CGM Library Version 0.9.5
Beta
## Contents

1 Introduction ................................................................. 7
2 Tutorial ............................................................................. 8
3 API ................................................................................... 9
   3.1 Comm ................................................................. 9
      3.1.1 BasicCommObject — Implementation of CommOb-  
       ject for basic C++ data types .......................... 10
      3.1.1.1 constructors ............................................. 10
      3.1.1.2 methods .................................................. 11
      3.1.1.3 destructor ............................................... 15
      3.1.2 Comm — Interface for inter-process communica-  
       tion. .................................................................. 16
      3.1.2.1 constants .................................................. 16
      3.1.2.2 methods ................................................... 18
      3.1.2.3 destructor ............................................... 27
      3.1.3 CommObject — Interface for CGM algorithm  
       data. .................................................................. 28
      3.1.3.1 methods ................................................... 28
      3.1.3.2 destructor ............................................... 31
      3.1.4 CommObjectList — A container to encapsulate a  
       list of CommObject’s. ............................................. 32
      3.1.4.1 constructors ............................................. 33
      3.1.4.2 methods ................................................... 34
      3.1.4.3 destructor ............................................... 34
      3.1.5 MPIComm — Implementation of Comm using  
       MPI .................................................................... 35
      3.1.5.1 methods ................................................... 35
      3.1.6 SimpleCommObject — Implementation of Com-  
       mObject for Simple C++ data types ..................... 45
      3.1.6.1 constructors ............................................. 46
      3.1.6.2 methods ................................................... 47
      3.1.6.3 destructor ............................................... 50
   3.2 Parallel Prefix Sum .................................................... 51
      3.2.1 ParallelPrefixSummer — Calculate prefix sum ... 51
      3.2.1.1 constructor ............................................... 52
      3.2.1.2 method .................................................... 52
      3.2.1.3 destructor ............................................... 53
   3.3 Parallel Sorting .......................................................... 54
      3.3.1 ParallelSorter — Sort data ................................ 54
      3.3.1.1 constructors ............................................. 54

