Los Alamos
NATIONAL LABORATORY
memorandum

Computer and Computational Sciences Division

CCS-4: Transport Methods Group

To/MS: Distribution
From/MS: Kelly Thompson, MS D409
Phone/Fax: (505) 665-3929 / (505) 665-5538
E-Mail: kgt@lanl.gov
Symbol: CCS-4:01-05(U)
Date: February 12, 2001

Subject: Gandolf Opacity Package for DRACO

1 Introduction

The `cdi_gandolf` package has been introduced into the DRACO project to provide a solution to two issues. First, it provides a C++ object based interface to IPCRESS (Independent of Platform and Can be Read by existing Software Subroutines) opacity data. These data files are produced by the TOPS code via the FORTRAN77 Gandolf (Get All Needed Data from an Opacity Laden File) libraries. Second, it acts as a plug in component for DRACO's Common Data Interface (CDI) package.

This package consists of five classes and a C++ wrapper for the Gandolf vendor library.

Class	Role
GandolfWrapper	Provide C++ style access to the Gandolf library functions ensuring correct data type translation.
GandolfFile	Provide access to the fundamental information about an IPCRESS file. The two GandolfOpacity objects only access the data file through this object.
GandolfException	Provide detailed information about and handle any exception thrown by the Gandolf wrapper or library.
GandolfDataTable	Cache the opacity lookup table provided by the IPCRESS file. This is a component object of the two GandolfOpacity objects.
GandolfGrayOpacity	Provide access to gray opacity data for a single material found in the IPCRESS file.
GandolfMultigroup-Opacity	Provide access to multigroup opacity data for a single material found in the IPCRESS file.

2 GandolfFile

2.1 Instantiation

The GandolfFile object provides GandolfOpacity objects a path for interaction with an IPCRESS opacity data file. A GandolfFile object only requires the name of a valid IPCRESS file for instantiation. Its state consists of this filename plus a list of material identifiers read from the IPCRESS file. See Figure 1 for an example.

```
#include “GandolfFile.hh”
#include “GandolfException.hh”

main()
{
      // Specify the name of the IPCRESS data file.
      const string filename = “data.ipcress”;

      // Instantiate the object within a try block.
      try
            {
                  spGF = new rtt_cdi_gandolf::GandolfFile( filename );
            }
      catch (rtt_cdi_gandolf::GandolfException gerr )
            {
                  // Instantiation failed.
                  cout << gerr.errorSummary() << endl;
                  return 1;
            }
      return 0;  // Success.
}
```

Figure 1. Example of instantiating a GandolfFile object.

2.2 Access Functions

GanolfFile provides four access functions. These functions return simple information about the data found in the IPCRESS file.

Member Function	Role
`const string& getDataFilename()`	The name of the IPCRESS file.
`int getNumMaterials()`	The number of materials found in the IPCRESS file.
`const vector<int>& getMatIDs()`	A list of materials found in the IPCRESS file.
`bool materialFound( int matid )`	Returns true if material identifier `matid` exists in the IPCRESS file.

2.3 Typical Use

Normally, GandolfFile is only used as a hook to interface a GandolfOpacity object to an IPCRESS file. The member functions for GandolfFile will not need to be invoked by the user, but GandolfOpacity (Gray or Multigroup) will use them. The GandolfFile object must be instantiated prior to the instantiation of a GandolfOpacity object. Note that there should be a one-to-one correspondence between the GandolfFile object and an IPCRESS file. Multiple GandolfOpacity objects will reference the same GandolfFile object.

3 GandolfException, GandolfWrapper and GandolfDataTable

There is no requirement for the client code to interact with any of these classes. GandolfDataTable is a component of the GandolfOpacity object. It will be instantiated automatically when needed and while it interacts with other `cdi_gandolf` components the user does not need to interact directly with the GandolfDataTable object. There is always and one-to-one relationship between GandolfOpacity and GandolfDataTable.

The GandolfWrapper routines provide access to the FORTRAN77 Gandolf libraries. This set of routines act as the interface between the FORTRAN library and the C++ objects that make up the `cdi_gandolf` package. A client code could theoretically use the functions found in GandolfWrapper to access the FORTRAN Gandolf libraries directly without using the rest of the package. However, the design of this package did not have this approach in mind.

