: Viewport Object Methods**

Method	Returns	Description
addXAxisMarker()	None	Adds a Marker Line to the Xaxis. Displays the Marker Line Properties for you to edit the properties of the marker
addXAxisMarker(floating-point value)	None	Add an X Axis marker at the location specified by value
addXAxisMarker(string value)	None	Add a X Axis marker at the location specified by value
addYAxisMarker()	None	Display the Add Y Axis marker Dialog
addYAxisMarker(string value)	None	Add a Y Axis marker at the location specified by value
editProperties()	None	Display the Edit Properties dialog for this Viewport
getAxis(string axisName)	Axis	return the <b>Axis</b> specified by <b>axisName</b> for this Viewport

Method	Returns	Description
getAxisLabel(string axisName)	AxisLabel	Return the <b>AxisLabel</b> for the axis specified by <b>axisName</b> for this Viewport
getAxisTics(string axisName)	AxisTics	Return the <b>AxisTics</b> for the axis specified by <b>axisName</b> for this Viewport
GetBackground()	Color	Return the background color for the Viewport
getBackgroundString()	string	Return the background color name for the Viewport as a String
getBorderWeight()	float	Return the border weight for this Viewport
getFillPatternString()	string	Return the fill pattern for this Viewport as a String
getGridYColor()	Color	Return the Y grid color for this Viewport
getGridXColor()	Color	Return the X grid color for this Viewport
getGridXColorString()	string	Return the X grid color for this Viewport as a String
getGridYColorString()	string	Return the Y grid color for this Viewport as a String
getMajorGridXColorString()	string	Return the major grid X color for this Viewport as a String
getMajorGridYColorString()	string	Return the major grid Y color for this Viewport as a String
getMajorXGridWidth()	integer	Return the major X Grid width for this Viewport
getMajorYGridWidth()	integer	Return the major Y Grid width for this Viewport
getMinorGridXColorString()	string	Return the minor grid X color for this Viewport as a String
getMinorXGridWidth()	integer	Return the minor X Grid width for this Viewport
getMinorYGridWidth()	integer	Return the minor Y Grid width for this Viewport

Method	Returns	Description
isBackgroundDrawn()	0/1	Return whether the background is drawn or not
isBorderDrawn()	0/1	Return whether the border is drawn for this Viewport
isMajorXGridDrawn()	0/1	Return whether the Major X Grid is drawn
isMajorYGridDrawn()	0/1	Return whether the Major Y Grid is drawn
isMinorXGridDrawn()	0/1	Return whether the Minor X Grid is drawn
isMinorYGridDrawn()	0/1	Return whether the Minor Y Grid is drawn
setBackground(string colorString)	None	Set the background to the color specified by colorString
setBorderColor(string borderColor)	None	Set the border color for this Viewport
SetBorderWeight(floating-point borderWeight)	None	Set the border weight for this Viewport
setDrawBackgroundOff()	None	Set the background not to draw for this Viewport
setDrawBackgroundOn()	None	Set the background to draw for this Viewport
setDrawBorderOff()	None	Set the border not to draw for this Viewport
setDrawBorderOn()	None	Set the border to draw for this Viewport
setDrawMajorXGridOff()	None	Set the major X grid not to draw for this Viewport
setDrawMajorXGridOn()	None	Set the major X grid to draw for this Viewport
setDrawMajorYGridOff()	None	Set the major Y grid not to draw for this Viewport
setDrawMajorYGridOn()	None	Set the major Y grid to draw for this Viewport
setDrawMinorXGridOff()	None	Set the minor X grid not to draw for this Viewport
setDrawMinorXGridOn()	None	Set the minor X grid to draw for this Viewport
setDrawMinorYGridOff()	None	Set the minor Y grid not to draw for this Viewport

<b>Method</b>	<b>Returns</b>	<b>Description</b>
setDrawMinorYGridOn()	None	Set the minor Y grid to draw for this Viewport
setFillPattern(string pattern)	None	Set the fill pattern for this Viewport
setGridColor(string colorString)	None	Set the X and Y grid colors for this Viewport
setGridXColor(string colorString)	None	Set the X grid color to the color represented by colorString for this Viewport
setGridYColor(string colorString)	None	Set the Y grid color to the color represented by colorString for this Viewport
setMajorGridXColor(string majorGridXColor)	None	Set the major grid X color for this Viewport
setMajorGridYColor(string majorGridYColor)	None	Set the major Grid Y color for this Viewport
setMajorXGridWidth(floating-point gridLineWidth)	None	Set the major X Grid width for this Viewport
setMajorYGridWidth(floating-point gridLineWidth)	None	Set the major Y Grid width for this Viewport
setMinorGridXColor(string minorGridXColor)	None	Set the minor grid X color for this Viewport
setMinorGridYColor(string minorGridYColor)	None	Set the minor grid Y color for this Viewport
setMinorXGridWidth(floating-point gridLineWidth)	None	Set the minor X Grid width for this Viewport
setMinorYGridWidth(floating-point gridLineWidth)	None	Set the minor Y Grid width for this Viewport

Example 22 reads a data set from a DSS file, plots that data set, and sets the Viewport's background to light gray.

### Example 22: Viewport Objects

```

from hec.script import *           # for Plot class
from hec.hecmath import *        # for DSS class
from hec.hecmath.DisplayUtilities import * # for display objects
theFile = DSS.open("myFile.dss") # open myFile.dss
thePath = "/BASIN/LOC/FLOW/01NOV2002/1HOUR/OBS/"
flowDataSet = theFile.read(thePath) # read a path name

do = createDisplayObject(flowDataSet) # create a display obj
thePlot = Plot.newPlot()           # create a new Plot
thePlot.addDisplayObject(do)       # add the flow display obj
viewport0=thePlot.getViewport(0)   # get the first Viewport
viewport0.setBackground("lightgray") # set the Viewport's
background to lightgray
viewport0.setDrawBackgroundOn()    # tell the Viewport to draw
its background

```

## 7.7.5 Axis Objects

Table 7.5 describes **Axis Object** methods.

**Table 7.5 - Axis Object Methods**

Method	Returns	Description
getActMax()	floating-point	Return the actual maximum value for this Axis
getActMin()	floating-point	Return the actual minimum value for this Axis
getLabel()	string	Return the Axis label
getMajorTic()	floating-point	Return the major tic interval for this Axis
getMax()	floating-point	Return the maximum limit for this Axis
getMin()	floating-point	Return the minimum limit for this Axis
getMinorTic()	floating-point	Return the minor tic interval for this Axis
getNumberOfTicLabelLevels()	integer	Return the number of tic label layers
getNumTicLabelLevels()	integer	Return the number of tic label levels for this Axis

Method	Returns	Description
getReversed()	0/1	Returns if the Axis is reversed
getScaledLabel()	String	Return the label with scientific notation
getTicColor()	Color	Return the tic color
getTicColorString()	String	Return the Tic color as a String
getTicTextColor()	Color	Return the tic text color
getTicTextColorString()	String	Return the tic text color as a String
isComputingMajorTics()	0/1	Return if major tics are to be computed
isComputingMinorTics()	0/1	Return if minor tics are to be computed
isUsingAutomaticMaximum()	0/1	Return whether the Axis is using the automatic maximum value
isUsingAutomaticMinimum()	0/1	Return whether the Axis is using the automatic minimum value
isUsingAutomaticViewMaximum()	0/1	Return if view maximum is set to automatic
isUsingAutomaticViewMinimum()	0/1	Return if view minimum is set to automatic
isUsingDefaultLimits()	0/1	Return whether this Axis is using it's defaults limits
setActualMaximumValue(floating-point value)	None	Set the actual maximum value for this Axis to value
setActualMinimumValue(floating-point value)	None	Set the actual minimum value for this Axis to value
setAutomaticMaximumOff()	None	Set the Axis to use your supplied maximum value
setAutomaticMaximumOn()	None	Set the Axis to use the maximum value from it's DataSets
setAutomaticMinimumOff()	None	Set the Axis to use your supplied minimum value
setAutomaticMinimumOn()	None	Set the Axis to use the minimum value from it's DataSets
setAutomaticViewMaximumOff()	None	Set automatic view of maximum to off

Method	Returns	Description
setAutomaticViewMaximumOn()	None	Set automatic view of maximum to on
setAutomaticViewMinimumOff()	None	Set automatic view of minimum to off
setAutomaticViewMinimumOn()	None	Set automatic view of minimum to on
setComputeMajorTicsOff()	None	Set so that major tics are not computed
setComputeMajorTicsOn()	None	Set so that major tics are computed
setComputeMinorTicsOff()	None	Set so that minor tics are not computed
setComputeMinorTicsOn()	None	Set so that minor tics are computed
setLabel(string label)	None	Set the label of this Axis
setMajorTicInterval(floating-point interval)	None	Set the major tic interval for this Axis to interval
setMaximumLimit(floating-point max)	None	Set the maximum limit for this Axis to max
setMinimumLimit(floating-point min)	None	Set the minimum limit for this Axis to min
setMinorTicInterval(floating-point interval)	None	Set the minor tic interval for this Axis to interval
setNumberOfTicLabelLayers(integer layers)	None	Set the number of tic label layers to layers max number of tic label layers. -1 is unrestricted. Most important for time series axis.
setReversedOff()	None	Set that the Axis is not to be reversed
setReversedOn()	None	Set that the Axis is to be reversed
setTicColor(String colorString)	None	Set the tic color to the color represented by colorString
setTicTextColor(String colorString)	None	Set the tic text color to the color represented by colorString
setDefaultLimitsOff()	None	Set this Axis not to use defaults limits
setDefaultLimitsOn()	None	Set this Axis to use defaults limits

Method	Returns	Description
zoomByFactor(floating-point factor)	None	Change the zoom scaling by the given factor
zoomIn(floating-point wmin, floating-point wmax)	None	Zooms based on world coordinates

Example 23 reads a data set from a DSS file, adds that data set to a new Plot, and zooms in on the Y Axis.

### Example 23: Using Axis Objects

```

from hec.script import *                # for Plot class
from hec.hecmath import *              # for DSS class

from hec.hecmath.DisplayUtilities import * # for display objects
thePlot = Plot.newPlot()               # create a Plot
dssFile = DSS.open("J:/apps/forecast.dss") # open the DSS file
flow = dssFile.read("/BASIN/LOC/FLOW/01NOV2002/1HOUR/OBS/") # read a data set

thePlot.addDisplayObject(createDisplayObject(flow))

thePlot.showPlot()                     # add the data set
viewport0 = thePlot.getViewport(0)     # show the plot
yaxis = viewport0.getAxis("Y1")        # get the first Viewport
yaxis.zoomByFactor(".5")               # get the Y1 axis
                                        # zoom in

```

## 7.7.6 Axis Tics Objects

Table 7.6 describes **Axis Tics** object methods.

**Table 7.6 - Axis Tics Object Methods**

<b>Method</b>	<b>Returns</b>	<b>Description</b>
areMajorTicLabelsDrawn()	0/1	Return whether the major tic labels are drawn
areMajorTicsDrawn()	0/1	Return whether the major tics are drawn
areMinorTicLabelsDrawn()	0/1	Return whether the minor tic labels are drawn
areMinorTicsDrawn()	0/1	Return whether the minor tics are drawn
editProperties()	None	Display the Edit Properties Dialog for the AxisTics
getAxis()	Axis	Returns a reference to the axis that this object draws
getAxisTicColor()	Color	Return the tic color
getAxisTicColorString()	String	Return the tic color as a String
getFontSizes()	tuple of 3 integers	Return the regular, tiny, min and max font sizes for this AxisTics
getMajorTicLength()	integer	Return the major tic length
getMinorTicLength()	integer	Return the minor tic length
setAxisTicColor(string colorString)	None	Set the tic color to the color represented by colorString
setDrawMajorTicLabelsOn()	None	Set the major tic labels not to draw
setDrawMajorTicsOff()	None	Set the major tics not to draw
setDrawMajorTicsOn()	None	Set the major tics to draw
setDrawMinorTicLabelsOff()	None	Set the minor tic labels not to draw
setDrawMinorTicLabelsOn()	None	Set the minor tic labels to draw
setDrawMinorTicsOff()	None	Set the minor tics not to draw
setDrawMinorTicsOn()	None	Set the minor tics to draw
SetFontSizes(integer sz, integer tiny, integer min, integer max)	None	Set the regular, tiny, min and max font sizes for this AxisTics

Method	Returns	Description
setMajorTicLength(int ticLength)	None	Set the major tic length
setMinorTicLength(int ticLength)	None	Set the minor tic length

Example 24 creates a new Plot with a data set read from DSS and tells the data set's axis tics to draw its minor tic marks.

#### Example 24: Using Axis Tics Objects

```

from hec.script import *           # for Plot class
from hec.hecmath import *        # for DSS class

from hec.hecmath.DisplayUtilities import * # for display objects
thePlot = Plot.newPlot()         # create the Plot
dssFile = DSS.open("J:/apps/forecast.dss") # open the DSS file
flow = dssFile.read("/BASIN/LOC/FLOW/01NOV2002/1HOUR/OBS/") # read the data set

thePlot.addDisplayObject(createDisplayObject(flow))

                                # add the data set
thePlot.showPlot()              # show the plot
viewport0 = thePlot.getViewport(flow) # get the viewport for the
                                # flow data set
yaxistics = viewport0.getAxisTics("Y1") # get the axis tics for the
                                # Viewport
yaxistics.setDrawMinorTicsOn()   # tell axis tics to draw tics

```

### 7.7.7 G2dLine Objects

Table 7.7 describes **G2dLine** object methods.

