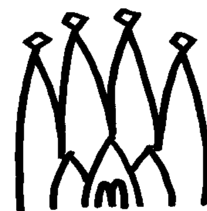

Optical Properties & Surfaces

Version 6.0 (March'**2k+1**)

Ivan Belyaev[1]
(*ITEP, Moscow*)

[1]E-mail: Ivan.Belyaev@itep.ru

Abstract

A way to handle the optical properties of materials and surfaces is described

Table of Contents

1 Optical Properties of Materials **1**

1.1 Transient representation 1

1.2 Filling of `TabulatedProperty` 2

1.3 XML description 3

2 Surfaces and their optical properties **6**

2.1 Transient representation 6

2.2 XML description 7

Chapter 1: Optical Properties of Materials

For **GEANT4** simulation of Cherenkov radiation one definitely needs to define the optical properties of material, like photon absorption length and refractive index, and some others. Since version of **v6** of DETDESC package there is way to specify such properties.

Usually optical properties of materials are given in the form of tables (or 1D-histograms) as a function of wavelength or energy of the photon.

1.1 Transient representation

The class **TabulatedProperty** is designed to describe each optical property of material. The relations between **Material** base class for description of general material properties and the specific tabulated properties of material are shown in figure 1.2.

Material class has two accessors to get an access to the it's own properties:

```
class Material : public DataObject
{
public:
  /// useful typedef
  typedef SmartRefVector<TabulatedProperty>   Tables;
  /// some tabulated properties
  inline       Tables&  tabulatedProperties()       ;
  inline const Tables&  tabulatedProperties() const ;
  ///
  ...
};
```

Class diagram for class **TabulatedProperty** is represented on figure 1.2. The essential public interface is:

```
class TabulatedProperty: public DataObject
{
public:
  /// useful typedefs
  typedef std::pair<double,double>   Entry;
  typedef std::vector<Entry>         Table;
  /// acessors to property type
  inline const std::string&     type () const ;
  inline TabulatedProperty&  setType ( const std::string& T );
  /// acessors to  name of x-axis
  inline const std::string&     xAxis() const ;
  inline TabulatedProperty&  setXAxis( const std::string& T );
  /// acessors to  name of y-axis
```

```
  inline const std::string&     yAxis() const ;
  inline TabulatedProperty&  setYAxis( const std::string& T );
  /// acessors to the table itself
  inline       Table&        table   ()       ;
  inline const Table&        table   () const ;
  /// another form of accessors to table
  inline operator       Table&       ()       ;
  inline operator const Table&       () const ;
  ///
  /** Fill a table from the function
      @param func  The function. It could of type
                   of function, pointer to function,
                   STL adaptor for member function,
                   STL functional or any type of functor
                   or function object
      @param first "Iterator"(in STL sence) to a  first
                   element in a sequence
      @param last  "Iterator"(in STL sence) to a (last+1)
                   element in a sequence
  */
  template< class Func , class Iter>
  inline const StatusCode
              fill( Func func , Iter first , Iter last );
  ...
};
```

In principle, since `TabulatedProperty` class is quite general (since it is simple), it could be reused for other purposes.

1.2 Filling of `TabulatedProperty`

The straightforward way to fill **`TabulatedProperty`** is:

```
  ///
  TabulatedProperty* tabProp = ...
  double x = 0 ;
  double y = 1 ;
  tabProp->table().push_back( Entry( x , y ) );
  ///
```

In addition one could fill **`TabulatedProperty`** using its templated functor method. In following example we have filled **`TabulatedProperty`** with 5 entries using array **`x`** and standard function **`sqrt`**:

```
///
TabulatedProperty* tabProp = ... ;
const double x[] = { 1, 10 , 30 , 3 , 40 };
tabProp->fill(sqrt,x,x+5);
///
```

The way of filling using pointer to standard container **`std::vector<double>* x`** and user function **`func`** is also illustrated:

```
///
double func(double);
///
const std::vector<double>*  x = ... ;
TabulatedProperty*   tabProp  = ... ;
///
tabProp->fill( func, x->begin(), x->end() );
///
```

Any other standard or user defined functions or function objects(functors) are acceptable also.

Since class `TabulatedProperty` inherits from `DataObject` and therefore it could be read/write from/to any type of data base supported within **GAUDI** framework.

