Compengine

A timeseries computation engine for use with CWMS 2.0+ databases

Version 1.3

15 September 2012

Introduction

The “compengine” is intended to be a supplementary computation and calculation engine for use with the US Army Corps of Engineers CWMS 2.0 database and greater. It has a simple, timeseries centric language that abstracts all the looping and programmatic cruft involved with timeseries computations. It allows the user to specify, chain and store timeseries computations into objects or directly into the database. Time series objects are containers that contain an arbitrary amount of time series data which consist of:

[timestamp, value, quality flag]

The language can be used directly in an interpreter. Scripts can be formed from commands and piped to the interpreter. The interpreter is case sensitive. Compengine can be run in both windows and unix environments. It requires Python 2.6+, oracle binaries and the python cx_Oracle package being installed. An example location for compengine to be stored is:

```
/usr2/rwcds/ce/v1/common/bin
```

The compengine can be run interactively from a shell prompt by executing the compengine:

```
rwcds@nww-wmlocal1:/usr2/rwcds/ce/v1/common/bin> ./compengine
```

Compengine scripts are collections of commands written in the compengine language. To execute batch scripts, simply pipe the contents to the compengine:

```
cat inflow.ce | /usr2/rwcds/ce/v1/common/bin/compengine
```

Syntax

The interpreter is *case sensitive*. The basic syntax of this engine follows the following rules:

command <parameters (optional)> <source TS> <destination TS (optional)>

Text following the hash symbol “#” is treated as comments by the interpreter.

Commands

add

The *add* command can add two timeseries together, or add a constant to a timeseries. The operands are communitive.

Usage:

add <timeries1|constant1> <timeseries2|constant2>

Examples:

The following example is equivalent to TS1 = TS2 + TS3:

```
def TS1 add TS2 TS3
```

The following example is equivalent to TS1 = TS2 + 3:

```
def TS1 add TS2 3
```

average

The *average* command steps through the specified timeseries, aggregates values according to the specified time interval and returns a timeseries with the average (mean) for each time interval.

Usage:

average <time interval> <timeseries>

Examples:

The following example averages a timeseries from the database and stores it in the variable AVG.

```
def AVG average 1d ARWO.Flow.Inst.0.0.MIXED-CCP-RAW
```

bye

The *bye* command tells the interpreter to stop what it is doing and exit back to the operating system. This command must be located at the end of all scripts. S*ee also exit*.

def

The *def* (for define) command defines a time series variable. Time series can be directly referenced in the database by specifying a path.

Usage:

def <timeseries>

Examples:

```
def numerator DWQI.Pres-Water-TotalGas.Inst.15Minutes.0.GOES-REV
def denominator DWQI.Pres-Air.Inst.15Minutes.0.GOES-REV
```

The results from computations can be stored in variables. The example below stores the resultant time series from the percent command into a variable called "TDG":

```
def TDG percent numerator denominator
```

divide

The *divide* command can divide one time series into another, divide a constant into a timeseries or divide a timeseries into a constant. The command is aliased as *div.*

Usage:

divide <timeries1|constant1> <timeseries2|contant2>

Examples:

The following example is equivalent to TS1 = TS2 / TS3:

```
def TS1 divide TS2 TS3
```

The following example is equivalent to TS1 = TS2 / 3:

```
def TS1 divide TS2 3
```

The following example is equivalent to TS3 = 3 / TS2:

```
def TS1 divide 3 TS2
```

exit

The *exit* command tells the interpreter to stop what it is doing and exit back to the operating system. This command must be located at the end of all scripts.

export

The *export* command writes a timeseries to the filesystem.

Usage:

Export <filename> <timeseries>

Examples:

Data directly from the database can be exported to the filesystem:

```
export foo.txt DWQI.Pres-Water.Inst.15Minutes.0.GOES-REV
```

Results from computations can be exported to the filesystem:

```
def BAR average 6h DWQI.Pres-Water.Inst.15Minutes.0.GOES-REV
export bar.txt BAR
```

inflow

The *inflow* command calculates inflow from a given storage and outflow using the following formula:

$$Qin = \Delta S + Qout$$

It expects input values to be in Acre-Feet and CFS. It converts Change in storage to volume by looking at the time interval between each timestep.