The client code will have some indirect interaction with the GandolfException object if the client wants to intercept exceptions thrown by various routines in this package. The client code should include a try block similar to the one shown in Figure 1 when instantiating either a GandolfFile object or a GandolfOpacity object.

4 GandolfGrayOpacity and GandolfMultigroupOpacity

These two classes provide very similar interfaces to IPCRESS opacity data. The only difference is that while GandolfGrayOpacity considers an opacity value to be a single double (i.e.: a single gray value), the GandolfMultigroupOpacity class considers an opacity value to be a vector of doubles (a value for each energy group). These objects inherit their interface from the Opacity interface definitions declared in the `cdi` package (see cdi/GrayOpacity.hh and cdi/MultigroupOpacity.hh).

4.1 Instantiation

The state of a GandolfOpacity (either gray or multigroup) object is determined by five pieces of information. All of these options must be explicitly defined during the instantiation of a GandolfOpacity object.

Required Information	Role
Smart pointer to a GandolfFile object	Specifies the associated IPCRESS data file.
Material Identifier	A GandolfOpacity object must represent a single material whose identity is specified by an integer value. This material must be found in the associated IPCRESS data file.
Opacity Model	A GandolfOpacity object must contain either Rosseland or Plank data.
Reaction Model	A GandolfOpacity object must represent the total, absorption-only or scattering-only opacities.
Energy Model	This is specified by instantiating either a GandolfGrayOpacity or a GandolfMultigroupOpacity object.

The GandolfFile object must already exist (see Section 2 of this document). The creator routine expects a DRACO smart pointer to this object (i.e.: `rtt_dsxx::SP<GandolfFile>`).

The material identifier is an integer that matches the material identifier of one of the materials found in the IPCRESS file. The user should be aware of the contents of the IPCRESS file (including knowing what materials exist in the file and knowing the identifier for each material). There is no mechanism built into this package for querying the IPCRESS file for material information.

The opacity model ("Rosseland" or "Plank") is an enumerated value defined in DRACO's `cdi` package. The user must specify what type of opacity information he/she wants this object to return.

The reaction model is also an enumerated value defined by the `cdi` package. Valid values are "total", "scattering", or "absorption". A GandolfOpacity object can only access a single type of opacity. If both "total" and "scattering" opacities are required by the client code then two distinct GandolfOpacity objects will need to be instantiated.

The energy model will be automatically specified when the client code chooses to instantiate either a GandolfGrayOpacity object or a GandolfMultigroupOpacity object.

An example of instantiating a GandolfOpacity object is shown in Figure 2.

```
#include “GandolfFile.hh”
#include “GandolfException.hh”

main()
{
   // Assume that we have already instantiated a GandolfFile object
   // named spGF. (see Figure 1 above for details.)

   // Assign a material identifier
   const int matid = 10001;

   // Declare a smart pointer to and GandolfGrayOpacity.  We use the
   // interface defined by rtt_cdi:GrayOpacity.
   rtt_dsxx::SP< rtt_cdi::GrayOpacity > spOpacity_GraRosTot;

   // Use a try block when instantiating the GrayOpacity object so that
   // we can catch any errors.
   try
      {
         spOpacity_GraRosTot = new
            rtt_cdi_gandolf::GandolfGrayOpacity(
               spGF,                // All data is loaded from this IPCRESS
                                    // file.
               matid,               // This object can only return information
                                    // about one material.
               rtt_cdi::Rosseland,  // All opacity values use the Rosseland
                                    // model.
               rtt_cdi::Total );    // All opacity values are total cross
                                    // sections.
      }
   catch ( rtt_cdi_gandolf::GandolfException gerr )
      {
         // Instantiation failed so print a report and exit.
         cout << gerr.errorSummary();
         return –1;
      }

   // Instantiation was successful.
   return 0;
}
```

Figure 2. Example of instantiating a GandolfGrayOpacity object.

4.2 Access Functions

The GandolfOpacity classes have been designed as a mechanism to obtain logarithmically interpolated opacity data from an IPCRESS file. As such the primary access function of GandolfGrayOpacity and GandolfMultigroupOpacity is the getOpacity() accessor. Other functions are also available that allow the user to query the opacity lookup table and to obtain text descriptions of the data represented by this object. Examples demonstrating the use of these accessor functions can be found in the next subsection or in the unit test “cdi_gandolf/test/tGandolfOpacity.”