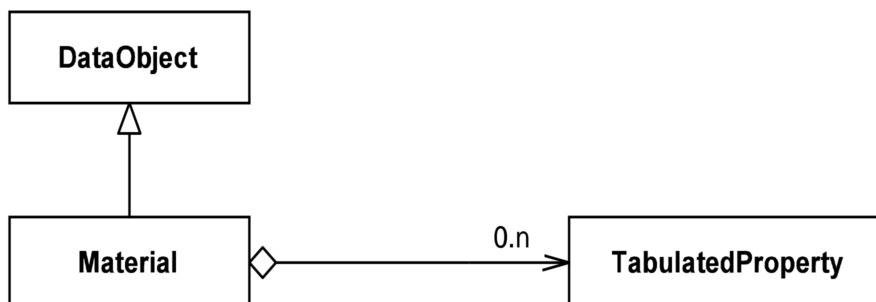

Figure 1.1: Relation between classes `Material` and `TabulatedProperty`. References are of type `SmartRefVector`.

1.3 XML description

XML description of **`TabulatedProperty`** looks like:

```
<tabproperty name  = 'Property1'
             type  = 'ABSLEN'
```

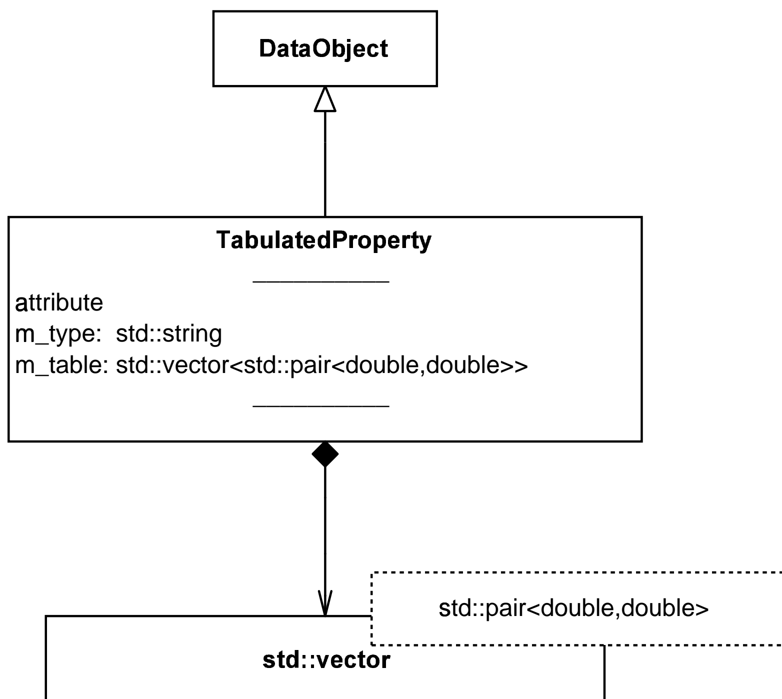

Figure 1.2: Class diagram for class `TabulatedProperty`

```
          xaxis = 'energy'
          yaxis = 'AbsLen'
          xunit = '1*eV'
          yunit = '1*mm' >
      <entry x = '0' y= '100'  />
      <entry x = '1' y= '200' text = 'special!' />
      <entry x = '2' y= '300' />
      4  400
        5  500
     6   600
      10 800
</tabproperty>,
```

Attributes **`xaxis`**,**`yaxis`**, **`xunit`**, **`yunit`** and **`text`** could be omitted, their default values are **`'xAxis'`**, **`'yAxis'`**, **`'1.0'`**,**`'1.0'`** and **`''`** respectively. The above listed XML description define table with 7 entries. First half of entries are given explicitly using **`<entry>`** tag and the remaining three entries are given in a form of free format table.

It is worth to mention that XML converters do not require that values for **`TabulatedProperty`** should be provided with units.