Usage:

inflow <storage timeseries> <outflow timeseries>

Examples:

The following example compute inflow and store it in the variable *INF*.

```
def INF inflow STOR DWR.Flow-Out.Ave.1Hour.1Hour.CBT-RAW
```

interpolate

The *interpolate* command linearly interpolates the specified timeseries on the specified interval.

Usage:

interpolate <interval> <timeseries>

Examples:

The following example will interpolate a 6hour forecast into timeseries variable called ALBO_1h with 1hour timesteps.

```
def ALBO_1h interpolate 1h  ALBO.Flow.Inst.6Hours.0.RFC-FCST
```

matchoffset

Lookback windows are calculated from the current time. This can cause inconsistencies if calculations are done on a data with intervals greater than one hour. The **matchoffset** command will match hours and minutes of the lookback window to ensure that the time windows are the same. All computations will use this offset until either the lookback window is changed or the matchoffset command is run again.

Usage:

matchoffset <timeseries>

Examples:

The following example will match the offset to the specified ~1Day path in the database:

```
matchoffset DWR.Flow-In.Ave.~1Day.1Day.CBT-RAW
```

The following example will match the offset to the specified timeseries container:

```
Matchoffset DAILY_INFLOW
```

multiply

The *multiply* command can multiply two timeseries, or multiply a constant and a timeseries. the operands are communitive.

Usage:

multiply <timeries1|constant1> <timeseries2|constant2>

Examples:

The following example is equivalent to TS1 = TS2 * TS3:

```
def TS1 multiply TS2 TS3
```

The following example is equivalent to TS1 = TS2 * 3:

```
def TS1 multiply TS2 3
```

percent

The *percent* command calculates a percentage for every matching timeseries stamp in the two timeseries specified.

Usage:

percent <numerator> <denominator>

Examples:

The following example will calculate Percent TDG from two timeseries in the database and store it in a variable called TDG.

```
def TDG percent PEKI.Pres-Water-TotalGas.Inst.1hour.0.GOES-REV PEKI.Pres-Air.Inst.1hour.0.GOES-REV
```

print

The *print* command writes the string representation of timeseries objects or internal variables to standard out. This can be used to inspect timeseries while using the computation engine console or log results while running batch jobs on a schedule (such as cron). *See also string.*

Usage:

print <timeseries|internal|string>

Examples:

The following example sets a lookback period and prints the contents of the timeseries within the lookback specified:

```
set lookback 4d
print DWR.Flow-In.Ave.~1Day.1Day.CBT-RAW
```

Example of output:

```
18-May-2012 0700        22400.00        0.00
19-May-2012 0700        20400.00        0.00
20-May-2012 0700        18200.00        0.00
21-May-2012 0700        16900.00        0.00
```

set

The *set* command allows the user to change internal variables. Internal variables influence the computations performed on timeseries, but are not timeseries themselves. Examples of internal variables are the lookback and lookforward windows. Time windows can be specified using a notation that describes days, hours and minutes of granularity. Examples:

```
set lookback 7d10h5m
set lookforward 10d
```

store

The *store* command writes a timeseries to the database to the specified path. The destination path is treated as a string literal. If a destination path does not exist in the database, it will be created. It returns a timeseries containter with the contents of what is stored to the database.

Usage:

store <source timeseries> <destination path>

Examples:

The timeseries can be a time series variable. The following example stores the contents in `SMOOTH_INF` to the database.

```
      store SMOOTH_INF DWR.Flow-In.Ave.1Hour.1Hour.CBT-COMPUTED-RAW
```

The timeseries can the result of a computation. The following example stores the 1day average of `SMOOTH_INF` to the database.

```
      store average 1d SMOOTH_INF DWR.Flow-In.Ave.~1Day.1Day.CBT-COMPUTED-RAW
```

subtract

The *subtract* command can subtract one time series from another, subtract a constant from a timeseries or subtract a timeseries from a constant. The command is aliased as *sub.*

Usage:

subtract <timeries1|constant1> <timeseries2|contant2>

Examples:

The following example is equivalent to TS1 = TS2 - TS3:

```
      def TS1 subtract TS2 TS3
```

The following example is equivalent to TS1 = TS2 - 3:

```
      def TS1 subtract TS2 3
```

The following example is equivalent to TS3 = 3 – TS2:

```
      def TS1 subtract 3 TS2
```

rate

The *rate* command looks up values from the specified timeseries and rates them using the specified rdb file. The command will look for rdb files in the directory specified in the *rdbpath* internal variable. The rate command will rate any parameter.