Table 7.7 - G2dLine Object Methods

Method	Returns	Description
AreSymbolsDrawn()	0/1	Return whether this line draws its symbols
editLineProperties()	None	Method that allows the editing of line properties. This method displays a visible dialog for line editing.
getAutoSkipSymbols()	0/1	Return whether the line auto skips its symbols
getFillColor()	Color	Return the fill color for this line

<b>Method</b>	<b>Returns</b>	<b>Description</b>
getFillColorString()	string	Return the fill color for this line as a String
getFillPattern()	integer	Return the fill pattern for this line
GetFillPatternString()	string	Return the fill pattern for this line as a String
getFillType()	integer	Return the Fill type for this line.
getFillTypeString()	string	Return the Fill type for this line as a String.
getLineColor()	Color	Return the line color for this line
getLineColorString()	string	Return the line color for this line as a String
getLineStyleStyle()	integer	Return the line step style for this line
getLineStyleStyleString()	string	Return the line step style for this line as a String
getLineStyleString()	string	Return the line style for this line as a string
getLineWidth()	floating-point	Return the Line Width of the line
getNumPoints()	integer	Returns the Number of Points that this line has
getSymbolFillColor()	Color	Return the symbol fill color for this line's symbols
getSymbolFillColorString()	string	Return the symbol fill color for this line's symbols as a String
getSymbolLineColor()	Color	Return the symbol line color for this line's symbols
getSymbolLineColorString()	string	Return the symbol line color for this line's symbols as a String
getSymbolOffset()	integer	Return the symbol offset for this line
getSymbolSize()	floating-point	Return the symbol size for this line
getSymbolsSkipInterval()	integer	Return the symbol skip interval for this line

<b>Method</b>	<b>Returns</b>	<b>Description</b>
getSymbolType()	integer	Return the symbol type for this line
isLineDrawn()	0/1	Return this line is drawn
setAutoSkipSymbolsOff()	None	Set the line to not auto skip its symbols
setAutoSkipSymbolsOn()	None	Set the line to auto skip its symbols
setDrawLineOff()	None	Set this line not to draw
setDrawLineOn()	None	Set this line to draw
setDrawSymbolsOff()	None	Set this line not to draw its symbols
setDrawSymbolsOn()	None	Set this line to draw its symbols
setFillColor(string fillColor)	None	Set the fill color for this line
setFillPattern(string fillPattern)	None	Set the fill pattern for this line
setFillType(string fillType)	None	Set the Fill type for this line
setLineColor(string lineColor)	None	Set the line color for this line
setLineStyle(string stepStyle)	None	Set the line step style for this line
setLineStyle(string style)	None	Set the line style for this line
setLineWidth(floating-point width)	None	Set the width for this line
setSymbolFillColor(string symbolFillColor)	None	Set the symbol fill color for this line's symbols
setSymbolLineColor(string symbolLineColor)	None	Set the symbol line color for this line's symbols
setSymbolOffset(integer offset)	None	Set the symbol offset for this line
setSymbolSize(floating-point size)	None	Set the symbol size for this line
setSymbolsSkipInterval(integer skipInterval)	None	Set the symbol skip interval for this line
setSymbolType(integer type)	None	Set the symbol type for this line

Example 25 creates a plot with a data set read from DSS, then tells that data set's curve to draw its symbols auto skipped.

#### Example 25: Using G2dLine Objects

```

from hec.script import *           # for Plot class
from hec.hecmath import *         # for DSS class
from hec.hecmath.DisplayUtilities import * # for display objects
thePlot = Plot.newPlot()          # create the Plot
file = "j:/apps/forecast.dss";
dssfile = DSS.open(file)          # open the file
stage = dssfile.read("/BASIN/LOC/STAGE/01NOV2002/1HOURL/OBS/"); # read the data set
thePlot.addDisplayObject(createDisplayObject(stage))
                                   # add the data set to the
                                   plot
thePlot.showPlot()                # show the plot
stageCurve = thePlot.getCurve(stage) # get the stage curve
stageCurve.setAutoSkipSymbolsOn()  # turn on symbols auto skip

```

## 7.7.8 G2dLabel and AxisLabel Objects

Table 7.8 describes **G2dLabel** and **AxisLabel** object methods.

Table 7.8 - Label Object Methods

Method	Returns	Description
editProperties()	None	Display the Edit Properties Dialog for the label
getAlignment()	integer	Return the text alignment for this label
getAlignmentString()	string	Return the text alignment for this label as a String
getBackground()	Color	Return the background color for the label
getBackgroundString()	string	Return the background color for the label as a String
getBorderStyleString()	string	Return the border style for this label as a String
getBorderWeight()	floating-point	Return the border weight for this label
getFillPattern()	integer	Return the background fill pattern for this label
getFillPatternString()	string	Return the background fill pattern for this label as a String

Method	Returns	Description
getFontFamily()	string	Return the font family for the label
getFontSize()	integer	Return the font size for the label
getFontSizes()	integer[]	Return the regular, tiny, min and max font sizes for this label
getFontStyle()	integer	Return the font style for the label
getFontStyleString()	string	Return the font style for the label as a String
getForeground()	Color	Return the foreground color for the label
getForegroundString()	string	Return the foreground color for the label as a String
getIcon()	Icon	Return the Icon to display for this label
getIconPath()	string	Return the Icon path to display for this label
getRotation()	integer	Return the text rotation for this label
getSpacing()	integer	Return the spacing around this label
getText(String txt)	string	Return the text for the label
isBackgroundDrawn()	0/1	Return whether the background is drawn
isBorderDrawn()	0/1	Return whether the border is drawn
setAlignment(string alignment)	None	Set the text alignment for this label
setBackground(string colorString)	None	Set the background color for the label
setBorderColor(string colorString)	None	Set the border color for this label
setBorderStyle(string style)	None	Set the border style for this label
setBorderWeight(floating-point weight)	None	Set the border weight for this label
setDrawBackgroundOff()	None	Set the background not to draw
setDrawBackgroundOn()	None	Set the background to draw
setDrawBorderOff()	None	Set the border not to draw
setDrawBorderOn()	None	Set the border to draw
setFillPattern(string pattern)	None	Set the background fill pattern for this label
setFontFamily(string fam)	None	Set the font family for the label
setFontSize(integer sz)	None	Set the font size for the label

Method	Returns	Description
setFontSizes(integer sz, integer tiny, integer min, integer max)	None	Set the regular, tiny, min and max font sizes for this label
setFontStyle(integer style)	None	Set the font style for the label
setFontStyle(string style)	None	Set the font style for the label
setForeground(string colorString)	None	Set the foreground color for the label
setIcon(Icon icon)	None	Set the Icon to display for this label
setIcon(string iconPath)	None	Set the Icon to display for this label
setRotation(integer rotation)	None	Set the text rotation for this label
setSpacing(integer space)	None	Set the spacing around this label
setText(string text)	None	Set the text for the label

Example 26 creates a plot from a DSS data set and sets the Y1 axis label text to blue.

#### Example 26: Using Axis Label Objects

```

from hec.script import *           # for Plot class
from hec.hecmath import *         # for DSS class
from hec.hecmath.DisplayUtilities import * # for display objects
thePlot = Plot.newPlot()         # create the plot
dssFile = DSS.open("J:/apps/forecast.dss") # open the DSS file
flow = dssFile.read("/BASIN/LOC/FLOW/01NOV2002/1HOUR/OBS/") # read the data set
thePlot.addDisplayObject(createDisplayObject(flow))

                                # add the data set to the
                                plot
thePlot.showPlot()              # show the plot
viewport0 = thePlot.getViewport(0) # get the first viewport
ylabel = viewport0.getAxisLabel("Y1") # get the Y1 axis label
ylabel.setForeground("blue")      # set the Y1 axis label text
                                to blue

```

## 7.7.9 G2dTitle Objects

**G2dTitle objects** represent the text on the Plot above the first Viewport. They have all the methods of G2dLabels plus a few more.

Table 7.9 describes **G2dTitle** object methods.

**Table 7.9 - G2dTitle Object Methods**

Method	Returns	Description
isTitleDrawn()	0/1	Return whether the title is drawn
setDrawTitleOff()	None	Set the title not to draw
setDrawTitleOn()	None	Set the title to draw

Example 27 creates a plot from a DSS data set, sets the Plot's title to "Axema Stage", and has it draw.

### Example 27: Using G2dTitle Objects

```

from hec.script import *           # for Plot class
from hec.hecmath import *        # for DSS class
from hec.hecmath.DisplayUtilities import * # for display objects

thePlot = Plot.newPlot()         # create a new Plot
file = "j:/apps/forecast.dss"    # define the DSS file
dssfile = DSS.open(file)        # open the DSS file
stage = dssfile.read("//AXEMA/STAGE/01OCT2001/1HOUR/OBS/"); # read the data set

thePlot.addDisplayObject(createDisplayObject(stage))

                                # add the data set to the
                                plot
thePlot.showPlot()              # show the plot
title=thePlot.getPlotTitle()    # get the plots title
title.setText("Axema Stage")    # set the plot's title's text to
                                "Axema Stage"
title.setDrawTitleOn()          # tell the title to draw

```

## 7.7.10 Templates

Template files saved interactively from HEC-DSSVue may be applied to plots via scripting. When saving a template interactively from the plot window via the “Save Template...” entry on the “File” menu, HEC-DSSVue:

1. Chooses the “My Documents” subdirectory of the directory specified in the USERPROFILE environment variable as the default location for the template file.
2. Appends “.template” to the end of the specified file name.

The applyTemplate(string filename) G2dDialog method requires the actual file name for the template file. To apply a template saved in the default directory, the complete template file name must be re-created as demonstrated in Example 28.

### Example 28: Applying Template Saved in Default Directory

```
import os                                # for getenv() & sep
from hec.script import *                 # for Plot class
from hec.hecmath import *               # for DSS class
from hec.hecmath.DisplayUtilities import * # for display objects
thePlot = Plot.newPlot()                # create a new Plot
file = "j:/apps/forecast.dss";
dssfile = DSS.open(file)                # open the DSS file
stage = dssfile.read("//AXEMA/STAGE/01OCT2001/1HOUR/OBS/");
                                        # read the data set
thePlot.addDisplayObject(createDisplayObject(stage))
                                        # add the data set
thePlot.showPlot()                      # show the plot
templateName = "myTemplate"             # template base name
templateFileName =                      # re-create the file name
  os.getenv("userprofile")              \
  + os.sep                               \
  + "My Documents"                      \
  + os.sep                               \
  + templateName                        \
  + ".template"
thePlot.applyTemplate(templateFileName) # apply the template
```

## 7.8 Plot Component Properties

The following tables are the valid values to be used when calling plot related functions that take a color (`setBackground(string color)`, etc...), an alignment (`setAlignment()`), a rotation (`setRotation()`), a fill pattern (`setFillPattern()`), a fill type (`setFillType()`), a line style (`setLineStyle()`) or a step style (`setLineStepStyle()`).

### 7.8.1 Colors

Colors can be specified either by a String or by a `java.awt.Color` object. If setting a color through the use of a String object the String can either be a standard color name (i.e. `darkred`) or an RGB string (i.e. `255,20,20`). Table 7.10 lists standard color names.

**Table 7.10 Standard Color Names**

black	darkmagenta	green	lightorange	orange
blue	darkorange	lightblue	lightpink	pink
cyan	darkpink	lightcyan	lightpurple	purple
darkblue	darkpurple	lightgray	lightred	red
darkcyan	darkred	lightgreen	lightyellow	white
darkgray	darkyellow	lightmagenta	magenta	yellow
darkgreen	gray			

### 7.8.2 Alignment

Table 7.11 lists supported alignments.

**Table 7.11 Alignment Values**

Left	Center	Right
------	--------	-------

### 7.8.3 Rotation

Table 7.12 lists supported rotation values.

**Table 7.12 Rotation Values**

0	90	180	270
---	----	-----	-----

## 7.8.4 Fill Patterns

Table 7.13 lists supported fill patterns.

**Table 7.13 Fill Patterns**

Solid	Horizontal	Vertical
Cross	FDiagonal (Forward)	BDiagonal (Back)
Diagonal Cross		

## 7.8.5 Fill Style

Table 7.14 lists supported fill values.

**Table 7.14 Fill Values**

None	above	below
------	-------	-------

## 7.8.6 Line Styles

Table 7.15 lists supported step style values.

**Table 7.15 Line Style Values**

Solid	Dash	Dot
Dash Dot	Dash Dot-Dot	

## 7.8.7 Step Style

Table 7.16 lists supported step style values.

**Table 7.16 Step Style Values**

Normal	step	cubic
--------	------	-------

## 7.9 Tables

Tables allow you to view data in a vertical scrolling window that shows the ordinates, the dates and times and the values for the selected data sets.

### 7.9.1 Tabulate Class

```
Tabulate.newTable()
Tabulate.newTable(string title)
```

The Tabulate class in the hec.script module is used to create a new Table dialog. It contains two functions to create a Table dialog, each of which returns as a TableFrame object.

Example 29 illustrates creation of a table.

#### Example 29: Creating a Table

```
from hec.script import *
myTable = Tabulate.newTable()

or

from hec.script import *
myTable = Tabulate.newTable("Elevation vs Flow")
```

### 7.9.2 TableFrame Objects

Table 7.17 describes **TableFrame** object methods.

Table 7.17 - TableFrame Object Methods

Method	Returns	Description
addDisplayObject(DisplayObject dspObj)	integer	Adds Data Set to the table.
getCommaState()	0/1	Get whether the commas are shown
getDateAsTwoColumnsState()	0/1	Get whether date/time columns are shown as 1 or 2 columns in the table
hide()	None	Hide the table
print()	None	Display the print dialog
setCommaState(0/1 showCommas)	None	Set state to show commas or not
setDateAsTwoColumnsState (Integer showDateAs2Column)	None	Set whether date/time columns should show as 1 or 2 columns in the table
ShowTable()	None	Show the table

Example 30 creates a table from two DSS data sets and display the print dialog.

**Example 30: Filling and Displaying a Table**

```
from hec.hecmath import *           # for DSS
from hec.script import *           # for Tabulate
from hec.hecmath.DisplayUtilities import * # for display objects

file = "j:/apps/forecast.dss"      # specify the DSS file
dssfile = DSS.open(file)          # open the file
# read 2 paths
stage = dssfile.read("//AXEMA/STAGE/01OCT2001/1HOUR/OBS/")
flow = dssfile.read("//AXEMA/FLOW/01OCT2001/1HOUR/OBS/")
theTable = Tabulate.newTable()     # create the table
theTable.setTitle("Test Table")     # set the table title
theTable.setLocation(50,50)        # set the location of the table on
                                   # the screen

flowObj = createDisplayObject(flow) # create display objects
stageObj = createDisplayObject(stage)
theTable.addDisplayObject(flowObj) # add the display objects
theTable.addDisplayObject(stageObj)
theTable.showTable()               # show the table
theTable.print()                   # print the table
theTable.hide()                    # hide the table
```

## 7.10 Math Functions

Math functions are accessible through the general class called **HecMath**. **HecMath** objects hold data sets and allow you to perform mathematical operations on them. They can also be passed to plots and tables to display the data. A HecMath object is either a TimeSeriesMath object or a PairedDataMath object, which handle time series and paired data sets, respectively.

Before using PairedDataMath methods, be sure to read the description for the setCurve Method. Paired data sets may contain multiple curves. The setCurve method provides user control over which paired data curve is operated upon by subsequent function calls.

### 7.10.1 Absolute Value

`abs()`

Derive a new time series or paired data set from the absolute value of values of the current data set. For time series data, missing values are kept as missing. For paired data sets, use the setCurve method to first select the paired data curve(s).

**See also:** setCurve().

**Parameters:** Takes no parameters.

**Example:** `NewDataSet = dataSet.abs()`

**Returns:** A new HecMath object of the same type as the current object.

### 7.10.2 Accumulation (Running)

`accumulation()`

Derive a new time series by computing a running accumulation of the current time series.

For time points in which the current time series value are missing, the value in the accumulation time series remains constant (same as the accumulated value at the last valid point location).

**Parameters:** Takes no parameters.

**Example:** `NewTimeSeries = timeSeries.accumulation()`

**Returns:** A new TimeSeriesMath object.

### 7.10.3 Add a Constant

```
add(floating-point constant)
```

Add the value **constant** to all valid values in the current time series or paired data set. For time series data, missing values are kept as missing.

For paired data, **constant** is added to y-values only. Use the `setCurve` method to first select the paired data curve(s).

