Ogshow: Overlapping Grid Show File Class
Saving Solutions to be Displayed with plotStuff
ShowFileReader: A Class for Reading Solutions from a Show File

User Guide, Version 1.00

Bill Henshaw

Centre for Applied Scientific Computing
Lawrence Livermore National Laboratory
Livermore, CA, 94551
henshaw@llnl.gov
http://www.llnl.gov/casc/people/henshaw
http://www.llnl.gov/casc/Overture

May 20, 2011 UCRL-MA-132235

Abstract: We describe the class Ogshow for creating show files with Overture. This class can be used in solvers in order to save solution and comments into a data base file ("show file") that can be later read by plotStuff. "plotStuff" can be used graphically display the results found in the show file.
We also describe the class ShowFileReader which can be used to read grids and solutions from a show file. The ShowFileReader can be used by PDE solvers to read in initial conditions from soluitions that have previously been saved in a show file.

Contents

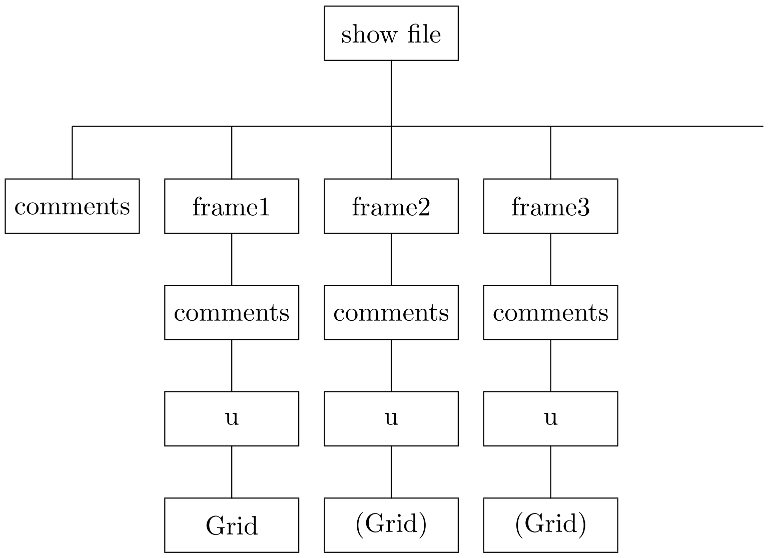

Figure 1: A show file with multiple frames, frames hold solutions at different times. For moving grids, a grid is found in each frame, otherwise the grid is only in frame 1

1 Introduction

This class can be used in solvers in order to save solution and comments into a data base file ("show file") that can be later read by plotStuff. "plotStuff" can be used graphically display the results found in the show file. The executable plotStuff is found in Overture/bin/plotStuff.

A showfile consists of a sequence of "frames". A frame will hold "items" that are related. In a typical case, there will be one frame for each time when the solution is saved. Each frame will hold one or more grid functions. Grid functions appearing in different frames but having the same name will be recognized by plotStuff. This sequence of grid functions can be used to make a "movie".

Comments can also be saved in the showfile. There are two types of comments. General comments are associated with the whole showfile and are printed out when plotStuff first reads the showfile. There are also comments that are associated with each solution in the showfile. These comments are displayed when plotStuff plots the solution.

Other things can be saved in the showfile, such as a time sequence of some values. Currently plotStuff only knows how to plot realCompositeGridFunction's although one can write a subroutine to tell plotStuff how to plot other things.

2 Class Ogshow

2.1 Interface

2.1.1 constructors

Ogshow()

Description: default constructor

Author: WDH

Ogshow(const aString & nameOfShowFile,
const aString & nameOfDirectory = ".",
int useStreamMode =false,
ShowFileOpenOption openOption = openNewFileForWriting)

Description: construct a show file

nameOfShowFile (input) : name of the new show file to create

nameOfDirectory (input) : directory in the Overlapping grid data base file to use

useStreamMode (input): if true, save file in streaming mode (compressed)

openOption (input) : specifies whether to open a new file for writing (openNewFileForWriting) or (openOldFileForWriting) open an old file to append to.

Author: WDH

2.1.2 close

int
close()

Description: Close a show file.

2.1.3 cleanup

int
cleanup()

Access: protected

Description: Close and cleanup the show file.