Some of the more commonly used access functions for GandolfGrayOpacity are listed in the table below. These access functions expect temperatures to be given in keV and densities to given in g/cm^3. The retuned opacity values have units of cm^2/g.

GandolfGrayOpacity Member Functions	**Role**
`double getOpacity( double temp, double density )`	Given the temperature and density information, return a logarithmically interpolated opacity value. *The type of opacity returned will correspond to GandolfGrayOpacity's state (e.g.: Rosseland, absorption-only).*
`vector<double> getOpacity( vector<double> temp, double density )`	Return a list of opacities corresponding to the given density and each of the given temperatures.
`vector<double> getOpacity( double temp, vector<double> density )`	Returns a list of opacities corresponding to the given temperature and each of the given densities.
`const vector<double>& getTemperatureGrid()`	Access the opacity lookup table.
`const vector<double>& getDensityGrid()`	Access the opacity lookup table.

GandolfMultigroupOpacity has a similar set of functions. The primary difference is that GandolfMultigroupOpacity returns a list of multigroup opacity values instead of a single opacity value. It should be noted that the functions that access the lookup table return references to the actual table data. The opacity accessors return a copy of the data and not a reference. Also, the number of groups in the data table is set by TOPS in the IPCRESS file.

GandolfMultigroupOpacity Member Functions	Role
`vector<double> getOpacity( double temp, double density )`	Given the temperature and density information, return a vector of logarithmically interpolated opacity values. The length of this vector will be equal to the number of groups in the data set. *The type of opacity returned will correspond to GandolfGrayOpacity's state (e.g.: Rosseland, absorption-only).*
`vector< vector<double> > getOpacity( vector<double> temp, double density )`	Return a list of multigroup opacities corresponding to the given density and each of the given temperatures.
`vector< vector<double> > getOpacity( double temp, vector<double> density )`	Returns a list of multigroup opacities corresponding to the given temperature and each of the given densities.
`const vector<double>& getTemperatureGrid()`	Access the opacity lookup table grid.
`const vector<double>& getDensityGrid()`	Access the opacity lookup table grid.
`cont vector<double>& getGroupBoundaries()`	Access the opacity lookup table grid.

In Addition to these *standard* access functions, both GandolfOpacity objects have STL-like accessor routines. These accessors can be used if the desired opacity container is not a `vector<double>`. The opacity container must be a 1-D container for both GandolfGrayOpacity and GandolfMultigroupOpacity.

```
#include “GandolfMultigroupOpacity.hh”
#include “GandolfMultigroupOpacity.t.hh”
#include “GandolfFile.hh”
#include “GandolfException.hh”

main()
{
   // Assume that we have already instantiated a GandolfFile object
   // named spGF. (see Figure 1 above for details.)

   // Assign a material identifier
   const int matid = 10001;

   // Declare a smart pointer to and GandolfMultigroupOpacity.  We use the
   // interface defined by rtt_cdi:MultigroupOpacity.
   rtt_dsxx::SP< rtt_cdi::MultigroupOpacity > spOpacity_mgRosTot;

   // Use a try block when instantiating the MultigroupOpacity object so that
   // we can catch any errors.
   try
      {
         spOpacity_mgRosTot = new
            rtt_cdi_gandolf::GandolfMultigroupOpacity(
               spGF,                 // All data is loaded from this IPCRESS
                                     // file.
               matid,                // This object can only return
                                     // information about one material.
               rtt_cdi::Rosseland,   // All opacity values use the Rosseland
                                     // model.
               rtt_cdi::Total );     // All opacity values are total cross
                                     // sections.
      }
   catch ( rtt_cdi_gandolf::GandolfException gerr )
      {
         // Instantiation failed so print a report and exit.
         cout << gerr.errorSummary();
         return –1;
      }

   double temp = 5.0; // keV
   double dens = 27.0; // g/cm^3

   // Interpolate and return an opacity value for this material at
   // (temp,dens).  There will be ngroup entries.
   vector<double> mgOpacity = spOpacity_mgRosTot->getOpacity( temp, dens );

   // Interpolate and return an a vector of multigroup opacity values for
   // this material at each of the temperature points and the single
   // density provided.  There will be ngroup*ntemp entries.
   const int ngroups = spOpacity_mgRosTot->getNumGroups();
   const int ntemps = 3;
   vector<double> vtemp( ntemps );
   for ( int it=0; it<ntemps; ++it)
      vtemp[it] = 5.0*(it+1);

   vector< vector< double > > vmgOpacity
      = spOpacity_mgRosTot->getOpacity( vtemp, dens );

   return 0;
}
```

Figure 3. Example of using a GandolfMultigroupOpacity object.