**See also:** `add(HecMath dataSet)`

```
setCurve().
```

**Parameters:** `constant` - A floating-point value.

**Example:** `newDataSet = dataSet.add(2.5)`

**Returns:** A new HecMath object of the same type as the current object.

### 7.10.4 Add a Data Set

```
add(TimeSeriesMath tsData)
```

Add the values in the data set **tsData** to the values in the current data set. The function only applies to time series data sets.

When adding one time series data set to another, there is no restriction that times in the two data sets match exactly. However, only values with coincident times will be added. Times in the current time series data set that cannot be matched with times in the second data set are set to missing. Values in the current data set that are missing are kept as missing. Either or both data sets may be regular or irregular interval time series.

This function will not merge data sets. Use the `mergeTimeSeries` (for time series data sets) or the `mergePairedData` (for paired data sets) functions for this purpose.

**See also:** `add(floating-point constant)`

```
mergeTimeSeries(TimeSeriesMath)
```

```
mergePairedData(PairedDataMath)
```

**Parameters:** `tsData` - A TimeSeriesMath object.

**Example:** `newTsData = tsData.add(otherTsData)`

**Returns:** A new TimeSeriesMath object.

### 7.10.5 Apply Multiple Linear Regression Equation

```
applyMultipleLinearRegression(string startTimeString,
                              string endTimeString,
                              sequence tsDataSetSequence,
                              floating-point minimumLimit,
                              floating-point maximumLimit)
```

Apply the regression coefficients contained in the current paired data set to the array of time series data sets in **tsDataSetSequence** to develop a new time series data set. The `applyMultipleLinearRegression` function applies the multiple linear regression coefficients computed with the `multipleLinearRegression` function (see section 7.10.45).

For the general linear regression equation, a dependent variable, Y, may be computed from a set independent variables, Xn:

$$Y = B_0 + B_1 * X_1 + B_2 * X_2 + B_3 * X_3$$

where  $B_n$  are linear regression coefficients.

For time series data sets, an estimate of the original time series data set values may be computed from a set of independent time series data sets using regression coefficients such that:

$$TsEstimate(t) = B_0 + B_1 * TS_1(t) + B_2 * TS_2(t) + \dots + B_n * TS_n(t)$$

where  $B_n$  are the set of regression coefficients and  $TS_n$  are the time series data sets contained in **tsDataSetSequence**.

The number of regression coefficients in the current `PairedDataMath` object must be one more than the number of independent time series data sets in **tsDataSetSequence**. The collection of selected time series data sets must be in the same order as when the regression coefficients were computed with the `multipleLinearRegression` method.

All the time series data sets must be regular interval and have the same time interval. The function filters the data to determine the time period common to all time series data sets and uses only those points in the regression analysis. For any given time, if a value is missing in any time series, the value in resultant time series is set to missing.

The parameters **minimumLimit** and **maximumLimit** can be used to specify the range of valid values for the resultant data set. Values which fall outside the specified range are set to missing. **minimumLimit** or **maximumLimit** may be entered as **Constants.UNDEFINED** to ignore the minimum or maximum value check.

If **startTimeString** or **endTimeString** are blank strings, the start and end time of the resultant time series will be defined by the time period common to all time series data sets in **tsDataSetSequence**. Otherwise the time series start

and end may be defined using **startTimeString** and **endTimeString** which have the usual HEC time window format (e.g. "01JAN2001 1400").

Names, parameter type and unit labels for the new time series data set are copied over from the first time series data set in **tsDataSetSequence**. The F part in the new data set is set to "COMPUTED."

**Parameters:**

**startTimeString** – A string containing an HEC time (e.g. "01JAN2001 1400") specifying the start time of the resultant time series data set. May be blank ("").

**endTimeString** – A string containing an HEC time (e.g. "01JAN2001 1400") specifying the ending time of the resultant time series data set. May be blank ("").

**tsDataSetSequence** – Sequence of TimeSeriesMath objects. Must all be regular interval and have the same time interval.

**minimumLimit** – A floating-point value specifying the minimum valid value in the resultant time series data set. Set to Constants.UNDEFINED to ignore this option.

**maximumLimit** – A floating-point value specifying the maximum valid value in the resultant time series data set. Set to Constants.UNDEFINED to ignore this option.

**Example:**

```
newTsData =
pairedData.applyMultipleLinearRegression(
    "01Jan2000 0000",
    "31Dec2000 2300",
    (tsData1, tsData2, tsData3),
    Constants.UNDEFINED,
    Constants.UNDEFINED)
```

**Returns:** A new regular interval TimeSeriesMath object.

**Generated Exceptions:** Throws an `hec.hecmath.HecMathException` if the number of data sets in **tsDataSetSequence** is not equal to the number of regression coefficients -1, or if the data sets in **tsDataSetSequence** are not regular interval time series data sets with the same interval time.

## 7.10.6 Centered Moving Average Smoothing

```
centeredMovingAverage(integer numberToAverageOver,  
    boolean onlyValidValues,  
    boolean useReduced)
```

Derive a new time series from the centered moving average of **numberToAverageOver** values in the current time series. **numberToAverageOver** must be an odd integer greater than 2.

If **onlyValidValues** is set to true, then if any points in the averaging interval are missing, the point in the new time series is set to missing. If **onlyValidValues** is set to false and missing values are contained in the averaging interval, a smoothed point is still computed using the remaining valid values in the interval. If there are no valid values in the averaging interval, the point is set to missing.

If **useReduced** is set to true, then centered moving average points can still be computed at the beginning and end of the time series, even if there are less than **numberToAverageOver** values in the averaging interval. If **useReduced** is set to false, then the first and last **numberToAverageOver**/2 points of the resultant time series are set to missing.

### Parameters:

**numberToAverageOver** – An integer containing the number of values to average over for computing the centered moving average. Must be odd and greater than 2.

**onlyValidValues** – Either Constants.TRUE, or Constants.FALSE, specifying whether all values in the averaging interval must be valid for the computed point in the new time series to be valid.

**useReduced** – Either Constants.TRUE, or Constants.FALSE, specifying whether to allow points at the beginning and end of the resultant time series to be computed from a reduced ( less than **numberToAverageOver** ) set of points.

### Example:

```
avgData = tsData.centeredMovingAverage(  
    5,  
    Constants.TRUE,  
    Constants.TRUE)
```

**Returns:** A new TimeSeriesMath object.

**Generated Exceptions:** Throws an `hec.hecmath.HecMathException` if the **numberToAverageOver** is less than 3 or not odd.

## 7.10.7 Conic Interpolation from Elevation/Area Table

```
conicInterpolation( TimeSeriesMath tsData,
                   string inputType,
                   string outputType,
                   floating-point storageScaleFactor )
```

Use the conic interpolation table in the current paired data set to develop a new time series data set from the interpolation of **tsData**.

The current paired data should be an Elevation-Area table. However, the first data pair are the initial conic depth, and the storage value at the first elevation in the table. If the initial conic depth is undefined, the function will calculate a value. Elevation-Area values in the table must be in ascending order.

**tsData** is either a time series of reservoir elevation or storage. The type is specified by setting **inputType** as "S(TORAGE)" or "E(LEVATION)." The desired output time series type is similarly set using **outputType**. The valid settings for **outputType** are "S(TORAGE)", "E(LEVATION)" or "A(REA)." **inputType** and **outputType** must not be the same.

**storageScaleFactor** is an optional parameter used to scale input (by multiplying) and output (by dividing) storage values. For example, if the area in the conic interpolation table is expressed in sq.ft., **storageScaleFactor** could be set to 43560. to convert the storage output to acre-ft.

Parameter type in the new time series is set according to **outputType**. If the output time series values are elevation, the time series units are set to the paired data x-units label. If the output time series values are area, the time series units are set to the paired data y-units label. If the output is storage, the units are not set and should be set by the user with the `setUnits` function.

**See also:** `setUnits()`.

### Parameters:

**tsData** – A TimeSeriesMath object representing elevation or storage.

**inputType** – A string specifying the parameter type for the input time series, either "S(TORAGE)" or "E(LEVATION)." Only the first character of the string is interpreted by the function.

**outputType** – A string specifying the parameter type for the output time series, either "S(TORAGE)", "E(LEVATION)" or "A(REA)." Only the first character of the string is interpreted by the function.

**storageScaleFactor** – A floating-point number used to scale input (by multiplying) and output (by dividing) storage values.

**Examples:**

```
tsStorage =
    conicElevAreaCurve.conicInterpolation(
        tsElev,
        "Elevation",
        "Storage",
        1.0)

tsArea =
    conicElevAreaCurve.conicInterpolation(
        tsElev,
        "Elevation",
        "Area",
        1.0)
```

**Returns:** A new TimeSeriesMath object.

**Generated Exceptions:** Throws an `hec.hecmath.HecMathException` if **inputType** or **outputType** cannot be interpreted as one of the allowed values; if **inputType** and **outputType** are the same parameters; if values in the conic interpolation table are not in ascending order.

## 7.10.8 Convert Values to English Units

```
convertToEnglishUnits()
```

Perform unit conversion of data values and unit labels in the current time series or paired data set from Metric (SI) units to English units.

Determination of the unit system will be based upon the current units labels and parameter types. If the data units are already in English units or the unit system cannot be determined, no conversion occurs.

For paired data, both x and y values are converted. For time series data, missing values remain missing.

**See also:** `convertToMetricUnits()`, `isEnglish()`, `isMetric()`.

**Parameters:** Takes no parameters

**Example:** `englishDataSet = siDataSet.convertToEnglishUnits()`

**Returns:** A `HecMath` object of the same type as the current object.

## 7.10.9 Convert Values to Metric (SI) Units

### `convertToMetricUnits()`

Perform unit conversion of data values and unit labels in the current time series or paired data set from English units to Metric (SI) units. Determination of the unit system will be based upon the current units labels and parameter types. If the units are already in Metric units or the unit system cannot be determined, no conversion occurs.

For paired data, both x and y values are converted. For time series data, missing values remain missing.

**See also:** `convertToEnglishUnits()`, `isEnglish()`, `isMetric()`.

**Parameters:** Takes no parameters

**Example:** `siDataSet = englishDataSet.convertToMetricUnits()`

**Returns:** An HecMath object of the same type as the current object.

## 7.10.10 Correlation Coefficients

### `correlationCoefficients`(TimeSeriesMath tsData)

Computes the linear regression and other correlation coefficients between data in the current time series and `tsData`. Values in the current time series and `tsData` are matched by time to form data pairs for the correlation analysis. The data sets may be either regular or irregular time interval data.

The correlations statistics computed by the function are:

- Number of valid values
- Regression constant
- Regression coefficient
- Determination coefficient
- Standard error of regression
- Determination coefficient adjusted for degrees of freedom
- Standard error adjusted for degrees of freedom

These values are contained in a `LinearRegressionStatistics` object.

The current `TimeSeriesMath` object forms the values of the independent variable (x-values), while values of the second time series comprise the dependent variable (y-values). The linear regression coefficients thus express how values in the second data set can be derived from values in the primary data set:

$$TS2(t) = a + b * TS1(t)$$

where “a” is the regression constant and “b” the regression coefficient.

**See also:** LinearRegressionStatistics.

**Parameters:** tsData - A TimeSeriesMath object that forms the dependent variable for the regression analysis.

**Example:**

```
linearRegressionData =  
    tsData.correlationCoefficients(otherTsData)
```

**Returns:** A LinearRegressionStatistics object holding the correlation data.

**Generated Exceptions:** Throws an hec.hecmath.HecMathException if the times in the current time series do not exactly match times in tsData.

## 7.10.11 Cosine Trigonometric Function

`cos()`

Derive a new time series or paired data set from the cosine of values of the current data set. The resultant data set values are in radians. For time series data, missing values are kept as missing.

For paired data sets, use the setCurve (see sections 7.10.61 and 7.10.62) function to first select the paired data curve (or all curves) to apply the function. By default the function is applied to all paired data curves.

**See also:** setCurve().

**Parameters:** Takes no parameters

**Returns:** A HecMath object of the same type as the current object.

## 7.10.12 Cyclic Analysis (Time Series)

### `cyclicAnalysis()`

Derive a set of cyclic statistics from the current regular interval time series data set. The time series data set must have a time interval of "1HOUR", "1DAY" or "1MONTH." The function sorts the time series values into statistical "bins" relevant to the time interval. Values for the 1HOUR interval data are sorted into 24 bins representing the hours of the day, 0100 to 2400. The 1DAY interval data is apportioned to 365 bins for the days of the year. The 1MONTH interval data is sorted into 12 bins for the months of the year.

The format of the resultant data sets is as a "pseudo" time series for the year 3000. For example, the cyclic analysis of one month of hourly interval data will produce pseudo time series data sets having 24 hourly values for the day January 1, 3000. If the statistical parameter is the "maximum" value, then the 24 values represent the maximum value occurring at that hour of the day in the current time series. The cyclic analysis of daily interval data will produce pseudo time series data sets having 365 daily values for the year 3000. The cyclic analysis of monthly interval data will result in pseudo time series data sets having 12 monthly values for the year 3000.

Fourteen pseudo time series data sets are derived by the cyclic analysis function for the following statistical parameters:

- Number of values processed for each time interval
- Maximum value
- Time of maximum value
- Minimum value
- Time of minimum value
- Average value
- Probability exceedence percentiles for 5%, 10%, 25%, 50% (median value), 75%, 90%, and 95%
- Standard deviation

The 14 pseudo time series of cyclic statistics are returned by the function as an array of time series data sets. The parameter part of the record path for each time series is modified to indicate the type of the statistical parameter. For a flow record, the parameter "FLOW" would become "FLOW-MAX" for the maximum values statistics, "FLOW-P5" for the 5% percentile statistics, etc.**Parameters:** Takes no parameters

**Example:** `cyclicData = tsData.cyclicAnalysis()`

**Returns:** A sequence of 14 TimeSeriesMath objects, each of which is a pseudo time series data sets representing a statistical parameter.

**Generated Exceptions:** Throws an `hec.hecmath.HecMathException` if the time series is not regular interval or does not have a time interval of "1HOUR", "1DAY", or "1MONTH".

### 7.10.13 Decaying Basin Wetness Parameter

```
decayingBasinWetnessParameter( TimeSeriesMath tsPrecip,
    floating-point decayRate )
```

Compute a time series of decaying basin wetness parameters from the regular interval time series data set of incremental precipitation, **tsPrecip**, by:

$$\text{TSResult}(t) = \text{Rate} * \text{TSResult}(t-1) + \text{TSPrecip}(t)$$

where Rate is **decayRate**, and  $0 < \text{Rate} < 1$ .

The first value of the resultant time series data set, **TSResult(1)**, is set to the first value in the current time series data set. The current time series data set can be the same time series data set as **tsPrecip**. Missing values in the precipitation time series are taken as zero when applying the above equation.