2.1.4 open

int
open(const aString & nameOfShowFile,
  const aString & nameOfDirectory = ".",
  int useStreamMode =false,
  ShowFileOpenOption openOption = openNewFileForWriting)

Description: Open a show file (close any currently open file).

nameOfShowFile (input) : name of the new show file to create

nameOfDirectory (input) : directory in the Overlapping grid data base file to use

useStreamMode (input): if true, save file in streaming mode (compressed)

openOption (input) : specifies whether to open a new file for writing (openNewFileForWriting) or (openOldFileForWriting) open an old file to append to.

2.1.5 getFrame

HDF_DataBase*
getFrame(const FrameSeriesID /*= 0*/)

Description: Return a pointer to the data base directory holding the current frame. You could use this pointer to save additional data in the frame. In the following example some extra data in the form of a realArray is saved in the frame.

```
Ogshow show(...);
...
show.startFrame();
realArray myData(10);
myData(0)=1.; myData(1)=2.; ...
show.getFrame()->put(myData,"my data");
...
```

This data can be retrieved using the ShowFileReader.

Return value: Return a pointer to the data base directory holding the current frame, possibly NULL.

Author: WDH

2.1.6 setFlushFrequency

void
setFlushFrequency(const int flushFrequency = 5)

Description: Flush the file every time "flushFrequency" frames have been added. In the current implementation "flushing the file" consists of closing the file and opening a new file to save new frames in.

flushFrequency (input): If positive then the file is "flushed" when every time this many new frames have been added.

Author: WDH

2.1.7 getFlushFrequency

int
getFlushFrequency() const

Description: Return the flush frequency.

2.1.8 isFirstFrameInSubFile

bool
isFirstFrameInSubFile() const

Description: Return true if the current frame is the first frame in the current subFile (subfile's are named fileName.show, fileName.show1, fileName.show2, ...) frame will go into a new sub-file.

2.1.9 isLastFrameInSubFile

bool
isLastFrameInSubFile() const

Description: Return true if the current frame is the last frame in the current subFile (subfile's are named fileName.show, fileName.show1, fileName.show2, ...) frame will go into a new sub-file.

2.1.10 setMovingGridProblem

bool
getIsMovingGridProblem() const

Description: Return true if this is a moving grid problem.

Author: WDH

2.1.11 setMovingGridProblem

void
setIsMovingGridProblem(const bool trueOrFalse)

Description: Indicate if this is a moving grid problem so that the grid is saved in every frame

trueOrFalse (input): TRUE is this is a moving grid problem

Author: WDH

2.1.12 getNumberOfFrames

int
getTotalNumberOfFrames() const

Description: return the number of frames that exist in the show file.

Author: KKC

2.1.13 getNumberOfFrames

int
getNumberOfFrames() const

Description: return the number of frames that exist in a particular frame series

Author: WDH

2.1.14 getShowFileName

const aString &
getShowFileName() const

Description: return the name of the show file.

Author: WDH

2.1.15 startFrame

int
startFrame(const int frameNo = newFrame)

Description: start a new frame or write to an existing one

frameNo (input): by default start a new frame, otherwise open a frame with the given value.

Author: WDH

2.1.16 endFrame

int
endFrame()

Description: End the currently open frame (if any). The main purpose of calling this routine is to close a sub-file if this was the last frame in the sub-file. This will allow the sub-file to be read programs such as plotStuff. WARNING: once a sub-file is closed you can no longer write to a frame in that sub-file. This needs to be fixed.

Author: WDH

2.1.17 saveGeneralComment

int
saveGeneralComment(const aString & comment0)

Description: Save a general comment (this comment is associated with the entire show file). Multiple comments can be saved by repeatedly calling this function.

comment0 (input): comment to save.

Author: WDH

2.1.18 saveComment

int
saveComment(const int commentNumber, const aString & comment0)

Description: Save a comment to go in the current frame.

commentNumber (input): An integer, 0,1,2,.. that numbers the comment

comment0 (input): comment to save.

Author: WDH

2.1.19 saveGeneralParameters

int
saveGeneralParameters(ListOfShowFileParameters & params, const PlaceToSaveGeneralParameters placeToSave)

Description: Save parameters that apply to the whole file.

params (input): A list of parameters to save

placeToSave (input) : save in the root directory (THEShowFileRoot) or in the current frame (THECurrentFrameSeries).