This page has been automatically generated with DOC++

DOC++ is ©1995 by Roland Wunderling
Malte Zöckler
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.4.1</td>
<td>BasicRequest — <em>Implement a basic request</em></td>
<td>60</td>
</tr>
<tr>
<td>3.4.1.1</td>
<td>constructors</td>
<td>60</td>
</tr>
<tr>
<td>3.4.1.2</td>
<td>methods</td>
<td>62</td>
</tr>
<tr>
<td>3.4.1.3</td>
<td>destructor</td>
<td>64</td>
</tr>
<tr>
<td>3.4.2</td>
<td>Request — <em>Interface for an enquiry request</em></td>
<td>65</td>
</tr>
<tr>
<td>3.4.2.1</td>
<td>methods</td>
<td>66</td>
</tr>
<tr>
<td>3.4.2.2</td>
<td>destructor</td>
<td>68</td>
</tr>
<tr>
<td>3.4.3</td>
<td>RequestSystem — <em>A class to send requests and responses</em></td>
<td>69</td>
</tr>
<tr>
<td>3.4.3.1</td>
<td>methods</td>
<td>69</td>
</tr>
<tr>
<td>3.4.4</td>
<td>Response — <em>Interface for an response</em></td>
<td>71</td>
</tr>
<tr>
<td>3.4.4.1</td>
<td>methods</td>
<td>72</td>
</tr>
<tr>
<td>3.4.4.2</td>
<td>destructor</td>
<td>75</td>
</tr>
<tr>
<td>3.4.5</td>
<td>SimpleRequest — <em>A templated class to implement a data carried request</em></td>
<td>75</td>
</tr>
<tr>
<td>3.4.5.1</td>
<td>constructors</td>
<td>76</td>
</tr>
<tr>
<td>3.4.5.2</td>
<td>methods</td>
<td>77</td>
</tr>
<tr>
<td>3.4.5.3</td>
<td>destructor</td>
<td>81</td>
</tr>
<tr>
<td>3.4.6</td>
<td>SimpleResponse — <em>A templated class to implement a data carried response</em></td>
<td>81</td>
</tr>
<tr>
<td>3.4.6.1</td>
<td>constructors</td>
<td>82</td>
</tr>
<tr>
<td>3.4.6.3</td>
<td>destructor</td>
<td>87</td>
</tr>
<tr>
<td>3.4.1.2</td>
<td>methods</td>
<td>55</td>
</tr>
<tr>
<td>3.4.1.3</td>
<td>destructor</td>
<td>59</td>
</tr>
<tr>
<td>3.4.2</td>
<td>Request — <em>Interface for an enquiry request</em></td>
<td>65</td>
</tr>
<tr>
<td>3.4.2.1</td>
<td>methods</td>
<td>66</td>
</tr>
<tr>
<td>3.4.2.2</td>
<td>destructor</td>
<td>68</td>
</tr>
<tr>
<td>3.4.3</td>
<td>RequestSystem — <em>A class to send requests and responses</em></td>
<td>69</td>
</tr>
<tr>
<td>3.4.3.1</td>
<td>methods</td>
<td>69</td>
</tr>
<tr>
<td>3.4.4</td>
<td>Response — <em>Interface for an response</em></td>
<td>71</td>
</tr>
<tr>
<td>3.4.4.1</td>
<td>methods</td>
<td>72</td>
</tr>
<tr>
<td>3.4.4.2</td>
<td>destructor</td>
<td>75</td>
</tr>
<tr>
<td>3.4.5</td>
<td>SimpleRequest — <em>A templated class to implement a data carried request</em></td>
<td>75</td>
</tr>
<tr>
<td>3.4.5.1</td>
<td>constructors</td>
<td>76</td>
</tr>
<tr>
<td>3.4.5.2</td>
<td>methods</td>
<td>77</td>
</tr>
<tr>
<td>3.4.5.3</td>
<td>destructor</td>
<td>81</td>
</tr>
<tr>
<td>3.4.6</td>
<td>SimpleResponse — <em>A templated class to implement a data carried response</em></td>
<td>81</td>
</tr>
<tr>
<td>3.4.6.1</td>
<td>constructors</td>
<td>82</td>
</tr>
<tr>
<td>3.4.6.3</td>
<td>destructor</td>
<td>87</td>
</tr>
<tr>
<td>3.4.1.2</td>
<td>methods</td>
<td>55</td>
</tr>
<tr>
<td>3.4.1.3</td>
<td>destructor</td>
<td>59</td>
</tr>
<tr>
<td>3.4.2</td>
<td>Request — <em>Interface for an enquiry request</em></td>
<td>65</td>
</tr>
<tr>
<td>3.4.2.1</td>
<td>methods</td>
<td>66</td>
</tr>
<tr>
<td>3.4.2.2</td>
<td>destructor</td>
<td>68</td>
</tr>
<tr>
<td>3.4.3</td>
<td>RequestSystem — <em>A class to send requests and responses</em></td>
<td>69</td>
</tr>
<tr>
<td>3.4.3.1</td>
<td>methods</td>
<td>69</td>
</tr>
<tr>
<td>3.4.4</td>
<td>Response — <em>Interface for an response</em></td>
<td>71</td>
</tr>
<tr>
<td>3.4.4.1</td>
<td>methods</td>
<td>72</td>
</tr>
<tr>
<td>3.4.4.2</td>
<td>destructor</td>
<td>75</td>
</tr>
<tr>
<td>3.4.5</td>
<td>SimpleRequest — <em>A templated class to implement a data carried request</em></td>
<td>75</td>
</tr>
<tr>
<td>3.4.5.1</td>
<td>constructors</td>
<td>76</td>
</tr>
<tr>
<td>3.4.5.2</td>
<td>methods</td>
<td>77</td>
</tr>
<tr>
<td>3.4.5.3</td>
<td>destructor</td>
<td>81</td>
</tr>
<tr>
<td>3.4.6</td>
<td>SimpleResponse — <em>A templated class to implement a data carried response</em></td>
<td>81</td>
</tr>
<tr>
<td>3.4.6.1</td>
<td>constructors</td>
<td>82</td>
</tr>
<tr>
<td>3.4.6.3</td>
<td>destructor</td>
<td>87</td>
</tr>
<tr>
<td>3.4.1.2</td>
<td>methods</td>
<td>55</td>
</tr>
<tr>
<td>3.4.1.3</td>
<td>destructor</td>
<td>59</td>
</tr>
<tr>
<td>3.4.2</td>
<td>Request — <em>Interface for an enquiry request</em></td>
<td>65</td>
</tr>
<tr>
<td>3.4.2.1</td>
<td>methods</td>
<td>66</td>
</tr>
<tr>
<td>3.4.2.2</td>
<td>destructor</td>
<td>68</td>
</tr>
<tr>
<td>3.4.3</td>
<td>RequestSystem — <em>A class to send requests and responses</em></td>
<td>69</td>
</tr>
<tr>
<td>3.4.3.1</td>
<td>methods</td>
<td>69</td>
</tr>
<tr>
<td>3.4.4</td>
<td>Response — <em>Interface for an response</em></td>
<td>71</td>
</tr>
<tr>
<td>3.4.4.1</td>
<td>methods</td>
<td>72</td>
</tr>
<tr>
<td>3.4.4.2</td>
<td>destructor</td>
<td>75</td>
</tr>
</tbody>
</table>
### 3.5.2.3 IntegerComparator — *Interface for the integer comparator.*
- 3.5.2.3.1 method ........................................ 99