#### Parameters:

**tsPrecip** – A regular interval TimeSeriesMath object representing precipitation.

**decayRate** – a floating-point number in the range  $0 < \text{decayRate} < 1$ .

#### Example:

```
tsWetness =
    tsPrecip.decayingBasinWetnessParameter (
        tsPrecip,
        0.87)
```

**Returns:** A new TimeSeriesMath object.

### 7.10.14 Divide by a Constant

```
divide(floating-point constant)
```

Divide all valid values in the current time series or paired data set by the value **constant**. For time series data, missing values are kept as missing.

For paired data, **constant** divides the y-values only. Use the `setCurve` method to select the paired data curve(s).

**See also:** `divide(TimeSeriesMath tsData); setCurve()`.

#### Parameters:

**constant** - A floating-point value to divide the values in the current data set (cannot be zero).

**Example:** `newDataSet = dataSet.divide(1.1)`

**Returns:** A new HecMath object of the same type as the current object.

## 7.10.15 Divide by a Data Set

`divide`(TimeSeriesMath tsData)

Divide valid values in the current data set by the corresponding values in the data set **tsData**. Both data sets must be time series data sets.

When dividing one time series data set by another, there is no restriction that times in the two data sets match exactly. However, only values with coincident times will be divided. Times in the current time series data set that cannot be matched with times in the second data set are set to missing. Values in the current data set that are missing are kept as missing. If a value in the second data set is zero or missing, the value in the resultant data set is set to missing (divide by zero not allowed). Either or both data sets may be regular or irregular interval time series.

**See also:** `divide`(floating-point constant).

**Parameters:**

`tsData` - A time series data set.

**Example:** `newTsData = tsData.divide(otherTsData)`

**Returns:** A new TimeSeriesMath object.

## 7.10.16 Estimate Values for Missing Precipitation Data

`estimateForMissingPrecipValues`(integer maxMissingAllowed)

Linearly interpolate estimates for missing values in the current regular or irregular interval time series data set. The current data set is expected to be cumulative precipitation and the data must be of type “INST-CUM”. Use the `estimateForMissingValues` method for filling missing values in other types of time series data.

The rules used for interpolation of missing cumulative precipitation data are:

- If the values bracketing the missing period are increasing with time, only interpolate if the number of successive missing values does not exceed the value of **maxMissingAllowed**.
- If the values bracketing the missing period are decreasing with time, do not estimate for any missing values.
- If the values bracketing the missing period are equal, then estimate any number of missing values.

**See also:** `estimateForMissingValues`()

**Parameters:**

**maxMissingAllowed** - An integer value for the maximum number of consecutive missing values between valid values.

**Example:**

```
newPrecip =
    tsPrecip.estimateForMissingPrecipValues(5)
```

**Returns:** A new TimeSeriesMath object.

## 7.10.17 Estimate Values for Missing Data

**estimateForMissingValues**(integer maxMissingAllowed)

Linearly interpolate estimates for missing values in the current regular or irregular interval time series data set. Do not interpolate if the number of successive missing values exceeds **maxMissingAllowed**.

**See also:** estimateForMissingPrecipValues().

**Parameters:**

**maxMissingAllowed** - An integer value for the maximum number of consecutive missing values allowed for interpolation.

**Example:** newTsData = tsData.estimateForMissingValues(5)

**Returns:** A new TimeSeriesMath object.

## 7.10.18 Exponentiation Function

**exponentiation**(floating-point constant)

Derive a new time series or paired data set from the exponentiation of values in the current data set by **constant**, by:

$$T2(i) = T1(i)^{\text{constant}}$$

For time series data, values that are missing in the current time series remain missing in the new time series.

For paired data sets, use the setCurve method to first select the paired data curve(s).

**See also:** setCurve().

**Parameters:**

**constant** – a floating-point value representing the exponent.

**Example:** squaredDataSet = dataSet.exponentiation(2.)

**Returns:** A new HecMath object of the same type as the current object.

## 7.10.19 Extract Time Series Data at Unique Time Specification

```
extractTimeSeriesDataForTimeSpecification (
    string timeLevelString,
    string rangeString,
    boolean isInclusive,
    integer intervalWindow,
    boolean setAsIrregular )
```

Select/extract data points from the current regular or irregular interval time series data set based upon user defined time specifications. For example, the function may be used to extract from hourly interval data, the values observed every day at noon.

`timeLevelString` defines the time level/interval for extraction (year, month, day of the month, day of the week, or 24-hour time). `rangeString` defines the interval range for data extraction applicable to the time level. For example, if `timeLevelString` is "MONTH", a valid range would be "JAN-MAR". The `rangeString` variable can define a single interval value (e.g. "JAN" - select data from January only) or a beginning and ending range (e.g. "JAN-MAR" - select data for January through March). Table 7.18 shows the valid `timeLevelString` and `rangeString` values.

**Table 7.18 - Valid `timeLevelString` and `rangeString` Values**

<b>timeLevelString</b>	<b>rangeString</b>	<b>Example rangeString</b>
"YEAR"	Four-digit year value	"1938" or "1938-1945"
"MONTH"	Standard three-character abbreviation for month	"JAN" or "JAN-MAR" or "OCT-FEB"
"DAYMON(TH)"	Day of the month or "LASTDAY" string	"15" or "1-15" or "27-5" or "16-LASTDAY"
"DAYWEE(K)"	Standard three-character abbreviation for day of the week	"MON" or "SUN-TUE" or "FRI-WED"
"TIME"	Four digit 24-hour military-style clock time	"2400" or "0300-0600" or "2200-0130"

If desired, you may use one of the enumerated string constants to specify **timeLevelString**:

<b>Year</b>	TimeSeriesMath.LEVEL_YEAR_STRING
<b>Month</b>	TimeSeriesMath.LEVEL_MONTH_STRING
<b>Day of Month</b>	TimeSeriesMath.LEVEL_DAYMONTH_STRING
<b>Day of Week</b>	TimeSeriesMath.LEVEL_DAYWEEK_STRING
<b>24-hour time</b>	TimeSeriesMath.LEVEL_TIME_STRING

The parameter `isInclusive` determines whether the data extraction operation is either inclusive or exclusive of the specified range. For example, if `isInclusive` is “true” and the range is set to "JAN-MAR" for the "MONTH" time level, the extracted data will include all data in the months January through March for all the years of time series data. If `isInclusive` is “false” for this example, the extracted data covers the time April through December (is exclusive of the period January through March).

`intervalWindow` is only used when the `timeLevelString` is "TIME.” `intervalWindow` is the minutes before and after the time of day within which the data will be extracted. `intervalWindow` effectively increases the time range at the beginning and end `intervalWindow` minutes. For example, with a `rangeString` of “0300” and an `intervalWindow` of “10 minutes”, data will be extracted from the selected time series if times falls within in the period 0250 to 0310.

`setAsIrregular` defines whether the extracted data is saved as regular interval or irregular interval data. Most often the time series data formed by the extraction process will no longer be regular interval, and `setAsIrregular` should be set to “true.” Setting `setAsIrregular` to “false” will force an attempt to save the data as regular interval time data.

#### Parameters:

**timeLevelString** – A string specifying the time level selection.

**rangeString** – A string specifying time or time range for selection. Must be consistent with `timeLevelString`.

**isInclusive** – Either Constants.TRUE, or Constants.FALSE, value. If true, data is extracted inclusive of the range specified by `rangeString`. If false, data is extracted exclusive of the range specified by `rangeString`.

**intervalWindow** – An integer value representing the minutes before and after the time of day within which the data will be extracted. Only applied when the `timeLevelString` is “TIME.”

**setAsIrregular** – Either Constants.TRUE, or Constants.FALSE, value. If true, data is automatically set as irregular time interval data. If false, the function will attempt to classify the data as regular time interval data.

**Example:**

```
SelectedData =
    tsData.extractTimeSeriesDataForTimeSpecification(
        "DAYMONTH",
        "16-LASTDAY",
        Constants.TRUE,
        0,
        Constants.TRUE)
```

**Returns:** A new TimeSeriesMath object.

**Generated Exceptions:** Throws an hec.hecmath.HecMathException if the function could not successfully interpret timeLevelString or rangeString.

## 7.10.20 Flow Accumulator Gage (Compute Period Average Flows)

**flowAccumulatorGageProcessor**(TimeSeriesMath tsCounts)

Derive a new time series of period-average flows from a flow accumulator type gage. The current time series is assumed to contain the accumulated flow data, while the parameter time series, **tsCounts**, is assumed to have the corresponding time series of counts. The two time series data sets must match times exactly. The two time series are combined to compute a new time series of period average flow:

$$TsNew(t) = ( TsAccFlow(t) - TsAccFlow(t-1) ) / ( TsCount(t) - TsCount(t-1) )$$

where TsAccFlow is the gage accumulated flow time series and TsCount is the gage time series of counts.

In the above equation, if TsAccFlow(t), TsAccFlow(t-1), TsCount(t) or TsCount(t-1) are missing, TsNew(t) is set to missing. The new time series is assigned the data type "PER-AVER".

**Parameters:**

**tsCounts** – A TimeSeriesMath object containing the counts for the flow accumulator gage.

**Example:**

```
tsPerAvgFlow =
    tsAccumFlow.flowAccumulatorGageProcessor(tsCounts)
```

**Returns:** A new TimeSeriesMath object.

**Generated Exceptions:** Throws an `hec.hecmath.HecMathException` if times in the current object do not exactly match the times in **tsCounts**.

### 7.10.21 Forward Moving Average Smoothing

`forwardMovingAverage`(integer numberToAverageOver)

Derive a new time series from the forward moving average of **numberToAverageOver** values in the current time series. **numberToAverageOver** must be an integer greater than 2.

If the averaging interval contains a missing value, the smoothed value is computed from the remaining valid values in the interval. However, if there are less than 2 valid values in the interval, the value in the resultant data set is set to missing.

**Parameters:**

**numberToAverageOver** – An integer containing the number of values to average over for computing the forward moving average.

**Example:** `tsAveraged = tsData.forwardMovingAverage(4)`

**Returns:** A new `TimeSeriesMath` object.

**Generated Exceptions:** Throws an `hec.hecmath.HecMathException` if the `numberToAverageOver` is less than 2.

### 7.10.22 Generate Data Pairs from Two Time Series

`generateDataPairs`(`TimeSeriesMath tsData`,  
boolean sort)

Generate a paired data set by pairing values (by time) from the current time series data set and the time series data set **tsData**. The values of the current time series form the x-ordinates, while values from `tsData` form the y-ordinates of the resulting paired data set. The times in the two time series data sets must match exactly. If a value for a time is missing in either time series, no data value pair is formed or added to the paired data set. If **sort** is “true”, data pairs in the paired data set are sorted by ascending x-value.

The units and parameter type from the current time series data set are assigned to the paired data set x-units and x-parameter type. The units and parameter type from **tsData** are assigned to the paired data set y-units and y-parameter type.

An example application of the function would be to mate a time series record of stage to one of flow to generate a stage-flow paired data set.

**Parameters:**

**tsData** – A TimeSeriesMath object that forms the y-ordinates of the resulting paired data set.

**sort** – Either Constants.TRUE, or Constants.FALSE, value. If true, sort data pairs in ascending x-value. If false, leave unsorted.

**Example:** `ratingCurve = tsStage.generateDataPairs(tsFlow)`

**Returns:** A PairedDataMath object with x-ordinates from the current time series, and y-ordinates from **tsData**.

**Generated Exceptions:** Throws an `hec.hecmath.HecMathException` if times from the current time series and **tsData** do not match exactly.

### 7.10.23 Generate a Regular Interval Time Series

```
generateRegularIntervalTimeSeries(string startTimeString,
    string endTimeString,
    string timeIntervalString,
    string timeOffsetString,
    floating-point initialValue )
```

Generate a new regular interval time series data set from scratch with times and values specified by the parameters. This is a function provided by the TimeSeriesMath module, and not an object method.

The parameters **startTimeString** and **endTimeString** are strings used to specify the beginning and ending time of the generated data set. These two parameters have the form of the standard HEC time string (e.g. "01JAN2001 0100").

The regular time interval is specified by **timeIntervalString**, and is a valid HEC time increment string (e.g. "1MIN", "15MIN", "1HOUR", "6HOUR", "1DAY", "1MONTH").

**timeOffsetString** is used to shift times in the resultant time series from the standard interval time. As an example, the offset could be used to shift times in regular hourly interval data from the top of the hour to 6 minutes past the hour. The parameter has the form "nT", where "n" is an integer number and "T" is one of the time increments: "M(INUTES)", "D(AYS)", "H(OUR)", "W(EEKS)", "MON(THS)" or "Y(EARS)" ( characters in the parenthesis are optional ). For example, a time offset of 9 minutes would be expressed as "9M" or "9MIN."

Values in the time series data set are initialized to **initialValue**. **Parameters:**

**startTimeString** - a string specifying a standard HEC time defining the time series data start date/time.

**endTimeString** - a string specifying a standard HEC time defining the time series data end date/time.

**timeIntervalString** - a string specifying a valid DSS regular time interval which defines the time interval of the new time series.

**timeOffsetString** – a string specifying the offset of the new time points from the regular interval time. This string may be an empty string or None.

**initialValue** –a floating-point number set to the initial value for all time series points. Set to HecMath.UNDEFINED to set all values to missing.

**Example:**

```
newTsData =
    TimeSeriesMath.generateRegularIntervalTimeSeries(
        "01FEB2002 0100",
        "28FEB2002 2400",
        "1HOUR",
        "0M",
        100.)
```

**Returns:** A new regular interval TimeSeriesMath object initialized to initialValue. Data units and type are unset.

**Generated Exceptions:** Throws an hec.hecmath.HecMathException if time parameters cannot be successfully interpreted.

## 7.10.24 Get Data Container

**getData()**

Returns a copy of the hec.io.DataContainer for the current data set. For time series data sets, returns a hec.io.TimeSeriesContainer. For paired data sets, returns a hec.io.PairedDataContainer.

The hec.io.TimeSeriesContainer contains the time series values for a time series data set. The hec.io.PairedDataContainer contains the paired data values for a paired data set.

**Parameters:** Takes no parameters

**Example:** `container = dataset.getData()`

**Returns:** A hec.io.DataContainer.

### 7.10.25 Get Data Type for Time Series Data Set

```
getType()
```

Get the data type for a time series data set.

**Parameters:** Takes no parameters

**Example:** `dataSet.getType()`

**Returns:** A string - “INST-CUM”, “INST-VAL”, “PER-AVER” or “PER-CUM”.

### 7.10.26 Get Units Label for Data Set

```
getUnits()
```

Get the units label of the current data set. For a paired data set, returns the y-units label.

**Parameters:** Takes no parameters

**Example:** `dataSet.getUnits()`

**Returns:** A string.

### 7.10.27 Interpolate Time Series Data at Regular Intervals

```
interpolateDataAtRegularInterval(string timeIntervalString,  
                                string timeOffsetString)
```

Derive a regular interval time series data set by interpolation of the current regular or irregular interval time series data set.

The new time interval is set by **timeIntervalString** which must be a valid HEC time interval string (e.g. “1MIN”, “15MIN”, “1HOUR”, “6HOUR”, “1DAY”, “1MONTH”).