Usage:

rate <rdbfile> <timeseries>

Examples:

The following example turns an elevation into storage.

```
      rate DWR.rdb DWR.Elev-Forebay.Inst.1Hour.0.CBT-RAW
```

The following example converts a stage to flow and stores it in a variable called WYNW_FLOW:

```
      def WYNW_FLOW rate WYNW.rdb WYNW.Stage.Inst.15Minutes.0.GOES-RAW
```

rate2

The *rate2* command differs from the rate command in that it switches the dependent and independent variables. It looks up values from the specified timeseries and rates them using the specified rdb file. The command will look for rdb files in the directory specified in the *rdbpath* internal variable. The rate command will rate any parameter.

Usage:

Rate2 <rdbfile> <timeseries>

Examples:

The following example turns an elevation into storage.

```
      Rate2 DWR.rdb DWR.Storage.Inst.1Hour.0.CBT-RAW
```

The following example converts a flow to a stage and stores it in a variable called WYNW_STAGE:

```
      def WYNW_STAGE rate WYNW.rdb WYNW.Flow.Inst.15Minutes.0.GOES-RAW
```

rollingaverage

The *rollingaverage* command calculates a simple moving average for the previous n datapoints in the specified time interval. The timestamp for the first datapoint returned will be $T_{start}+T_{interval}$.

Usage:

rollingaverage <timeinterval> <timeseries>

Examples:

The following example returns the rolling average (simple moving average) over 6 hours for values stored in the variable *INF*.

```
      rollingaverage 6h INF
```

snap

The *snap* command attempts to make a regular interval timeseries by assigning values closest to each interval a new timestamp. The searchspace is defined by a timebuffer value. If the timebuffer is greater than interval/2 results can become inconsistent. The computation yield a warning if the timebuffer is greater than half of the interval.

Usage:

snap <interval> <time buffer> <timeseries>

Examples:

The following example snaps an irregular path in the database to a timeseries with a 1h interval. The search buffer around each interval is +- 30 minutes.

```
      snap 1h 30m ACRW.Stage.Inst.0.0.RFC-RAW
```

string

The *string* command converts all characters following the command into a string.

Usage:

string <string literal>

Examples:

The following example writes a string literal to standard out:

```
      print string Averaged hourly computed inflow
```

Example of output:

```
      Averaged hourly computed inflow
```

tablelookup

The *tablelookup* command looks up values in a table using two specified timeseries and rates them by bilinearly interpolating nearby values using the specified table.. The command will look for table files in the directory specified in the *rdbpath* internal variable. Table files can be either comma delimeted or tab delimited. This command is aliased as *bicurve.*

Usage:

tablelookup <tablefile> <timeseries1> <timeseries2>

Examples:

The following example turns a forebay elevation into storage.

```
      tablelookup sta_gate.table GATE_OPENING FOREBAY_ELEVATION
```

timeshift

The *timeshift* command shifts the timestamp of every entry in a timeseries by the specified time period. This can be used for lagging flows to compute virtual gages among other things. NOTE: you can use negative timeperiods if needed.

Usage:

timeshift < time period> <timeseries>

Examples:

The following shifts data stored in a path 6 hours into the future.

```
     def VIRT timeshift 6h ACRW.Stage.Inst.0.0.RFC-RAW
```

Example Programs

Hello World

```
print string HELLO WORLD!
exit
```

Rating storage-elevation, computing smoothed inflow

```
set lookback 1d
matchoffset DWR.Flow-In.Ave.~1Day.1Day.CBT-RAW
print string Averaged hourly computed inflow
def STOR rate DWR.rdb DWR.Elev-Forebay.Inst.1Hour.0.CBT-RAW
def INF inflow STOR DWR.Flow-Out.Ave.1Hour.1Hour.CBT-RAW
def SMOOTH_INF rollingaverage 6h INF
store SMOOTH_INF DWR.Flow-In.Ave.1Hour.1Hour.CBT-COMPUTED-RAW
store average 1d INF DWR.Flow-In.Ave.~1Day.1Day.CBT-COMPUTED-RAW
exit
```