Author: WDH

2.1.20 saveGeneralParameters

// Save a named set of parameters
int
saveParameters(const aString & nameOfDirectory, ListOfShowFileParameters & params)

Description: Save a named set of parameters in the current frame.

nameOfDirectory (input) : the name of the (new) directory where to save the parameters

params (input): A list of parameters to save

Author: WDH

2.1.21 saveSolution

```
int
saveSolution(realMappedGridFunction & u,
             const aString & name = "u",
             int frameForGrid = useDefaultLocation)
```

Description: Save a mappedGridFunction in the current frame. (for now save a CompositeGridFunction)

u (input) : grid function to save

name (input): save in the frame under this name. (Currently if you change this name from the default then plotStuff will not find the solution).

frameForGrid : indicates where in the show file the grid for this solution can be found. This grid will saved in this frame if it does not already exist. useDefaultLocation : use default location (frame 1), useCurrentFrame : current frame, > 0 : specify a frame number.

Author: WDH

2.1.22 saveSolution

int
saveSolution(realGridCollectionFunction & u_,
const aString & name = "u",
int frameForGrid = useDefaultLocation)

Description: Save a realGridCollectionFunction or realCompositeGridFunction in the current frame.

u (input) : grid function to save

name (input): save in the frame under this name. (Currently if you change this name from the default then plotStuff will not find the solution).

frameForGrid : indicates where in the show file the grid for this solution can be found. This grid will saved in this frame if it does not already exist. useDefaultLocation : use default location (frame 1), useCurrentFrame : current frame, > 0 : specify a frame number.

Author: WDH

2.1.23 newFrameSeries

FrameSeriesID
newFrameSeries(const aString & name)

Description: Create a new frame series with the given name.

name (input) : name of a new frame series

Return value: the frameSeriesID

2.1.24 setCurrentFrameSeries

int
getNumberOfFrameSeries() const

Description: Return the number of frame series in this show file.

2.1.25 getFrameSeriesID

FrameSeriesID
getFrameSeriesID(const aString & name)

Description: Return the FrameSeriesID corresponding to a given name.

name (input) : name of an existing frame series

Return value: -1 if the name was not found.

2.1.26 getFrameSeriesName

const aString&
getFrameSeriesName(const FrameSeriesID frameSeries)

Description: Return the name of a frame series in this show file.

frameSeries (input) : a frame series ID.

Return value: A nullString if the frameSeries was not found.

2.1.27 setFrameSeriesName

int
setFrameSeriesName(const FrameSeriesID frameSeries, const aString & name)

Description: Assign a new name to an existing frame series.

frameSeries (input) : a frame series ID.

name (input) : name for the frame series

Return value: 0=success, 1=failure.

2.1.28 getCurrentFrameSeries

FrameSeriesID
getCurrentFrameSeries() const

Description: Return the number of current frame series in this show file.

2.1.29 setCurrentFrameSeries

int
setCurrentFrameSeries(const FrameSeriesID frameSeries)

Description: Set the current frame series to the given ID.

frameSeries (input) : a frame series ID.

Return value: 1 if the frameSeries is not valid.

2.1.30 setCurrentFrameSeries

FrameSeriesID
setCurrentFrameSeries(const aString & name)

Description: Set the current frame series to be ”name”. Create a new frame series with this name if it does not already exist.