Times in the resultant time series may be shifted (offset) from the regular interval time by the increment specified by `timeOffsetString`. As an example, the offset could be used to shift times from the top of the hour to 6 minutes past the hour. If no offset is used `timeOffsetString` should be an blank or empty string.

Whether the time series data type is “INST-VAL”, “INST-CUM”, “PER-AVE”, or “PER-CUM” controls how the interpolation is performed. Interpolated values are derived from “INST-VAL” or “INST-CUM” data using linear interpolation. Values are derived from “PER-AVE” data by computing the period average value over the time interval. Values are derived from “PER-CUM” data by computing the period cumulative value over the new time interval

For example, if the original data set is hourly data and the new regular interval data set is to have a six hour time interval:

- The value for “INST-VAL” or “INST-CUM” type data is computed from the linear interpolation of the hourly points bracketing the new six hour time point.
- The value for “PER-AVE” type data is computed from the period average value over the six hour interval.
- The value for “PER-CUM” type data is computed from the accumulated value over the six hour interval.

The treatment of missing value data is also dependent upon data type. Interpolated “INST-VAL” or “INST-CUM” points must be bracketed or coincident with valid (not missing) values in the original time series; otherwise the interpolated values are set as missing. Interpolated “PER-AVE” or “PER-CUM” data must contain all valid values over the interpolation interval; otherwise the interpolated value is set as missing.

**Parameters:**

**timeIntervalString** – A string specifying the regular time interval for the resultant time series.

**timeOffsetString** – A string specifying the offset of the new time points from the regular interval time. This variable may be an empty string (“ ”).

**Example:**

```
newTsData =  
    tsData.interpolateDataAtRegularInterval(  
        "15MIN",  
        " ")
```

**Returns:** A new regular interval TimeSeriesMath object.

## 7.10.28 Inverse ( 1/X ) Function

`inverse()`

Derive a new time series or paired data set from the inverse (1/x) of values of the current data set. The inverse value is computed by 1.0 divided by the value of the current data set. If a data value is equal to 0.0, the value in the resultant data set is set to missing. For time series data, if the original value is missing, the value remains missing in the resultant data set.

For paired data sets, use the `setCurve` method to first select the paired data curve(s).

**See also:** `setCurve()`.

**Parameters:** Takes no parameters

**Example:** `newDataSet = dataSet.inverse()`

**Returns:** A `HecMath` object of the same type as the current object.

## 7.10.29 Determine if Data Is in English Units

`isEnglish()`

Determine if the current time series or paired data set is in English units. The function examines the data set parameter type and units label to establish the unit system.

**See also:** `isMetric()`; `convertToEnglishUnits()`.

**Parameters:** No parameters.

**Example:** `if dataSet.isEnglish() : print "English Units"`

**Returns:** `Constants.TRUE` if the data set units are English, otherwise `Constants.FALSE`.

**Generated Exceptions:** Throws an `hec.hecmath.HecMathException` if the unit system cannot be determined (parameter type and units label undefined).

## 7.10.30 Determine if Data is in Metric Units

`isMetric()`

Determine if the current time series or paired data set is in Metric (SI) units. The function examines the data set parameter type and units label to establish the unit system.

**See also:** `isEnglish()`;

`convertToMetricUnits()`.

**Parameters:** No parameters.

**Example:** `if dataSet.isMetric() : print "SI Units"`

**Returns:** Constants.TRUE if the data set units are Metric, otherwise Constants.FALSE.

**Generated Exceptions:** Throws an `hec.hecmath.HecMathException` if the unit system cannot be determined (parameter type and units label undefined).

### 7.10.31 Determine if Computation Stable for Given Muskingum Routing Parameters

```
isMuskingumRoutingStable(integer numberSubreaches,
    floating-point muskingumK,
    floating-point muskingumX)
```

Check for possible instability for the given Muskingum Routing parameters.

Test if the input parameters satisfy the stability criteria:

$$1/(2(1-x)) \leq K/\delta T \leq 1/2x$$

where  $\delta T$  = (time series time interval)/numberSubreaches

**Parameters:**

**numberSubreaches** – integer specifying the number of routing subreaches.

**muskingumK** –floating-point number specifying the Muskingum "K" parameter, in hours.

**muskingumX** - floating-point number specifying the Muskingum "x" parameter, between 0.0 and 0.5 (inclusive).

**Example:**

```
warning = tsDataSet.isMuskingumRoutingStable(
    reachCount,
    kVal,
    xVal)
if warning :
    print warning
return
```

**Returns:** A string if the stability criteria is not met. The string contains a warning message detailing the specific instability problem. Otherwise returns None.

**Generated Exceptions:** Throws an `hec.hecmath.HecMathException` if the current time series is not a regular interval time series, or if values for numberSubreaches or muskingumX are invalid.

### 7.10.32 Last Valid Value's Date and Time

```
lastValidDate()
```

Find and return the date and time of the last valid (non-missing) value in a time series data set.

**Parameters:** Takes no parameters

**Example:** `tsData.lastValidDate()`

**Returns:** An integer value translatable by HecTime representing the date and time of the last valid time series value.

### 7.10.33 Last Valid Value in a Time Series

```
lastValidValue()
```

Find and return the last valid (non-missing) value in a time series data set.

**Parameters:** Takes no parameters

**Example:** `tsData.lastValidValue()`

**Returns:** A floating-point value representing the last valid time series value.

### 7.10.34 Linear Regression Statistics

**LinearRegressionStatistics** is a class used to contain the linear regression and other correlation coefficients computed by the “correlationCoefficients” function.

The data members of LinearRegressionStatistics are:

integer	numberValidValues	
floating-point	regressionConstant	- intercept of regression line
floating-point	regressionCoefficient	- slope of regression line
floating-point	determinationCoefficient	
floating-point	standardErrorOfRegression	
floating-point	adjustedDeterminationCoefficient	
floating-point	adjustedStandardErrorOfRegression	

The “toString()” method will produce a multi-line character string that can be used to printout the correlation values and description.

**Example:**

```
linRegData =
    tsData.correlationCoefficients(otherTsData)
regCoef = linRegData.regressionCoefficient
print linRegData
```

**See also:** correlationCoefficients().

### 7.10.35 Natural Log, Base “e” Function

`log()`

Derive a new time series or paired data set from the natural log (log base “e”) of values of the current data set. Missing values in the original data set remain missing. Values less than or equal to 0.0 will be set to missing.

For paired data sets, use the `setCurve` method to first select the paired data curve(s).

**See also:** `log10()`, `setCurve()`.

**Parameters:** Takes no parameters

**Example:** `newDataSet = dataSet.log()`

**Returns:** A new `HecMath` object of the same type as the current object.

### 7.10.36 Log Base 10 Function

`log10()`

Derive a new time series or paired data set from the log base 10 of values of the current data set. Missing values in the original data set remain missing. Values less than or equal to 0.0 will be set to missing.

For paired data sets, use the `setCurve` method to first select the paired data curve(s).

**See also:** `log()`, `setCurve()`.

**Parameters:** Takes no parameters

**Example:** `newDataSet = dataSet.log10()`

**Returns:** A new `HecMath` object of the same type as the current object.

### 7.10.37 Maximum Value in a Time Series

`max()`

Find and return the maximum value of the current time series data set. Missing values are ignored.

**Parameters:** Takes no parameters

**Example:** `maxVal = tsData.max()`

**Returns:** A floating-point value representing the maximum value of the current time series.

### 7.10.38 Maximum Value's Date and Time

`maxDate()`

Find and return the date and time of the maximum value for the current time series data set. Missing values are ignored.

**Parameters:** Takes no parameters

**Example:** `maxDateTime = tsData.maxDate()`

**Returns:** An integer value translatable by HecTime representing the date and time of the maximum time series value.

### 7.10.39 Mean Time Series Value

`mean()`

Compute the mean value of the current time series data set. Missing values are ignored.

**Parameters:** Takes no parameters

**Example:** `meanVal = tsData.mean()`

**Returns:** A floating-point value representing the mean value of the current time series.

### 7.10.40 Merge Paired Data Sets

`mergePairedData(PairedDataMath pdData)`

Merge the current paired data set with the paired data set **pdData**. The resultant paired data set includes all the paired data curves from the current data set. Depending upon a previous use of the `setCurveMethod` on **pdData**, a single selected paired data curve or all curves from **pdData** are appended to the merged data set. The x-values for the two paired data sets must match exactly.

**See also:** `setCurve()`.

**Parameters:**

**pdData** – A paired data set with x-ordinates matching those of the current data set.

**Example:** `mergedCurve = curve.mergePairedData(anotherCurve)`

**Returns:** A new PairedDataMath object.

### 7.10.41 Merge Two Time Series Data Sets

`mergeTimeSeries`(TimeSeriesMath tsData)

Merge data from the current time series data set with the time series data set **tsData**. The resultant time series data set includes all the data points in the two time series, except where the data points occur at the same time. When data points from the two data sets are coincident in time, valid values in the current time series take precedence over valid values from **tsData**. However, if a coincident point is set to missing in the current time series data set, a valid value from **tsData** will be used for time in the resultant data set. If the values are missing for both data sets, the value is missing in the resultant data set.

The data sets for merging may have either regular or irregular time interval time series data. The data sets are tested to determine if they both have the same regular time interval. If not, the resultant data set is typed as an irregular interval data set.

**Parameters:**

**tsData** – A time series data set for merging with the current time series data set.

**Example:** `tsMerged = tsData.merge(otherTsData)`

**Returns:** A new TimeSeriesMath object.

### 7.10.42 Minimum Value in a Time Series

`min()`

Find and return the minimum value of the current a time series data set. Missing values are ignored.

**Parameters:** Takes no parameters

**Example:** `minVal = tsData.min()`

**Returns:** A floating-point value representing the minimum value of the current time series.

### 7.10.43 Minimum Value's Date and Time

`minDate()`

Find and return the date and time of the minimum value for the current time series data set. Missing values are ignored.

**Parameters:** Takes no parameters

**Example:** `minDateTime = tsData.minDate()`

**Returns:** An integer value translatable by HecTime representing the date and time of the minimum time series value.

## 7.10.44 Modified Puls or Working R&D Routing Function

```
modifiedPulsRouting( TimeSeriesMath tsFlow,
                    integer numberSubreaches,
                    floating-point muskingumX )
```

The current data set is a paired data set containing the storage-discharge table for Puls routing, where the x-values are storage and the y-values are discharge. The function derives a new time series data set from the Modified Puls or Working R&D routing of the time series data set **tsFlow**.

**numberSubreaches** is the number of routing subreaches.

The Working R&D method provides a means of including the effects of inflow on reach storage by use of the Muskingum “x” wedge coefficient. The Working R&D method is activated in the computation if **muskingumX** is greater than 0.0. However, **muskingumX** cannot be greater than 0.5.

### Parameters:

**tsFlow** – A regular interval time series data set for routing.

**numberSubreaches** – Number of routing subreaches.

**muskingumX** - Muskingum "X" parameter, between 0.0 and 0.5 (inclusive). Enter 0.0 to route by the Modified Puls method, or a value greater than 0.0 to apply the Working R&D.

### Example:

```
routedFlow =
    storDichareCurve.modifiedPulsRouting(
        tsFlow,
        reachCount,
        coefficient)
```

**Returns:** A new TimeSeriesMath object.

**Generated Exceptions:** Throws an `hec.hecmath.HecMathException` if the `tsMath` is not a regular interval time series; if `muskingumX` is less than 0.0 or greater than 0.5; if the current paired data set does not have both ascending x and y values.

## 7.10.45 Multiple Linear Regression Coefficients

```
multipleLinearRegression( sequence tsDataSequence,
                          floating-point minimumLimit,
                          floating-point maximumLimit )
```

Compute the multiple linear regression coefficients between the current time series data set and the array of independent time series data sets in **tsDataSequence**. The function stores the regression coefficients in a new paired data set. This paired data set may be used with the `multipleLinearRegression` function to derive a new estimated time series data set.

For the general linear regression equation, a dependent variable, Y, may be computed from a set independent variables, Xn:

$$Y = B_0 + B_1 * X_1 + B_2 * X_2 + B_3 * X_3$$

where Bn are linear regression coefficients.

For time series data sets, an estimate of the original time series data set values may be computed from a set of independent time series data sets using regression coefficients such that:

$$TsEstimate(t) = B_0 + B_1 * TS_1(t) + B_2 * TS_2(t) + \dots + B_n * TS_n(t)$$

where Bn are the set of regression coefficients and TSn are the time series data sets contained in **tsDataSequence**.

The parameters **minimumLimit** and **maximumLimit** may be used to exclude out of range values in the current time series data set from the regression determination. **minimumLimit** or **maximumLimit** may be entered as “Constants.UNDEFINED” to ignore the minimum or maximum value check.

**See also:** `applyMultipleLinearRegression()`.

**Parameters:**

**tsDataSequence** – sequence of TimeSeriesMath objects, which form the independent variables in the regression equation. Must all be regular interval and have the same time interval.

**minimumLimit** – A floating-point value. Values in the current time series exceeding minimumLimit are excluded from the regression analysis. Set to Constants.UNDEFINED to ignore this option.

**maximumLimit** – A floating-point value. Values in the current time series exceeding maximumLimit are excluded from the regression analysis. Set to Constants.UNDEFINED to ignore this option.

**Example:**

```
regression = tsFlow.multipleLinearRegression (
    [tsUpstrFlow1, tsUpstrFlow2, tsUpstrFlow3],
    0.,
    100000.)
```

**Returns:** A new PairedDataMath object containing the computed regression coefficients.

**Generated Exceptions:** Throws an `hec.hecmath.HecMathException` if the current data set and the data sets in `tsDataSequence` are not regular interval time series data sets with the same interval time.

### 7.10.46 Multiply by a Constant

`multiply(floating-point constant)`

Multiply the value **constant** to all valid values in the current time series or paired data set. For time series data, missing values are kept as missing.

For paired data, **constant** multiplies the y-values only. Use the `setCurveMethod` to first select the paired data curve(s).

**See also:** `multiply(TimeSeriesMath tsData); setCurve()`.

**Parameters:**

**constant** - A floating-point precision value.

**Example:** `newDataSet = dataSet.multiply(1.5)`

**Returns:** A new HecMath object of the same type as the current object.

### 7.10.47 Multiply by a Data Set

`multiply(TimeSeriesMath tsData)`

Multiply valid values in the current data set by the corresponding values in the data set **tsData**. Both data sets must be time series data set.

When multiplying one time series data set to another, there is no restriction that times in the two data sets match exactly. However, only values with coincident times will be multiplied. Times in the current time series data set that cannot be matched with times in the second data set are set to missing. Values in the current data set that are missing are kept as missing. Either or both data sets may be regular or irregular interval time series.

