tTimeSeries class: A short manual

This document describes the tTimeSeries class, which adds time variable parameters to the CHILD model. It has a tutorial form and first shows how to use the tTimeSeries class in a CHILD moule. Then it is shown how to add entries into the `*.in` parameter files.

For more information:

Patrick W. Bogaart
Faculty of Earth Sciences, Vrije Universiteit Amsterdam
bogw@geo.vu.nl

How to use tTimeSeries within a CHILD module

In the existing version of CHILD, parameters are read as static in time. A typical example might be:

```
double par;
par = infile.ReadItem(par,"PARNAME");
```

This case is easily converted towards the TimeSeries approach. First we introduce a tTimeSeries object which acts as a interface to the hidden TimeSeries database:

```
# include "tTimeSeries.h"

double par;
tTimeSeries *par_ts = infile.ReadTimeSeries("PARNAME");
```

This object can now be used to obtain the value of the parameter for any moment in time:

```
par = par_ts->calc(time);
```

How to create tTimeSeries compatible parameter files.

Currently, CHILD uses `*.in` files to define parameter values. Parameter names and values are on alternating lines. for example:

```
PARNAME: sample parameter
3.14
```

tTimeSeries uses a similar approach. First of all, it is backward compatible, thus the example above is also valid if PARNAME is a tTimeSeries parameter. In this case `par_ts(time)` returns 3.14 for every value of time.
This behaviour can also be set explicitly:

```
PARNAME: sample parameter
@constant 3.14
```
The keyword `@constant:` is a keyword which tells the TimeSeries system how to cope with the rst of the string. In this case, `always return a value of 3.14'.

Wave-form parameters

A more interesting case is of couse a genuine time variable parameter. say we want a parameter which behaves as a sine function with mean 2.0, period 100.0 and amplitude 1.5. We then say:

```
PARNAME: sample sine mimicing parameter
@sinwave 2.0 1.5 100.0 0.0
```

The general pattern is:

```
PARNAME: comment
@sinwave <mean> <amplitude> <period> <lag>
```

where mean is the average value of the sine wave, amplitude is half the difference between min. and max. values, period is the wave period, and lag is the horizontal shift of the wave with respect to a standard sine function. The standard sine function is thus generated by

```
PARNAME: standard sine wave
@sinwave 0.0 1.0 6.28 0.0
```

More specifically, the result which is returned by the tTimeSeries is

```
result = mean + amp*sin((time-lag)*twopi/period);
```

A variant of the SineWave function is the blockwave function, which operates exactly like the sinewave function, except that the wave swaps between *mean–average* and *mean+average*. For example:

```
PARNAME: sample bloc wave
@BLOCKWAVE 100.0 50.0 1000.0 0.0
```

results in a value of 150 for the time interval 0–500, and a value of 50 for the interval 500–1000 and so on.

Parameter values read from a file.

It is also possible to read paramter values from a file, which in it's simplest case is done like this:

```
PARNAME: example showing parameter read from a file.
@file: data.dat
```

which in combination with the file `data.dat`, having a content like

```
0.0 10.0
1.0 30.0
2.0 25.0
```

generates the following output:

10.0, for $0.0 \leq$ time < 1.0
20.0, for $1.0 \leq$ time < 2.0
25.0, for $2.0 \leq$ time

In general, time is assumed to be in the first column of the datafile, and the parameter value in the second column. If this is not the case, then columns can be set explicitly:

```
PARNAME: example showing parameter read from a file.
@file: data.dat 1 3
```

reads time from column 1 and parameter values from column 3.

By default, parameter values are not interpolated between time points. A given parameter is valid until a new time/parameter pair is given. Linear interpolation is easily obtained by appending the `interpolate` keyword:

```
PARNAME: example showing parameters read from a file.
@file: data.dat 1 3 interpolate
```

Note that the default behavioue can be explicitely set by adding the `forward` keyword.

An extra option is to indentify time and parametervalue columns by name instead of column number. If a data file a contents like:

```
%columns: time rainfall
0.0 10.0
1.0 30.0
2.0 25.0
```

then a entry in the `*.in` file like

```
PARNAME: example showing parameters read from a file.
@file: data.dat time rainfall interpolate
```

yields the epected result.
In fact, much more information can be stored within a parameter file. The following example speaks for itself:

```
%info: This is an example of a data-file.
%This is a comment. it start with an %
%
%Now a name=value parameter:
%pi=3.1415
%
%The column names are stored like this:
%columns:  time  discharge sediment_supply
%units:    [yr]  [m3/s]    [kg/s]
%
%That makes 3 columns. Note the units definition. this is not
%required
%
%now the raw data:
1970  2145.5  500.1
1971  2453.2  531.7
1972  1400.6  481.9
```

The general pattern of this version of TimeSeries definition is thus:

`@file:` *<filename>* *<time column id>* *<par column id>* *<mode>*

In-line parameter definition.

It seems awkwardy to introduce a different paramter file for each parameter whose temporal behaviour is only slighly non-trivial. We can therefore define the time/parametervalue pairs also in the `*.in` file directly:

```
PARNAME: example showing inline definition
@inline: 1970:2145.5  1971:2453.2  1972:1400.6 interpolate
```

which implements the same data as in the extended example above. Note that the colon (`:`) is used here as a time–value seperator, but a simple space works as well. It just increases the readability. The general structure of this version of TimeSeries is

`@inline:` *<time$_1$>* *<value$_1$>* <time$_2$> *<value$_2$>* `...` *<time$_n$>* *<value$_n$>* *<mode>*

Caveat: lines constructed in this way can be too long for child to handle. The maximal line length is set in the `kMaxNameLength` constant within `tInputFile.h`.

Reference

As always with complicated academic software which is forever under revision:

> *Use the source, Luke!*