### 3.5.2.4 IntegerSorter — *Implementation of the sorter using integer sort.*
- 3.5.2.4.1 constructors ...................... 100
- 3.5.2.4.2 methods ....................... 101
- 3.5.2.4.3 destructor ....................... 102

### 3.5.2.5 ReverseComparator — *A "reverse" comparator.*
- 3.5.2.5.1 constructors ...................... 104
- 3.5.2.5.2 methods ....................... 104
- 3.5.2.5.3 destructor ....................... 106

### 3.5.2.6 Sorter — *Interface for the sorter.*
- 3.5.2.6.1 methods ....................... 107
- 3.5.2.6.2 destructor ....................... 108

### 3.5.3 Timers
- 3.5.3.1 CGMTimers — *Combines a set of timers.*
  - 3.5.3.1.1 constructors ...................... 110
  - 3.5.3.1.2 methods ....................... 111
  - 3.5.3.1.3 destructor ....................... 116
- 3.5.3.2 CPUTimer — *Implementation of the timer using the UNIX CPU ticks.*
  - 3.5.3.2.1 constructors ...................... 117
  - 3.5.3.2.2 methods ....................... 118
  - 3.5.3.2.3 destructor ....................... 120
- 3.5.3.3 Timer — *Interface for the timer.*
  - 3.5.3.3.1 methods ....................... 121
  - 3.5.3.3.2 destructor ....................... 123
- 3.5.3.4 UnixTimer — *Implementation of the timer using the UNIX built-in timing facility.*
  - 3.5.3.4.1 constructors ...................... 124
  - 3.5.3.4.2 methods ....................... 125
  - 3.5.3.4.3 destructor ....................... 127

### 3.5.4 Miscellaneous Utilities
- 3.5.4.1 Debug — *Debug utilities.*
  - 3.5.4.1.1 methods ....................... 128
- 3.5.4.2 GeneralUtilities — *General utilities for use by the users of this library.*
  - 3.5.4.2.1 methods ....................... 131
3.5.4.3 ObjList — A container to encapsulate a Objlist of objects. 136
3.5.4.3.1 constructors 137
3.5.4.3.2 methods 138
3.5.4.3.3 destructor 143
3.5.4.4 Operator — Interface for the operator to perform prefix sum. 144
3.5.4.4.1 method 144
3.5.4.4.2 destructor 145

4 graphlib API 146
4.1 List Ranking 146
4.1.1 ListRanker — An implementation of the list ranking algorithm 146
4.1.1.1 constants 147
4.1.1.2 method 148
4.1.2 Node — A class to implement a node 148
4.1.2.1 constructors 149
4.1.2.2 methods 150

4.2 Graphs 155
4.2.1 General 155
4.2.1.1 Edge — An implementation of an edge 155
4.2.1.1.1 constructors 156
4.2.1.1.2 methods 158
4.2.1.1.3 destructor 162
4.2.1.2 Graph — An implementation of a graph 163
4.2.1.2.1 constructors 163
4.2.1.2.2 methods 164
4.2.1.3 Vertex — Implementation of a vertex 169
4.2.1.3.1 constants 169
4.2.1.3.2 constructors 171
4.2.1.3.3 methods 172
4.2.1.3.4 destructor 178

4.2.2 Euler Tour 178
4.2.2.1 EulerNode — A node that is used in the Euler Tour algorithm 178
4.2.2.1.1 constructors 179
4.2.2.1.2 methods 180
4.2.2.2 EulerTourer — An implementation of the Euler Tour algorithm 182
4.2.2.2.1 method 183

4.2.3 Connected Components and Spanning Forest 184
4.2.3.1 ConnectedComponents — *An implementation of the connected components and spanning forest algorithms* ........ 184
4.2.3.1.1 methods ...................... 184

4.2.4 Bipartite Graph Detection ......................... 186
4.2.4.1 BipartiteDetector — *An implementation of the bipartite graph detection algorithm* .......................... 186
4.2.4.1.1 method ...................... 186

5 Distribution and Installation ............................... 188
6 Licensing .............................................. 190
7 Bug Report ........................................... 202
8 Acknowledgement ........................................ 203
Class Graph ................................................ 204
A CGM \((s,p)\) consists of \(p\) processors, \(P_1 \ldots P_p\), each with local memory of size \(O(s)\). The processors are connected via an arbitrary communication network or shared memory.