**See also:** `multiply(floating-point constant)`.

**Parameters:**

**tsData** - A time series data set.

**Example:** `newTsData = tsData.multiply(otherTsData)`

**Returns:** A new TimeSeriesMath object.

### 7.10.48 Muskingum Hydrologic Routing Function

`muskingumRouting( integer numberSubreaches,  
floating-point muskingumK,  
floating-point muskingumX)`

Route the current regular interval time series data set by the Muskingum Routing method. The current data set must be a regular interval time series data set. **muskingumK** is the Muskingum “K” parameter, in hours, and **muskingumX** is the Muskingum “x” parameter. **muskingumX** cannot be less than 0.0 or greater than 0.5.

The set of Muskingum routing parameters may potentially produce numerical instabilities in the routed time series. Use the function `isMuskingumRoutingStable()` to test if the Muskingum routing parameters may potentially have instabilities.

**See also:** `isMuskingumRoutingStable()`.

**Parameters:**

`numberSubreaches` – An integer specifying the number of routing subreaches.

`muskingumK` – A floating-point number specifying the Muskingum "K" parameter in hours.

`muskingumX` – A floating-point number specifying the Muskingum "x" parameter, between 0.0 and 0.5

**Example:**

```
routedFlows = tsFlows.muskingumRouting(reachCount, K, x)
```

**Returns:** A new `TimeSeriesMath` object.

**Generated Exceptions:** Throws an `hec.hecmath.HecMathException` if the current time series is not a regular interval time series; if `muskingumX` is less than 0.0 or greater than 0.5.

## 7.10.49 Number of Missing Values in a Time Series

`numberMissingValues()`

Count and return the number of missing values in the current time series data set.

**Parameters:** Takes no parameters

**Example:** `missingCount = tsData.numberMissingValues()`

**Returns:** An integer of the count of missing time series values.

## 7.10.50 Number of Valid Values in a Time Series

`numberValidValues()`

Count and return the number of valid values in the current time series data set.

**Parameters:** Takes no parameters

**Example:** `validCount = tsData.numberValidValues()`

**Returns:** An integer of the count of valid (non-missing) time series values.

## 7.10.51 Olympic Smoothing

```
olympicSmoothing( integer numberToAverageOver,  
                  boolean onlyValidValues,  
                  boolean useReduced)
```

Derive a new time series from the Olympic smoothing of **numberToAverageOver** values in the current time series.

**numberToAverageOver** must be an odd integer and greater than 2. Similar to centered moving average smoothing, except that the minimum and maximum values over the averaging interval are excluded from the computation.

If **onlyValidValues** is set to true, then if any values in the averaging interval are missing, the point in the resultant time series is set to missing. If **onlyValidValues** is set to false and there are missing values in the averaging interval, a smoothed point is still computed using the remaining valid values in the interval. If there are no valid values in the averaging interval, the point in the resultant time series is set to missing.

If **useReduced** is set to true, then moving average values can be still be computed at the beginning and end of the time series even if there are less than **numberToAverageOver** values in the interval. If **useReduced** is set to false, then the first and last  $\text{numberToAverageOver}/2$  points of the resultant time series are set to missing.

### Parameters:

**numberToAverageOver** – An integer specifying the number of values to average over for computing the smoothed time series. Must be an odd integer greater than 2.

**onlyValidValues** – Either Constants.TRUE, or Constants.FALSE, specifying whether all values in the averaging interval must be valid for the computed point in the resultant time series to be valid.

**useReduced** - Either Constants.TRUE, or Constants.FALSE, specifying whether to allow points at the beginning and end of the smoothed time series to be computed from a reduced ( less than **numberToAverageOver** ) number of values. Otherwise, set the first and last  $\text{numberToAverageOver}/2$  points of the new time series to missing.

### Example:

```
avgData = tsData.olympicSmoothing(  
    5,  
    1)
```

**Returns:** A new TimeSeriesMath object.

**Generated Exceptions:** Throws an `hec.hecmath.HecMathException` if the **numberToAverageOver** is less than 3 or not odd.

## 7.10.52 Period Constants Generation

`periodConstants`(TimeSeriesMath tsData)

Derive a new time series data set by applying values in the current time series data set to the times defined by the time series data set **tsData**. Both time series data sets may be regular or irregular interval. Values in a new time series are set according to:

$$ts1(j) \leq tsnew(i) < ts1(j+1), \quad TSNEW(i) = TS1(j)$$

where *ts1* is the time in the current time series, *TS1* is the value in the current time series, *tsnew* is the time in the new time series, *TSNEW* is the value in the new time series.

If times in the new time series precede the first data point in the current time series, the value for these times is set to missing. If times in the new time series occur after the last data point in the current time series, the value for these times is set to the value of the last point in the current time series.

Figure 7.7 shows interpolation of values with the `periodConstants` function.

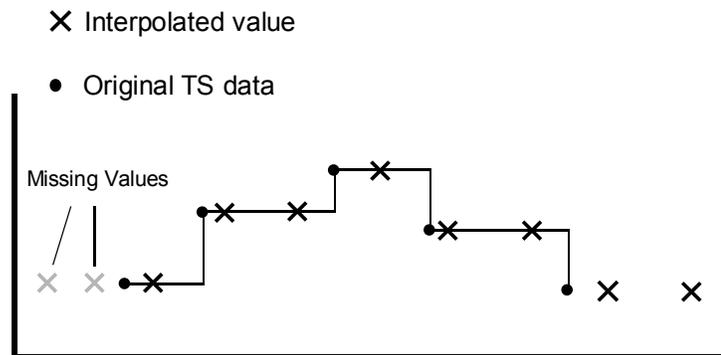


Figure 7.7 Interpolation of Time Series Values Using Period Constants function

### Parameters:

**tsData** – A regular or irregular interval time series data set.

### Example:

```
tsConstants = tsValues.periodConstants(tsData)
```

**Returns:** A new TimeSeriesMath object.

### 7.10.53 Polynomial Transformation

`polynomialTransformation`(TimeSeriesMath tsData)

Compute a polynomial transformation of a regular or irregular interval time series data set, **tsData**, using the polynomial coefficients stored in the current paired data set. Missing values in **tsData** remain missing in the resultant data set.

A new time series can be computed from an existing time series with the polynomial expression:

$$TS2(t) = B1 * TS1(t) + B2 * TS1(t)^2 + \dots + Bn * TS1(t)^n$$

where  $B_n$  are the polynomial coefficients for term “n.”

Values for the polynomial coefficients are stored in the x-values of the current paired data set. Before the above equation is applied, values in the input time series are adjusted by subtracting off the paired data “datum” value if defined. The units label and parameter type for the resultant time series are copied from the current paired data set x-units and parameter type.

**See also:** `polynomialTransformationWithIntegral()`.

**Parameters:**

**tsData** – A regular or irregular interval time series data set.

**Example:** `tsXform = pdCoef.polynomialTransformation(tsData)`

**Returns:** A new TimeSeriesMath object.

### 7.10.54 Polynomial Transformation with Integral

`polynomialTransformationWithIntegral`(TimeSeriesMath tsData)

Compute a polynomial transformation with integral of a regular or irregular interval time series data set, **tsData**, using the polynomial coefficients stored in the current paired data set. Missing values in **tsData** remain missing in the resultant data set.

This function is similar to the `polynomialTransformation` method, and the same set of polynomial coefficients are used. The equation for the polynomial transform is modified so that the transform of **tsData** is computed from the integral of the polynomial coefficients:

$$TS2(t) = B1 * TS1(t)^2/2 + B2 * TS1(t)^3/3 + \dots + Bn * TS1(t)^{n+1}/(n+1)$$

where  $B_n$  are the polynomial coefficients for term “n.”

Values for the polynomial coefficients are stored in the x-values of the current paired data set. Before the above equation is applied, values in the input time series are adjusted by subtracting off the paired data “datum” value

if defined. The units label and parameter type for the resultant time series are copied from the current paired data set x-units and parameter type.

**See also:** `polynomialTransformation()`.

**Parameters:**

`tsData` – A regular or irregular interval time series data set.

**Example:**

```
tsXform =
    pdCoef.polynomialTransformationWithIntegral(tsData)
```

**Returns:** A new TimeSeriesMath object.

## 7.10.55 Rating Table Interpolation

```
ratingTableInterpolation(TimeSeriesMath tsData)
```

Transform/interpolate values in the time series data set **tsData** using the rating table x-y values stored in the current paired data set. For example, you can use the function to transform a time series of stage to a time series of flow using a stage-flow rating table. **tsData** may be a regular or irregular time interval data set. Missing values in **tsData** are kept missing in the resultant data set.

Create the paired data set with the rating table option to set values for “**datum**”, “**shift**”, and “**offset**.” By default these values are 0.0. The shift is added to and the datum subtracted from all input time series values. If the rating table is Log-Log, the table x-values are adjusted by subtracting the offset.

Units and parameter type in resultant time series data set are defined by the y-units label and parameter type of the current paired data set. All other names and labels are copied over from **tsData**.

**See also:** `reverseRatingTableInterpolation()`.

**Parameters:**

`tsData` – A regular or irregular interval TimeSeriesMath object.

**Example:**

```
tsFlow =
    stageFlowCurve.ratingTableInterpolation(tsStage)
```

**Returns:** A new TimeSeriesMath object.

## 7.10.56 Reverse Rating Table Interpolation

`reverseRatingTableInterpolation`(TimeSeriesMath tsData)

Transform/interpolate values in the time series data set **tsData** using the reverse of the rating table stored in the current paired data set. For example, the function may be used to transform a time series of flow to a time series of stage using a stage-flow rating table. **tsData** may be a regular or irregular time interval data set. Missing values in **tsData** are kept missing in the resultant data set.

The paired data set should be created with the rating table option to set values for “datum”, “shift”, and “offset.” By default, these values are 0.0. The shift is subtracted from, and the datum added to all input time series values. If the rating table is Log-Log, the table x-values are adjusted by subtracting the offset. Refer to the `ratingTableInterpolation()` description for comparison to this function.

Units and parameter type in resultant time series data set are defined by the x-units label and parameter type of the current paired data set. All other names and labels are copied over from the **tsData**.

**See also:** `ratingTableInterpolation`.

### Parameters:

**tsData** – A regular or irregular interval TimeSeriesMath object.

### Example:

```
tsStage =
  stageFlowCurve.reverseRatingTableInterpolation(
    tsFlow)
```

**Returns:** A new TimeSeriesMath object.

## 7.10.57 Round to Nearest Whole Number

`round()`

Rounds values in a time series or paired data set to the nearest whole number.

The function rounds up the decimal portion of a number if equal to or greater than .5 and rounds down decimal values less than .5. For example:

10.5 is rounded to 11.

10.499 is rounded to 10.

The x-values in paired data sets are unaffected by the function, only the y-value data are rounded. For time series data sets, missing values are kept missing.

For paired data sets, use the `setCurve()` method to first select the paired data curve(s).

**See also:** `roundOff()`;

`truncate()`;

`setCurve()`.

**Parameters:** Takes no parameters

**Example:** `roundedData = dataSet.round()`

**Returns:** A new HecMath object of the same type as the current object.

### 7.10.58 Round Off to Specified Precision

`roundOff(integer significantDigits, integer powerOfTensPlace)`

Round values in a time series or paired data set to a specified number of significant digits and/or power of tens place. For the power of tens place, -1 specifies rounding to one-tenth (0.1), while +2 rounds to the hundreds (100). For example:

1234.123456 will round to:

1230.0 for number of significant digits = 3, power of tens place = -1

1234.1 for number of significant digits = 6, power of tens place = -1

1234 for number of significant digits = 6, power of tens place = 0

1230 for number of significant digits = 6, power of tens place = 1

The x-values in paired data sets are unaffected by the function, only the y-value data are rounded. For time series data sets, missing values are kept missing.

For paired data sets, use the `setCurve()` method to first select the paired data curve(s).

**See also:** `round()`;

`truncate()`;

`setCurve()`.

**Parameters:**

**significantDigits** – An integer specifying the number of significant digits to use in the rounding.

**powerOfTensPlace** – An integer specifying the power of tens place to use in the rounding.

**Example:** `roundedData = dataSet.roundOff(5, -2)`

**Returns:** A new HecMath object of the same type as the current object.

## 7.10.59 Screen for Erroneous Values Based on Forward Moving Average

```
screenWithForwardMovingAverage(integer numberToAverageOver,
    floating-point changeLimit,
    boolean setInvalidToMissingValue,
    string qualityFlagForInvalidValue)
```

Screen the current time series data set for possible erroneous values based on the deviation from the forward moving average over **numberToAverageOver** values computed at the previous point. If the deviation from the moving average is greater than **changeLimit**, the value fails the screening test. Data values failing the screening test are assigned a quality flag and/or are set to missing.

Missing values and values failing the screening test are not counted in the moving average and the divisor of the average is less one for each such value. At least 2 values must be defined in the moving average else the moving average is undefined and value being examined is screened acceptable.

If **setInvalidToMissingValue** is true, values failing the screening test are set to missing.

If **qualityFlagForInvalidValue** is set to a character or string recognized as a valid quality flag, the quality flag will be set for tested values. If there is no previously existing quality available for the time series, the quality flag array will be created for the time series. Values failing the quality test are set to the user specified quality flag for invalid values. If there is existing quality data and the time series value passes the quality test, the existing quality flag for the points is unchanged. If there was no previously existing quality and the time series value passes the quality test, the quality flag for the point is set to "Okay."

The acceptable values for **qualityFlagForInvalidValue** strings are: "M" or "Missing", "R" or "Rejected", "Q" or "Questionable." A blank string (" ") is entered to disable the setting of the quality flag.

For the example,

```
resultantDataSet = dataSet.screenWithForwardMovingAverage (
    16, 100., Constants.TRUE, "R" )
```

the forward moving average will be computed over 16 values, values deviating from the moving average by more than 100.0 will be set to missing and flagged as rejected.

**Parameters:**

**numberToAverageOver** – An integer specifying the number of averaging values. Must be at least 2.

**changeLimit** – A floating-point number specifying the maximum change allowed in the tested value from the forward moving average value.

**setInvalidToMissingValue** – Either Constants.TRUE, or Constants.FALSE, specifying whether time series values failing the screening test are set to the "Missing" value.

**qualityFlagForInvalidValue** – A string representing the quality flag setting for values failing the screening test. The accepted character strings are: "M" or "Missing", "R" or "Rejected", "Q" or "Questionable." An empty string ("") is entered to disable the setting of the quality flag.

**Example:**

```
screenedData = tsData.screenWithForwardMovingAverage(
    16,
    100.,
    Constants.TRUE,
    "R")
```

**Returns:** A new TimeSeriesMath object.

**Generated Exceptions:** Throws a HecMathException if **numberToAverageOver** is less than 2; if an unrecognized quality flag is entered for **qualityFlagForInvalidValue** or if **setInvalidToMissingValue** is false and **qualityFlagForInvalidValue** is blank (no action would occur).

## 7.10.60 Screen for Erroneous Values Based on Maximum/Minimum Range

```
screenWithMaxMin(floating-point minValueLimit,
    floating-point maxValueLimit,
    floating-point changeLimit,
    boolean setInvalidToMissingValue,
    string qualityFlagForInvalidValue)
```

Flag values in a time series data set exceeding minimum and maximum limit values or maximum change limit.