GandolfOpacity Member Functions	Role
`OpacityIterator getOpacity( TempIterator tempFirst, TempIterator tempLast, DensIterator densFirst, DensIterator densLast, OpacityIterator opFirst )`	The temperature and density containers are considered a tuple of values. An opacity value is returned for each pair of temperature and density values given. *All containers must be 1D.*
`OpacityIterator getOpacity( TempIterator tempFirst, TempIterator tempLast, double targetDensity, OpacityIterator opFirst )`	The 1D opacity STL container will be filled with values corresponding to each of the given temperatures and the single density value.
`OpacityIterator getOpacity( double targetTemperature, DensIterator densFirst, DensIterator densLast, OpacityIterator opFirst )`	The 1D opacity STL container will be filled with values corresponding to each of the given densities and the single temperature value.

An example using GandolfMultigroupOpacity to obtain opacity values is shown above in Figure 3. For more examples please examine the `tGandolfOpacity.cc` unit test.

5 Gandolf Libraries

Information about the Gandolf libraries can be found online at http://laurel.lanl.gov/XCI/PROJECTS/DATA/atomic/gandolf.html. This web site contains a .tar file with the most current libraries and documentation (LA-UR-00-3207) in a variety of formats.

6 TOPS

TOPS is a code that allows users to generate tables of multigroup opacities using the cross section files generated by Group T-4 (http://www.t4.lanl.gov/). Detailed information about TOPS can be found in the report LA-10454 (download: http://laurel.lanl.gov/XCI/PROJECTS/DATA/atomic/pdf/tops.pdf). The manual includes details about locating binaries and running the code to create IPCRESS files. A list of materials available for mixing by TOPS is located at http://www.t4.lanl.gov/opacity/avmat.html.

The TOPS manual was last printed in 1985 so there are few points in the text that are out of date. The binaries are located on Nirvana at `/usr/projects/data`. The subdirectory `bin` contains the binary and the data tables are located in the `atomic` subdirectory.

Here is a synopsis of one of my TOPS sessions:

1. Log on to Nirvana.

2. Connect to a compute note via `llogin`.

3. Added `/usr/projects/data/bin` to my path.

4. Loaded the MIPS libraries.

5. Run `topsn`.

6. TOPS Session:

```
a. f 0.91 Be 0.09 Cu ! Create a material (91% Be 9% Cu).
b. tby 5             ! Take only every 5th point in tgrid.
c. mgid 10002        ! Set material ID to 10002.
d. rlog 5 0.01 100.0 ! Set the density grid to cover the range 0.01…100 with
                     ! 5 data points per decade.
e. dl f              ! Use multigroup opacities if off density grid.
f. go                ! Compute opacities and write output files.
g. end               ! exit TOPS
```

7. Note:

    a. From within the TOPS program you can use “help” to obtain information about all commands.
    b. t, r, g show current temperature, density and energy group grids.

8. TOPS creates several files in the current working directory (you should probably be working in a scratch directory on Nirvana).

    a. bliss – IPCRESS formatted opacity data (approx 1.5 Mbytes).
    b. tape 6 – list of data exceptions.
    c. topslog – log of TOPS session.

TOPS can also be used to create a table of analytic opacity values specified by a formula from within the TOPS program. You may want to examine the TOPS help for the following commands: source, t, r, g, stem, sig0, topow, ropow, hnupow, and nbsubg.

KT:kt

Distribution:

CCS-4 MS D409:
D.G. Archuleta, CCS-4 MS D409
T.M. Evans, CCS-4 MS D409
M.G. Gray, CCS-4 MS D409
G.H. Hughes, CCS-4 MS D409
J.M. McGhee, CCS-4 MS D409
J.E. Morel, CCS-4 MS D409
G.L. Olson, CCS-4 MS D409
S.D. Pautz, CCS-4 MS D409
R.M. Roberts, CCS-4 MS D409
K.G. Thompson, CCS-4 MS D409
T.J. Urbatsch, CCS-4 MS D409
J.S. Warsa, CCS-4 MS D409