2.2 Typical Usage

Consider an example where a solver is computing some velocity field (u, v). The user would like to save u, v and the “Mach number” $u^2 + v^2$ in the show file. Here is how this might be done: (file “togshow.C”)

```
//================================================================================
//  Test the Overlapping Grid Show file class Ogshow
//
// Examples:
//   togshow -g=cic.hdf -show=cic.show
//  -- append to an old file:
//   togshow -g=cic.hdf -show=cic.show -append
//===============================================================================
#include "Overture.h"
#include "Ogshow.h"
#include "HDF_DataBase.h"

int
main(int argc, char *argv[])
{
  Overture::start(argc,argv);

  aString nameOfOGFile="cic.hdf", nameOfShowFile="cic.show";

  printF("Usage:togshow -g=gridName -show=showFileName -append\n");
  Ogshow::ShowFileOpenOption showFileOpenOption = Ogshow::openNewFileForWriting;

  if( argc>1 )
  {
    aString line;
    int len=0;
    for( int i=1; i<argc; i++ )
    {
      line=argv[i];
      if( len=line.matches("-g=") )
      {
        nameOfOGFile=line(len,line.length()-1);
      }
      else if( len=line.matches("-show=") )
      {
        nameOfShowFile=line(len,line.length()-1);
      }
      else if( line=="-append" )
      {
        showFileOpenOption=Ogshow::openOldFileForWriting;
      }
    }
  }

//   #ifndef USE_PPP
//   if( argc>1 )
//     nameOfOGFile=argv[1];
//   if( argc>2 )
//     nameOfShowFile=argv[2];
//   #endif

//   if( nameOfOGFile=="" )
//   {
//     cout << "togshow>> Enter the name of the (old) overlapping grid file:" << endl;
//     cin >> nameOfOGFile;
//   }
//   if( nameOfShowFile=="" )
//   {
//     cout << "togshow>> Enter the name of the (new) show file (blank for none):" << endl;
//     cin >> nameOfShowFile;
//   }

  CompositeGrid cg;
  getFromADataBase(cg,nameOfOGFile);          // read from a data base file
  cg.update(MappedGrid::THEmask | MappedGrid::THEvertex );

  // HDF_DataBase::debug=1;

  int useStreamMode=false; // true;

  Ogshow show(nameOfShowFile,".",useStreamMode,showFileOpenOption);  // create a show file

  show.saveGeneralComment("Solution to the Navier-Stokes"); // save a general comment in the show file
  show.saveGeneralComment(" file written on April 1");      // save another general comment

  Range all;
  realCompositeGridFunction q(cg,all,all,all,3); // create a grid function with 3 components
//   realCompositeGridFunction u,v,machNumber;  // create grid functions for components
//   u.link(q,Range(0,0));                              // link u to the first component of q
//   v.link(q,Range(1,1));                              // link v to the second component of q
//  machNumber.link(q,Range(2,2));                      // ...

  // save the names of components,  first name is the name of the vector
  q.setName("q");                            // assign name to grid function and components
  q.setName("u",0);                          // name of first component
  q.setName("v",1);                          // name of second component
  q.setName("T",2);                          // name of third component

  char buffer[80];                           // buffer for sprintf
  int numberOfTimeSteps=3;
  int flushFrequency=1; // 2;
//  cout << "Enter number of steps and the flush frequency" << endl;
//  cin >> numberOfTimeSteps >> flushFrequency;

```

```
  show.setFlushFrequency(flushFrequency);

  for( int i=1; i<=numberOfTimeSteps; i++ )  // Now save the grid functions at different time steps
  {
    show.startFrame();                       // start a new frame
    real t=i*.1;
    show.saveComment(0,sPrintF(buffer,"Here is solution %i",i));              // comment 0 (shown on plot)
    show.saveComment(1,sPrintF(buffer,"  t=%e ",t));              // comment 1 (shown on plot)

    for( int grid=0; grid<cg.numberOfComponentGrids(); grid++ )
    {
      const realArray & x = cg[grid].vertex()(all,all,all,0);
      const realArray & y = cg[grid].vertex()(all,all,all,1);

      real fx=Pi*i/numberOfTimeSteps;

      q[grid](all,all,all,0)=sin(fx*x)*cos(fx*y);
      q[grid](all,all,all,1)=cos(fx*x)*cos(fx*y);
      q[grid](all,all,all,2)=sin(fx*x)*sin(fx*y);
    }
    show.saveSolution( q );              // save the current grid function

    if( false )
    { // testing:
      show.saveSolution( q,"v" );  // save under a different name (these are not currently found by plotStuff)
    }

    show.getFrame()->put(t,"t");         // save some extra info using data base functions

    printF("save solution %i\n",i);

    // if( ( (i % flushFrequency) ==0 ) || i==numberOfTimeSteps )
    if( show.isLastFrameInSubFile() )
    {
      printf(" -- save seq info at step=%i --- \n",i);

      const int n=10*i;
      RealArray time(n);
      time.seqAdd(0,1./(n-1));
      RealArray value(n);
      value=sin(2.*Pi*time);
      show.saveSequence( "my seq",time,value);

      value=sin(Pi*time);
      show.saveSequence( "my next seq",time,value);
    }

    if( true ) show.endFrame();

    if( i==numberOfTimeSteps-3 ) Overture::abort("error");

  }

  show.close();

  Overture::finish();

  return 0;
}
```

In this example we also save some additional information into the data base (the time "t"). The function `getFrame` returns a pointer to the `HDF_DataBase` directory in which the current frame is being saved. The data base function `put` is then used to save a real number. It is also possible to save arrays and Strings in this way. See the data base documentation for further details.