Values in the time series are screened for quality. Values below **minValueLimit** or above **maxValueLimit** or with a change from the previous time series value greater than **changeLimit** fail the screening test. The maximum change comparison is done only when consecutive values are not flagged.

If **setInvalidToMissingValue** is set to true, values failing the screening test are set to the "Missing" value.

If **qualityFlagForInvalidValue** is set to a character or string recognized as a valid quality flag, the quality flag will be set for tested values. If there is no previously existing quality available for the time series, the quality flag array will be created for the time series. Values failing the quality test are set to the user specified quality flag for invalid values. If there is existing quality data and the time series value passes the quality test, the existing quality flag for the points is unchanged. If there was no previously existing quality and the time series value passes the quality test, the quality flag for the point is set to "Okay."

For example,

```
resultantDataSet = dataSet.screenWithMaxMin ( 0.0, 1000., 100.,
Constants.FALSE, "R" )
```

time series values less than 0.0, or greater than 1000., or with a change from a previous point greater than 100 will be flagged as "Rejected." Flagged points however will not be set to the "Missing" value.

#### Parameters:

**minValueLimit** – A floating-point number specifying the minimum valid value limit.

**maxValueLimit** - A floating-point number specifying the maximum valid value limit.

**changeLimit** - A floating-point number specifying the maximum change allowed in the tested value from the previous time series value.

**setInvalidToMissingValue** – Either Constants.TRUE, or Constants.FALSE, specifying whether time series values failing the screening test are set to the "Missing" value.

**qualityFlagForInvalidValue** - A string representing the quality flag setting for values failing the screening test. The accepted character strings are: "M" or "Missing", "R" or "Rejected", "Q" or "Questionable." An empty string (" ") is entered to disable the setting of the quality flag.

#### Example:

```
screenedData = tsData.screenWithMaxMin (
    0.,
    1000.,
    100.,
    Constants.FALSE,
    "R")
```

**Returns:** A new TimeSeriesMath object.

**Generated Exceptions:** Throws a HecMathException if an unrecognized quality flag is entered for `qualityFlagForInvalidValue` or if `setInvalidToMissingValue` is false and `qualityFlagForInvalidValue` is blank (no action would occur).

### 7.10.61 Select a Paired Data Curve by Curve Label

```
setCurve(string curveName)
```

Select, by curve label, the paired data curve for performing subsequent arithmetic operations or math functions. By default, a paired data set loaded from file has all curves selected.

A paired data set may contain more than one set of y-values. However, a user may wish to modify only one curve of the data set. For example, using the function `".add( 2.0 )"` would by default add 2.0 to all y-values for all curves. The `setCurve()` call may be used to limit the operation to just one selected set of y-values.

The function searches the paired data set list of curve labels for a match to **curveName**. If a match is found, that curve is set as the selected curve.

**See also:** `setCurve( integer curveNumber )`.

**Example:** `damageCurve.setCurve( "RESIDENTIAL" )`

**Parameters:**

**curveName** – The curve label (a string) to set as the selected curve.

**Returns:** Nothing.

**Generated Exceptions:** Throws a HecMathException – if `curveName` is not found in the paired data set curve labels.

### 7.10.62 Select a Paired Data Curve by Curve Number

```
setCurve(integer curveNumber)
```

Select, by curve number, the paired data curve for performing subsequent arithmetic operations or math functions. By default, a paired data set loaded from file has all curves selected.

A paired data set may contain more than one set of y-values. However, a user may wish to modify only one curve of the data set. For example, using the function `".add( 2.0 )"` would by default add 2.0 to all y-values for all curves. The `setCurve()` call can be used to limit the operation to just one selected set of y-values. The function sets a curve index internal to the paired data set. The option is to select one curve or all curves.

Curve numbering begins with “0.” If a paired data set has two curves, the first curve is selected by, “setCurve(0).” To select the second curve, use “setCurve(1).”

All curves in a paired data set are selected by setting **curveNumber** to -1.

**See also:** setCurve( String curveName).

**Parameters:**

**curveNumber** – An integer specifying the curve to set as the selected curve. Curve numbering begins with 0. Set to -1 to select all curves.

**Example:** `ruleCurve.setCurve(-1)`

**Returns:** Nothing.

### 7.10.63 Set Data

`setData(hec.io.DataContainer container)`

Sets the data container for the current data set. For time series data sets, this is a **hec.io.TimeSeriesContainer**. For paired data sets, container should be a **hec.io.PairedDataContainer**. Containers are generated by some of the other functions.

The **hec.io.DataContainer** class and the **hec.io.TimeSeriesContainer** and the **hec.io.PairedDataContainer** subclasses contain the time series and paired data values.

**Parameters:**

**container** – A **hec.io.TimeSeriesContainer** for time series data sets, or a **hec.io.PairedDataContainer** for paired data sets.

**Example:** `dataSet.setContainer(TSContainer)`

**Returns:** Nothing.

**Generated Exceptions:** Throws a **HecMathException** if container is not of type **hec.io.TimeSeriesContainer** for time series data sets or not of type **hec.io.PairedDataContainer** for paired data sets.

### 7.10.64 Set Location Name for Data Set

```
setLocation(String locationName)
```

Set the location name for a data set, which changes the B-Part of the HEC-DSS pathname. The new pathname will be used in plots, tables, and in the write() method of DSSFile objects.

**Parameters:**

`locationName` – A string specifying the new location name for the data set.

**Example:** `dataSet.setLocation("OAKVILLE")`

**Returns:** Nothing.

### 7.10.65 Set Parameter for Data Set

```
setParameterPart(String parameterName)
```

Set the parameter name for a data set, which changes the C-Part of the HEC-DSS pathname. The new pathname will be used in plots, tables, and in the write() method of DSSFile objects.

**Parameters:**

`parameterName` – A string specifying the new parameter name for the data set.

**Example:** `dataSet.setParameterPart("ELEV")`

**Returns:** Nothing.

### 7.10.66 Set Pathname for Data Set

```
setPathname(String pathname)
```

Set the pathname for a data set. The new pathname will be used in plots, tables, and in the write() method of DSSFile objects.

**Parameters:**

`pathname` – A string specifying the new pathname for the data set.

**Example:** `dataSet.setPathname("//OAKVILLE/STAGE//1HOUR/OBS/")`

**Returns:** Nothing.

### 7.10.67 Set Time Interval for Data Set

```
setTimeInterval(String interval)
```

Set the time interval for a data set, which changes the E-Part of the pathname. The new pathname will be used in plots, tables, and in the write() method of DSSFile objects.

**Parameters:**

`interval` – A string specifying the new interval for the data set.

**Example:** `dataSet.setTimeInterval("1HOUR")`

**Returns:** Nothing.

### 7.10.68 Set Data Type for Time Series Data Set

```
setType(string typeString)
```

Set the data for a time series data set.

**Parameters:**

`typeString` – A string specifying the data type for the data set. This should be "INST-CUM", "INST-VAL", "PER-AVER" or "PER-CUM"

**Example:** `dataSet.setType("PER-AVER")`

**Returns:** Nothing.

### 7.10.69 Set Units Label for Data Set

```
setUnits(String unitsString)
```

Set the units label for a data set. For a paired data set, the call sets the y-units label.**Parameters:**

`unitsString` – A string specifying the units label for the data set.

**Example:** `dataSet.setUnits("CFS")`

**Returns:** Nothing.

### 7.10.70 Set Version Name for Data Set

```
setVersion(String versionName)
```

Set the version name for a data set, which changes the F-Part of the pathname. The new pathname will be used in plots, tables, and in the write() method of DSSFile objects.

**Parameters:**

**version** – A string specifying the new location for the data set.

**Example:** `dataSet.setVersion("OBSERVED")`

**Returns:** Nothing.

### 7.10.71 Set Watershed Name for Data Set

```
setWatershed(String watershedName)
```

Set the watershed (or river) name for a data set, which changes the A-Part of the pathname. The new pathname will be used in plots, tables, and in the write() method of DSSFile objects.

**Parameters:**

**watershedName** – A string specifying the new watershed name for the data set.

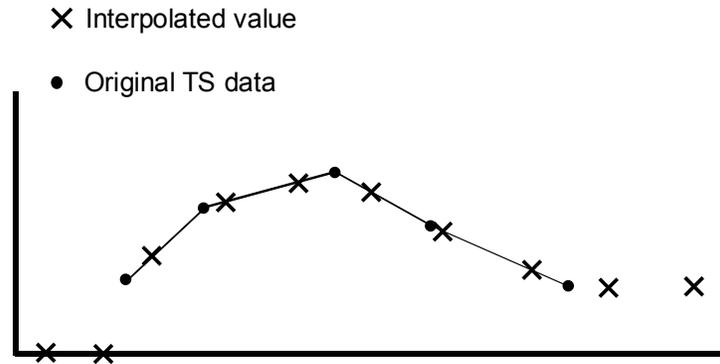
**Example:** `dataSet.setWatershed("OAK RIVER")`

**Returns:** Nothing.

### 7.10.72 Shift Adjustment of Time Series Data

```
shiftAdjustment(TimeSeriesMath tsData)
```

Derive a new time series data set by linear interpolation of values in the current time series data set at the times defined by the time series data set **tsData**. If times in the new time series precede the first data point in the current time series, the value for these times is set to 0.0. If times in the new time series occur after the last data point in the current time series, the value for these times is set to the value of the last point in the current time series. Interpolation of values with the **shiftAdjustment** function is shown in Figure 7.8.



**Figure 7.8 Interpolation of time series values using Shift Adjustment function**

Both time series data sets may be regular or irregular interval. Interpolated points must be bracketed or coincident with valid (not missing) values in the original time series, otherwise the values are set as missing.

**Parameters:**

**tsData** – A regular or irregular interval time series data set.

**Example:**

```
tsInterp = tsValues.shiftAdjustment(tsData)
```

**Returns:** A new TimeSeriesMath object.

### 7.10.73 Shift Time Series in Time

```
shiftInTime(string timeShiftString)
```

Shift the times in the current time series data set by the amount specified with **timeShiftString**. The data set may be regular or irregular interval time series data. Data set values are unchanged.

**timeShiftString** has the form “nT”, where “n” is an integer number and “T” is “M”(inute), “H”(our), or “D”(ay). Only the first character is significant for “T”.

**Parameters:**

**timeShiftString** – A string specifying the time increment to shift times in the current time series data set.

**Example:** `TsShifted = tsData.shiftInTime("3H")`

**Returns:** A new TimeSeriesMath object.

### 7.10.74 Sine Trigonometric Function

```
sin()
```

Derive a new time series or paired data set from the sine of values of the current data set. The resultant data set values are in radians. For time series data, missing values are kept as missing.

For paired data sets, use the `setCurveMethod` to first select the paired data curve(s).

**See also:** `setCurve()`.

**Parameters:** Takes no parameters

**Example:** `newDataSet = dataSet.sin()`

**Returns:** A new `HecMath` object of the same type as the current object.

### 7.10.75 Skew Coefficient

```
skewCoefficient()
```

Compute the skew coefficient of the current time series data set. Missing values are ignored.

**Parameters:** Takes no parameters

**Example:** `skewCoefficient = dataSet.skewCoefficient()`

**Returns:** A floating-point value representing the skew coefficient of the current time series.

### 7.10.76 Snap Irregular Times to Nearest Regular Period

```
snapToRegularInterval(string timeIntervalString,  
                      string timeOffsetString,  
                      string timeBackwardString,  
                      string timeForwardString )
```

"Snap" data from the current irregular or regular interval time series to form a new regular interval time series of the specified interval and offset. For example, a time series record from a gauge recorder collects readings 6 minutes past the hour. The function may be used to "snap" or shift the time points to the top of the hour.

The regular interval time of the resultant time series is specified by **timeIntervalString**. **timeIntervalString** is a valid HEC time increment string (e.g. "1MIN", "15MIN", "1HOUR", "6HOUR", "1DAY", "1MONTH").

Times in the resultant time series may be shifted (offset) from the regular interval time by the increment specified by `timeOffsetString`. As an example, the offset could be used to shift times from the top of the hour to instead 6 minutes past the hour. Data from the original time series is "snapped" to the regular interval if the time of the data falls within the time window set by the `timeBackwardString` and the `timeForwardString`. That is, if the new regular interval is at the top of the hour and the time window extends to 9 minutes before the hour and 15 minutes after the hour, an original data point at 0852 would be snapped to the time 0900 while a point at 0916 would be ignored.

`timeOffsetString`, `timeBackwardString` and `timeForwardString` are time increment strings expressed as "nT", where "n" is an integer number and "T" is one of the time increments: "M(INUTES)", "D(AYS)" or "H(OUR)" (characters in the parenthesis are optional). For the example of the previous paragraph, `timeIntervalString` would be "1HOUR", `timeOffsetString` would be "0M", `timeBackwardString` would be "9M" (or "9min") and `timeForwardString` would be "15M." A blank string (" ") is equivalent to "0M."

By default values in the resultant regular interval time series data set are set to missing unless matched to times in the current time series data set within the time window tolerance set by `timeBackwardString` and `timeForwardString`.

#### Parameters:

`timeIntervalString` – A string specifying the regular time interval for the resultant time series.

`timeOffsetString` – A string specifying the offset of the new time points from the regular interval time. This variable may be an empty string (" ") or None.

`timeBackwardString` – A string specifying the time to look backwards from the regular time interval for valid time points.

`timeForwardString` – A string specifying the time to look forward from the regular time interval for valid time points.

#### Example:

```
rtsData = itsData.snapToRegularInterval(
    "1HOUR",
    None,
    "5Min",
    "5Min")
```

**Returns:** A new regular interval TimeSeriesMath object.

### 7.10.77 Square Root

`sqrt()`

Derive a new time series or paired data set computed from the square root of values of the current data set. For time series data, missing values are kept as missing. Values less than zero are set to missing.

For paired data sets, use `setCurve` to first select the paired data curve(s).

**See also:** `setCurve()`.

**Parameters:** Takes no parameters

**Example:** `newDataSet = dataSet.sqrt()`

**Returns:** A new `HecMath` object of the same type as the current object.

### 7.10.78 Standard Deviation of Time Series

`standardDeviation()`

Compute the standard deviation value of the current time series data set. Missing values are ignored.

**Parameters:** Takes no parameters

**Example:** `stdDev = tsData.standardDeviation()`

**Returns:** A floating-point value representing the standard deviation of the current time series.

### 7.10.79 Straddle Stagger Hydrologic Routing

`straddleStaggerRouting`(integer numberToAverage,  
integer numberToLag,  
integer numberSubreaches )

Route the current regular interval time series data set using the Straddle-Stagger hydrologic routing method. **numberToAverage** specifies the number of ordinates to average over (Straddle). **numberToLag** specifies the number of ordinates to lag (Stagger). The number of routing **subreaches** is set by **numberSubreaches**.

