sample_RES_SPX_reader

This is sample C++ code to read MFIX RES and SPx files. This could be used as a starting point for code that translates MFIX output into a different format.

Build Instructions

To build using g++

```
g++  sample_RES_SPX_reader.cpp  MfixData.cpp -o sample_RES_SPX_reader.exe
```

Usage

sample_RES_SPX_reader.exe RUN_NAME

Where the MFIX files are named: RUN_NAME.RES , RUN_NAME.SP1 , etc. The output will be in a file named RUN_NAME_info.txt (RUN_NAME as above). As written, the code will output the following information:

- Run dimensions
- Variables in the SPx files
- List of all times found in the SPx files
- For each SPX file: a list of time and C/C++ tellg() offsets into the file where the data for that time starts.
- An output of EP_g at time equal to 0.0 (the first time step). If possible, you should open this file with an editor that does not wrap lines.

General Information for Modifying the Reader for Your Own Use

1. To use the code that processes MFIX RES and SPX data, you need to include

```
#include "MfixData.h"
```

2. To create an object used to process the files

```
MfixData data;

data.SetName("RUN");    // where RUN is the MFIX run-name
```

3. To read the RES file header information

```
data.ReadRes0();
```

4. To create the variable names

```
data.CreateVariableNames();
```

5. To get all the times found in all SPX files

```
data.GetTimes();
```

6. You can examine the sample_RES_SPX_reader.cpp code and sample output located in the SAMPLE_info.txt file to get a general idea of the data members and function use of the MfixData class. Some examples:

    a. dimensions in the variables : data.imax2 , data.jmax2 , data.kmax2 , and data.ijkmax2

    b. Currently, the code does not have arrays for all the field variables, so you can only read one variable at a time

c. The following loop would iterate over all the times and read EP_g

```
int variable_index = 0;   // EP_g ... see RUN_NAME_info.txt
                          // file for a mapping between index
                          // and variable
int ijk = 234;            // just some cell for sample output
for (int ts = 0; ts<data.Get_NTIMES()-1; ++ts)
{
    // the next line reads into a std::vector<float> data.var
    data.GetVariableAtTimestep(var_index,ts);
    out << "The value of " << data.variable_names[var_index]
        << "[" << ijk << "] = " << data.var[ijk]
        << " at timestep index = " << ts << "\n";
}
```

d. Currently, if you wanted the values for both EP_g and P_g you would need to do something like this:

```
data.GetVariableAtTimestep(0,ts);
std::vector<float> EP_g = var;
data.GetVariableAtTimestep(1,ts);
std::vector<float> P_g = var;
```

7. If you look at the sample SAMPLE_info.txt file, you will notice that EP_g was written to the SPx file every 0.01 seconds, while P_g was only written every 0.1 seconds. If you request P_g at a time = 0.55 , it returns the value at time = 0.5 (it does not do time interpolation)