2.3 Moving grids

The only difference with moving grids is that one should say `show.setMovingGridProblem(TRUE)`. In this case each frame will hold a new grid as well as the solution.

See the moving grid example in the primer documentation for an example of saving a moving grid in a show file.

3 ShowFileReader

The ShowFileReader class can be used to read in grids and solutions from a show file. This class can be used, for example, to read in initial conditions for a solver.

3.1 Interface

3.1.1 constructor

ShowFileReader(const aString & showFileName = nullString)

Description: Create an object for reading show files (generated by Ogshow) and optionally supply the name of the show file to open. A show file can also be opened with the `open` member function.

showFileName (input): name of an existing show file (if specified).

Author: WDH

3.1.2 close

int
close()

Description: Close an open show file.

Author: WDH

3.1.3 getNumberOfFrames

int
getNumberOfFrames() const

Description: Returns the number of frames that exist in the show file for the furrent frame series.

Author: WDH

3.1.4 getNumberOfSolutions

int
getNumberOfSolutions() const

Description: Returns the number of Solutions that exist in the show file for the furrent frame series.

Author: WDH

3.1.5 getNumberOfSequences

int
getNumberOfSequences() const

Description: Returns the number of Sequences that exist in the show file for the furrent frame series.

Author: WDH

3.1.6 setMaximumNumberOfOpenShowFiles

void
setMaximumNumberOfOpenShowFiles(const int maxNumber)

Description: For very large files we may have to reduce the number of files that we allow to be open at any time

3.1.7 getAGrid

ReturnType
getAGrid(MappedGrid & mg,
int & solutionNumber,
int frameForGrid =useDefaultLocation)

ReturnType
getAGrid(GridCollection & cg,
int & solutionNumber,
int frameForGrid =useDefaultLocation)

Description: Get grid GridCollection or CompositeGrid from a show file. If this a moving grid problem then return the grid corresponding to a give solutionNumber.

cg (output): The grid corresponding to the solution numbered `solutionNumber`. This routine will always read in a new grid if a grid is found.

solutionNumber (input): For moving grid problems only. Find the grid corresponding to this solution number. This number should be in the range [1,numberOfSolutions], where numberOfSolutions is the value by getNumberOfSolutions() If solutionNumber is out of range then the closest valid solution number is chosen, and this value is returned in solutionNumber. Thus if you want to get the last grid in the file choose solutionNumber to be any integer that is larger than the number of solutions in the file.

frameForGrid : indicates where in the show file the grid for this solution can be found. useDefaultLocation : use default location (frame 1 for non-moving grids or the current frame for moving grids), useCurrentFrame : current frame, > 0 : specify a frame number.

return values: notFound or gridFound.

Author: WDH

3.1.8 getHeaderComments

const aString*
getHeaderComments(int & numberOfHeaderComments0)

Description: Get header comments for the last grid or solution that was found

numberOfHeaderComments0 (output): The number of comments in the array of Strings

return value: An array of aString's with the comments that are associated with this solution. (You might use the declaration `const aString *headerComment`).

Author: WDH

3.1.9 getASolution

ReturnType
getASolution(int & solutionNumber,
MappedGrid & mg,
realMappedGridFunction & u)

ReturnType
getASolution(int & solutionNumber,
GridCollection & cg,
realGridCollectionFunction & u)

Description: Get grid (GridCollection or CompositeGrid) and a grid function (realGridCollectionFunction or realCompositeGridFunction) from a show file.

solutionNumber (input/ouptut): The number of the solution to get. This number should be in the range [1,numberOfSolutions], where numberOfSolutions is the value by getNumberOfSolutions() If solutionNumber is out of range then the closest valid solution number is chosen, and this value is returned in solutionNumber. Thus if you want to get the last solution in the file choose solutionNumber to be any integer that is larger than the number of solutions in the file.

cg (input/output): The grid corresponding to the solution numbered `solutionNumber`. The grid cg will only be changed under certain circumstances. The grid cg will be created or changed in the following cases:

- cg is a null grid on input.
- The show file contains moving grids and solutionNumber is not equal to the solutionNumber of the last time this routine was called.

u (output): The grid function corresponding to the solution numbered `solutionNumber`

return values: notFound or gridFound or solutionFound or solutionAndGridFound.