XML description of **`Material`**, which has certain optical properties looks like:

```
<material  name    = 'Water'
           density = '1*g/cm3' >
   <!-- elements -->
      <elementref href='#Hydrogen' natoms='2' />
      <elementref href='#Oxygen'   natoms='1' />
  <!-- optical properties  -->
      <tabprops address='/dd/Materials/Property1' />
      <tabprops address='/dd/Materials/Property2' />
      <tabprops address='/dd/Materials/Property3' />
</material>
```

Chapter 2: Surfaces and their optical properties

2.1 Transient representation

Description of optical properties of surfaces is very similar to optical properties of material. The proposed schema is shown in figure 2.1.
The essential public interface of **Surface** class is:

```
class Surface: public DataObject
{
public:
  /// useful typedef:
  typedef SmartRefVector<TabulatedProperty> Tables;
  /// accessors: (naming from Geant4, except for last)
  /// "model"
  inline const unsigned int model      () const ;
  inline Surface&         setModel     ( const unsigned int );
  /// "finish"
  inline const unsigned int finish     () const ;
  inline Surface&         setFinish    ( const unsigned int );
  /// "type"
  inline const unsigned int type       () const ;
  inline Surface&         setType      ( const unsigned int );
  /// "value" - (NB: quite ugly name!)
  inline const double        value     () const ;
  inline Surface&         setValue     ( const double       );
  /// name of first  volume
  inline const std::string& firstVol  () const ;
  inline Surface&         setFirstVol  ( const std::string& );
  /// name of second volume
  inline const std::string& secondVol () const ;
  inline Surface&         setSecondVol ( const std::string& );
  /// tables of optical prorties
  inline const Tables& tabulatedProperties () const ;
  inline       Tables& tabulatedProperties ()       ;
  ///
...
};
```

This proposal of class **Surface** allows to get a full set of information about the surface and its optical properties. Later this information is used for conversion of **Surface** into **GEANT4** classes **G4LogicalBorderSurface** (both physical names are given) or `G4-LogicalSkinSurface` (only one logical volume name is given) and to their corresponding `G4OpticalSurface` with their own properties.

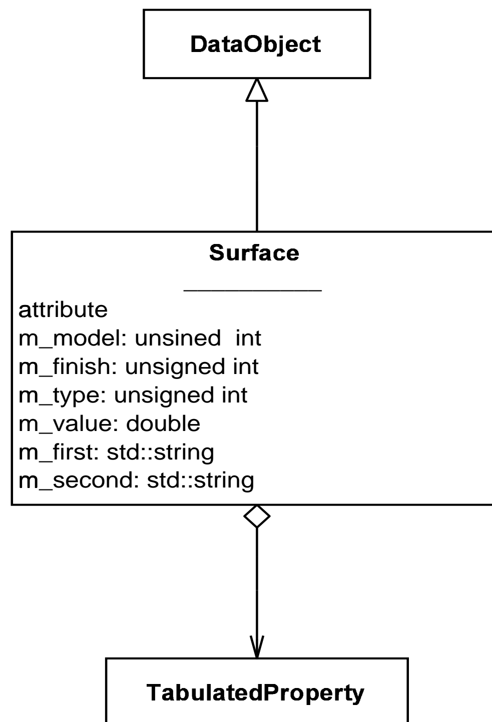

Figure 2.1: Class diagram for class **`Surface`**. References of type **`SmartRefVector`**

2.2 XML description

The simplest XML description of border surface could look like:

```
<surface name   = 'MirrorSurface1'
         model  = '0'
         finish = '0'
         type   = '0'
         value  = '0'
  <!-- it is border surface!
       both names are names
       of physical volumes  -->
  volfirst  = '/dd/Geometry/Rich1/lvRich1#part1'
  volsecond = '/dd/Geometry/Rich1/lvRich1#part2' >

  <!-- properties -->
  <tabprops address='/dd/Geometry/Rich1/Property1' />
  <tabprops address='/dd/Geometry/Rich1/Property2' />
  <tabprops address='/dd/Geometry/Rich1/Property3' />
```

```
</surface>
```

The simplest XML description of skin surface could look like:

```
<surface name   = 'MirrorSurface2'
         model  = '0'
         finish = '0'
         type   = '0'
         value  = '0'
  <!-- it is skin surface!
       the only name is the
       name of logical volume -->
  volfirst  = '/dd/Geometry/Rich1/lvRich1' >

  <!-- properties -->
  <tabprops address='/dd/Geometry/Rich1/Property1' />
  <tabprops address='/dd/Geometry/Rich1/Property2' />
  <tabprops address='/dd/Geometry/Rich1/Property3' />

</surface>
```

To be converted into **GEANT4** **`Surface`** should be attached to some logical volume:

```
<logvol name   = 'lvRich1'  .... >

  <surf address='/dd/Geometry/Rich1/Surface1' />
  <surf address='/dd/Geometry/Rich1/Surface2' />

</surface>
```