**Parameters:**

**numberToAverage** – An integer specifying the number of ordinates to average over (Straddle).

**numberToLag** – An integer specifying the number of ordinates to lag (Stagger).

**numberSubreaches** – An integer specifying the number of routing subreaches.

**Example:**

```
tsRouted = tsFlow.straddleStaggerRouting(  
    numberAver,  
    lag,  
    reachCount)
```

**Returns:** A new TimeSeriesMath object.

## 7.10.80 Subtract a Constant

```
subtract(floating-point constant)
```

Subtract the value constant from all valid values in the current time series or paired data set. For time series data, missing values are kept as missing.

For paired data, constant is subtracted from y-values only. Use the setCurve method to first select the paired data curve(s).

**See also:** add(HecMath hecMath);

```
setCurve()
```

**Parameters:**

**constant** - A floating-point value.

**Example:** `newDataSet = dataSet.subtract(5.3)`

**Returns:** A new HecMath object of the same type as the current object.

## 7.10.81 Subtract a Data Set

```
subtract(TimeSeriesMath tsData)
```

Subtract the values in the data set **tsData** from the values in the current data set. Both data sets must be time series data set.

When subtracting one time series data set from another, there is no restriction that times in the two data sets match exactly. However, only values with coincident times will be subtracted. Times in the current time series data set that cannot be matched with times in the second data set are set missing. Values in the current data set that are missing are kept as missing. Either or both data sets may be regular or irregular interval time series.

**See also:** subtract(floating-point constant).

**Parameters:**

**tsData** - A TimeSeriesMath object.

**Example:** `newDataSet = dataSet.subtract(otherDataSet)`

**Returns:** A new TimeSeriesMath object.

## 7.10.82 Successive Differences for Time Series

`successiveDifferences()`

Derive a new time series from the difference between successive values in the current regular or irregular interval time series data set. The current data must be of type "INST-VAL" or "INST-CUM." A value in the resultant time series is set to missing if either the current or previous value in the current time series is missing (need to have two consecutive valid values). If the data type of the current data set is "INST-CUM" the resultant time series data set is assigned the type "PER-CUM", otherwise the data type does not change.

**Parameters:** Takes no parameters

**Example:** `newTsData = tsData.successiveDifferences()`

**Returns:** A new TimeSeriesMath object.

**Generated Exceptions:** Throws a HecMathException if the current data set is not of type "INST-VAL" or "INST-CUM."

## 7.10.83 Sum Values in Time Series

`sum()`

Sum all the values of the current time series data set. Missing values are ignored.

**Parameters:** Takes no parameters

**Example:** `total = tsData.sum()`

**Returns:** A floating-point value representing the sum of all valid values of the current time series.

## 7.10.84 Tangent Trigonometric Function

`tan()`

Derive a new time series or paired data set computed from the tangent of values of the current data set. For time series data, missing values are kept as missing. If the cosine of the current time series value is zero, the value is set missing.

For paired data sets, use the `setCurve` method to first select the paired data curve(s).

**See also:** `setCurve()`.

**Example:** `newDataSet = dataSet.tan()`

**Parameters:** Takes no parameters

**Returns:** A new HecMath object of the same type as the current object.

### 7.10.85 Time Derivative (Difference Per Unit Time)

`timeDerivative()`

Derive a new time series data set from the successive differences per unit time of the current regular or irregular interval time series data set. For the time “t”,

$$TS2(t) = ( TS1(t) - TS1(t-1) ) / DT$$

where DT is the time difference between t and t-1. For the current form of the function, the units of DT are minutes.

A value in the resultant time series is set to missing if either the current or previous value in the original time series is missing (need to have two consecutive valid values). By default, the data type of the resultant time series data set is assigned as "PER-AVER."

**Parameters:** Takes no parameters

**Example:** `newTsData = tsData.timeDerivative()`

**Returns:** A new TimeSeriesMath object.

### 7.10.86 Transform Time Series to Regular Interval

`transformTimeSeries`(string timeIntervalString,  
string timeOffsetString,  
string functionTypeString )

Generate a new regular interval time series data set from the current regular or irregular time series. The new time series is computed having the regular time interval specified by **timeIntervalString** and time offset set by **timeOffsetString**.

Values for the new time series are computed from the original time series data set using one of seven available functions. The function is selected by setting **functionTypeString** to one of the following types:

- "INT" - Interpolate at end of interval
- "MAX" - Maximum over interval
- "MIN" - Minimum over interval
- "AVE" - Average over interval
- "ACC" - Accumulation over interval
- "ITG" - Integration over interval
- "NUM" - Number of valid data over interval

where “interval” is the interval between time points in the new time series.

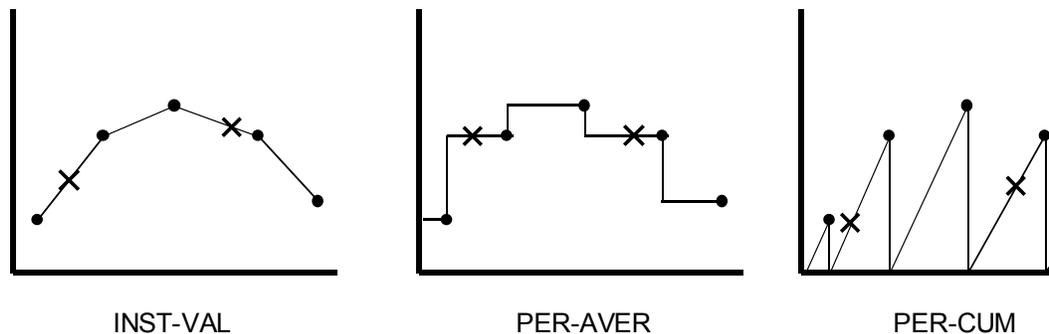
The regular interval time of the new time series is specified by **timeIntervalString**. **timeIntervalString** is a valid HEC time increment

string (e.g. “1MIN”, “15MIN”, “1HOUR”, “6HOUR”, “1DAY”, “1MONTH”).

Times in the resultant time series may be shifted (offset) from the regular interval time by the increment specified by `timeOffsetString`. As an example, the offset could be used to shift times from the top of the hour to 6 minutes past the hour. Typically no offset is used.

The data type of the original time series data governs how values are interpolated. Data type “INST-VAL” (or “INST-CUM”) considers the value to change linearly over the interval from the previous data value to the current data value. Data type “PER-AVER” considers the value to be constant at the current data value over the interval. Data type “PER-CUM” considers the value to increase from 0.0 (at the start of the interval) up to the current value over the interval. Interpolation of the three data types is illustrated in Figure 7.9.

✕ Interpolated value



**Figure 7.9** Interpolation of “INST-VAL”, “PER-AVER” and “PER-CUM” data

How interpolation is performed for a specific data type influences the computation of new time series values for the selected function. For example, if the data type is “INST-VAL”, the function “Maximum over interval” is evaluated by: Finding the maximum value of the data points from the original time series that are inclusive in the new time interval. Linearly interpolate values at beginning and ending of the new time interval, and determine if these values represent the maximum over the interval.

Referring to the plots in Figure 7.9, the “Average over interval” function is applied to a time series by integrating the area under the curve between interpolated points and dividing the result by the interval time.

**See also:** `transformTimeSeries( TimeSeriesMath tsData, string functionName )`

**Parameters:**

**timeIntervalString** – A string specifying the regular time interval for the resultant time series.

**timeOffsetString** – A string specifying the offset of the new time points from the regular interval time. This variable may be a blank string (“”).

**functionTypeString** – A string specifying the method for computing values for the new time series data set.

**Example:**

```
newTsData = tsData.transformTimeSeries(
    "1Day",
    "0M",
    "AVE")
```

**Returns:** A new regular interval TimeSeriesMath object.

## 7.10.87 Transform Time Series to Irregular Interval

```
transformTimeSeries(TimeSeriesMath tsData,
    string functionTypeString )
```

Generate a new time series data set from the current regular or irregular time series. The times for the new data set are defined by the times in tsData, which may be a regular or irregular time series data set.

Values for the new time series are computed from the original time series data set using one of seven available functions. The function is selected by setting **functionTypeString** to one of the following types:

- "INT" - Interpolate at end of interval
- "MAX" - Maximum over interval
- "MIN" - Minimum over interval
- "AVE" - Average over interval
- "ACC" - Accumulation over interval
- "ITG" - Integration over interval
- "NUM" - Number of valid data over interval

where “interval” is the interval between time points in the new time series.

The data type of the original time series data governs how values are interpolated. Data type “INST-VAL” (or “INST-CUM”) considers the value to change linearly over the interval from the previous data value to the current data value. Data type “PER-AVER” considers the value to be constant at the current data value over the interval. Data type “PER-CUM” considers the value to increase from 0.0 (at the start of the interval) up to the current value

over the interval. Interpolation of the three data types is illustrated in Figure 7.9.

How interpolation is performed for a specific data type influences the computation of new time series values for the selected function. For example, if the data type is “INST-VAL”, the function “Maximum over interval” is evaluated by: Finding the maximum value of the data points from the original time series that are inclusive in the new time interval. Linearly interpolate values at beginning and ending of the new time interval, and determine if these values represent the maximum over the interval.

Referring to the plots in Figure 7.9, the “Average over interval” function is applied to a time series by integrating the area under the curve between interpolated points and dividing the result by the interval time.

**See also:** `transformTimeSeries( string timeIntervalString,  
string timeOffsetString, string functionTypeString )`

**Parameters:**

`tsMath` – A TimeSeriesMath object used to define the times for the new data set.

`functionTypeString` – A String specifying the method for computing values for the new time series data set.

**Example:**

```
newTsData = tsValues.transformTimeSeries(  
    tsTimeTemplate,  
    "MAX")
```

**Returns:** A new TimeSeriesMath object.

## 7.10.88 Truncate to Whole Numbers

`truncate()`

Truncates values in a time series or paired data set to the nearest whole number. For example:

10.99 is truncated to 10.

10.499 is truncated to 10.

The x-values in paired data sets are unaffected by the function, only the y-value data are truncated. For time series data sets, missing values are kept missing.

For paired data sets, use the `setCurve` method to first select the paired data curve(s).

**See also:** `setCurve()`.

**Parameters:** Takes no parameters

**Example:** `newDataSet = dataSet.truncate()`

**Returns:** A new HecMath object of the same type as the current object.

### 7.10.89 Two Variable Rating Table Interpolation

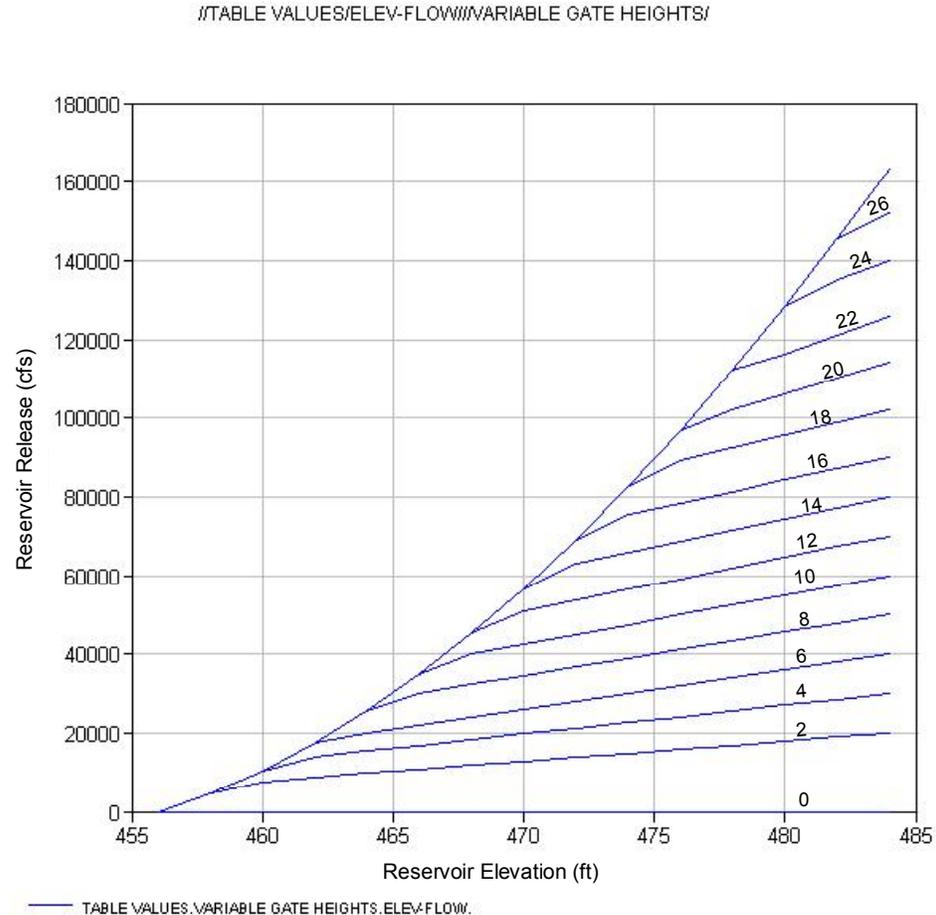
```
twoVariableRatingTableInterpolation(  
    TimeSeriesMath tsDataX,  
    TimeSeriesMath tsDataZ)
```

Derive a new time series data set by using the x-y curves in the current paired data set to perform two-variable rating table interpolation of the time series **tsDataX** and **tsDataZ**. For two-variable rating table interpolation, the current paired data set should have more than one curve (multiple sets of y-values).

As an example, reservoir release is a function of both the gate opening height and reservoir elevation (Figure 7.10). For each gate opening height, there is a reservoir elevation-reservoir release curve, where reservoir elevation is the independent variable (x-values) and reservoir release the dependent variable (y-values) of a paired data set. Each paired data curve has a curve label. In this case, the curve label is assigned the gate opening height. Using the paired data set shown in Figure 7.10, the function may be employed to interpolate time series values of reservoir elevation (**tsDataX**) and gate opening height (**tsDataZ**) to develop a time series of reservoir release.

No extrapolation is performed. If time series values from **tsDataX** or **tsDataZ** are outside the range bounded by the paired data, the new time series value is set to missing. Units and parameter type in the new time series are set to the y-units label and parameter of the current paired data set. All other names and labels are copied over from **tsDataX**.

Times for **tsDataX** and **tsDataZ** must match. Curve labels must be set for curves in the rating table paired data set and must be interpretable as numeric values.



**Figure 7.10** Example of two variable rating table paired data, reservoir release as a function of reservoir elevation and gate opening height (curve labels).

#### Parameters:

**tsDataX** – A regular or irregular interval TimeSeriesMath object, interpreted as x-ordinate values in the two variable interpolation.

**tsDataZ** – A regular or irregular interval TimeSeriesMath object, interpreted as z-ordinate values, (value defined by the paired data curve labels).

**Example:**

```
tsOutflow =
gateCurve.twoVariableRatingTableInterpolation(
    tsElevation,
    tsGateOpening)
```

**Returns:** A new TimeSeriesMath object.

**Generated Exceptions:** Throws a HecMathException if times do not match for **tsDataX** and **tsDataZ**; if the paired data curve labels are blank or cannot be interpreted as number values.