Author: WDH

3.1.10 getFrame

HDF_DataBase*
getFrame(int solutionNumber = -1)

Description: Return a pointer to the data base directory holding a frame; by default the current frame. You could use this pointer to get any additional data that has been saved in the frame. The current frame for all other calls is also set to the requested frame number. In the following example some extra data in the form of a realArray is retrieved.

```
ShowFileReader show(...);
...
show.getASolution(...);
realArray myData;
show.getFrame()->get(myData,"my data");
...
```

solutionNumber (input): get the frame for this solution number. If no argument is specified then return the current frame.

Return value: Return a pointer to the data base directory holding the frame, possibly NULL.

Author: WDH

3.1.11 getSequenceNames

int
getSequenceNames(aString *name, int maximumNumberOfNames)

Description: Return the names of the sequences, up to a maximum of maximumNumberOfNames,

Return value:

Author: WDH

3.1.12 getSequence

int
getSequence(int sequenceNumber,
aString & name, RealArray & time, RealArray & value,
aString *componentName1, int maxComponentName1,
aString *componentName2, int maxComponentName2)

Description: Return the data for a sequence.

name (output) : name of the sequence.

time (output) : time(0...n-1) - array of n 'time' values or other iteration variable.

value (output) : value(0...n-1,0..m-1) array of n values for each of m components.

componentName1 (output) : name1[0..m-1] name for the components.

maxComponentName1 (input) : maximum number of array elements in the array componentName1.

componentName2 (output) : names for a second level of components BUT DO NOT USE THIS for now.

maxComponentName2 (input) : maximum number of array elements in the array componentName2.

Return value: 0 for success.

Author: WDH

3.1.13 getGeneralParameters

int
getGeneralParameters(const int displayInfo =1)

Description: Get the list of parameters that go with this file.

displayInfo (input) : if not equal to zero output info about the parameters in the file.

return value:

Author: WDH

3.1.14 getGeneralParameter(int)

bool
getGeneralParameter(const aString & name, int & ivalue,const PlaceToSaveGeneralParameters placeToSave)

Description: Get a general parameter with type 'int'

return value:

Author: WDH

3.1.15 getGeneralParameter(int)

bool
getGeneralParameter(const aString & name, real & rvalue,const PlaceToSaveGeneralParameters placeToSave)

Description: Get a general parameter with type 'int'

return value:

Author: WDH

3.1.16 getGeneralParameter(int)

bool
getGeneralParameter(const aString & name, aString & stringValue,const PlaceToSaveGeneralParameters placeToSave)

Description: Get a general parameter with type 'int'

return value:

Author: WDH

3.1.17 getGeneralParameter(int)

bool
getGeneralParameter(const aString & name, ParameterType type, int & ivalue, real & rvalue, aString & stringValue,const Ogshow::PlaceToSaveGeneralParameters placeToSave)

Description: Get a general parameter with the given name and type

name, type (input) : get the parameter value for a parameter with this name and type.

ivalue (output) : return integer parameters in this variable (if type==intParameter)

rvalue (output) : return real parameters in this variable (if type==realParameter)

stringValue (output) : return string parameters in this variable (if type==stringParameter)

return value:

Author: WDH

3.1.18 getListOfGeneralParameters

ListOfShowFileParameters&
getListOfGeneralParameters(const Ogshow::PlaceToSaveGeneralParameters placeToSave =Ogshow::THECurrentFrameSeries)

Description: Return the general parameters

placeToSave (input): define which list to use.

return value: The list of show file parameters

Author: WDH

3.1.19 getParameters

bool
getParameters(const aString & nameOfDirectory, ListOfShowFileParameters & params)

Description: Get parameters from a given directory

nameOfDirectory (input) : look for the parameters in this directory.

params (input): The parameters are returned here.

return value:

Author: WDH

3.1.20 isAMovingGrid

bool
isAMovingGrid()

Description: Return TRUE if this is a moving grid problem

return value: Return TRUE if this is a moving grid problem

Author: WDH

3.1.21 open

int
open(const aString & showFileName, const int displayInfo =1)

Description: Open a show file that was generated by Ogshow.

showFileName (input): name of an existing show file.

displayInfo (input) : if not equal to zero output info about the parameters in the file.

Author: WDH

3.1.22 getNumberOfFrameSeries

int
getNumberOfFrameSeries() const

Description: Return the number of frame series in the show file.

Return value: 0 for success.

Author: KKC

3.1.23 getFrameSeriesName

aString
getFrameSeriesName(const FrameSeriesID frame_series)

Description: Return the string name of a particular frame series

frame_series (input) : integer specifying the frame series

Return value: the name of frame_series on success, an null string ("") on failure

Author: WDH

3.2 Example: Using ShowFileReader to Read in Initial Conditions to a PDE Solver

In this example we show how to mount a show file and read a grid and solution from the show file. We also show how to interpolate a solution on one CompositeGrid to a solution on another CompositeGrid using the `interpolateAllPoints` function. The `interpolateAllPoints` function is described in more detail in the GridFunction documentation.

This example shows how one could read initial conditions for a PDE solver from a show file. The CompositeGrid used by the PDE solver does not have to be the same as the CompositeGrid in the showFile. For example, the grid may be refined or a new component grid added (file `Overture/examples/readShowFile.C`).

```
//================================================================================
//  Test the ShowFileReader
//    o read a solution and grid from a show file
//    o interpolate the solution from one grid to another grid
//===============================================================================
#include "Overture.h"
#include "Ogshow.h"
#include "ShowFileReader.h"
#include "interpPoints.h"
#include "GL_GraphicsInterface.h"

#include <sys/resource.h>

int
main(int argc, char *argv[])
{
  Overture::start(argc,argv);



  aString nameOfShowFile;
  cout << ">> Enter the name of the (old) show file:" << endl;
  cin >> nameOfShowFile;
  ShowFileReader showFileReader(nameOfShowFile);

  int numberOfFrames=showFileReader.getNumberOfFrames();
  int numberOfSolutions = max(1,numberOfFrames);
  int solutionNumber;

  CompositeGrid cg;
  realCompositeGridFunction u;

  GL_GraphicsInterface ps;          // create a GL_GraphicsInterface object
  GraphicsParameters psp;           // create an object that is used to pass parameters

  aString answer,answer2;
  aString menu[] = { "get a solution",
                     "interpolate to new grid",
                     "exit",
                     "" };
  const aString *headerComment;
  int numberOfHeaderComments;
  char buff[80];

  for(;;)
  {
    ps.getMenuItem(menu,answer);
    if( answer=="get a solution" )
    {
      // In this case the user is asked to choose a solution to read in
      // Choosing a number that is too large will cause the last solution to be read

      aString line;
      ps.inputString(line,sPrintF(buff,"Enter the solution number, [1,%i] \n",numberOfSolutions));
      sscanf(line,"%i",&solutionNumber);

      showFileReader.getASolution(solutionNumber,cg,u);        // read in a grid and solution

      // read any header comments that go with this solution
      headerComment=showFileReader.getHeaderComments(numberOfHeaderComments);

      for( int i=0; i<numberOfHeaderComments; i++ )
```

```
        printf("Header comment: %s \n",(const char *)headerComment[i]);

      psp.set(GI_TOP_LABEL,sPrintF(buff,"solution number %i",solutionNumber));
      ps.erase();
      PlotIt::contour(ps,u,psp);
    }
    else if( answer=="interpolate to new grid" )
    {
      // In this case we read a solution from the showFile and save it in the grid cg and grid function u.
      // We then read in a different CompositeGrid, cgTo, and create a grid function uTo. We get values
      // on uTo by interpolating from u
      solutionNumber=1;
      showFileReader.getASolution(solutionNumber,cg,u);

      aString nameOfOGFile;
      CompositeGrid cgTo;
      ps.inputString(nameOfOGFile,">> Enter the name of the CompositeGrid file (to interpolate to):");

      getFromADataBase(cgTo,nameOfOGFile);          // read from a data base file
      cgTo.update();
      Range all;
      realCompositeGridFunction uTo(cgTo,all,all,all,10);

      interpolateAllPoints( u,uTo );  // interpolate uTo from u

      psp.set(GI_TOP_LABEL,sPrintF(buff,"interpolated solution at number %i",solutionNumber));
      ps.erase();
      PlotIt::contour(ps,uTo,psp);
    }
    else if( answer=="exit" )
    {
      break;
    }
  }

  Overture::finish();
  return 0;
}
```

Index