The fundamental inter-processor communication operation supported by the CGM \((s,p)\) is the \(h\)-Relation, \(h \leq s\), defined in the following. Each processor \(P_i\) has two arrays \(OUT_i\) and \(IN_i\), referred to as output buffer and input buffer, respectively, each containing at most \(O(h)\) data items. Every processor \(P_i\) partitions its output buffer, \(OUT_i\), into \(p\) subarrays \(OUT_{i,1} \ldots OUT_{i,p}\). The \(h\)-Relation operation consists of routing every \(OUT_{i,j}, 1 \leq i \leq j \leq p\), from processor \(P_i\) to processor \(P_j\). Each processor \(P_j\) receives buffer pieces \(OUT_{i,j}, 1 \leq i \leq p\), and stores them in its input buffer, \(IN_i\), in consecutive order. An additional important condition is that no processor receives more than \(O(h)\) data items, i.e. the output buffers are partitioned in such a way that \(\sum_{j=1}^{p} (size(OUT_{i,j})) \leq O(h)\) for all \(1 \leq i \leq p\). We refer to an \(h\)-Relation operation with arrays \(OUT_i\) and \(IN_i\) as described above as \(h\)-Relation \(OUT_i, IN_i\).

In addition to its fundamental communication operation, the \(h\)-Relation, a CGM has other basic communication operations which have the property that they can be implemented as a constant number of \(h\)-Relation operations. Such operations include broadcast (in which one processor sends a data set to all other processors) and several others.

A CGM algorithm, solving a problem with input size \(n\), will start with \(n/p\) of those input data items stored in the local memory of each processor. Unless stated otherwise, there is no assumption made about the initial distribution of those data items over the processors. A CGM algorithm consists of a sequence of rounds where each round is composed of a local computation rounds followed by a global communication rounds. In a local computation round, all processors compute independently, and in parallel, on data stored in their local memories, and without any communication between the processors. When all processors have completed this local computation, they collectively start a global communication round, which consists of a constant number of \(h\)-Relation operations.

Note that, the above definition of CGM rounds also permits the overlapping of computation and communication. An \(h\)-relation implementation for a particular machine is free to start communicating data as soon as it is placed in the \(OUT\) buffers. If the network at hand is based on packet switching, then it might be beneficial to immediately start sending packets during the communication round. If the network is based on circuit switching, then the network could start setting up communication channels immediately during the communication round. However, the cgm algorithm can only assume that the \(IN\) buffers are fully received at the beginning of the next round.
• A tutorial presented on January 31, 2003 at Carleton University
3.1 Comm

Names

3.1.1 template<class T> class

BasicCommObject : public CommObject

Implementation of CommObject for basic C++ data types 

3.1.2 class

Comm

Interface for inter-process communication. 

3.1.3 class

CommObject

Interface for CGM algorithm data. 

3.1.4 class

CommObjectList : public ObjList<CommObject>

A container to encapsulate a list of CommObject’s. 

3.1.5 class

MPIComm : public Comm

Implementation of Comm using MPI 

3.1.6 template<class T> class

SimpleCommObject : public CommObject

Implementation of CommObject for Simple C++ data types 

This page has been automatically generated with DOC++

DOC++ is ©1995 by Roland Wunderling
Malte Zickler
3.1.1  

\texttt{template<class T> class BasicCommObject: public CommObject}

\textit{Implementation of CommObject for basic C++ data types}

\section*{Inheritance}

\begin{center}
\begin{tikzpicture}[level 1/.style={sibling distance=20mm},level 2/.style={sibling distance=10mm}]
  \node (commobject) {CommObject}
    child {node (basiccommobject) {BasicCommObject}};
\end{tikzpicture}
\end{center}

\section*{Public Members}

\begin{itemize}
  \item \textbf{constructors} \hfill \begin{center} 10 \end{center}
  \item \textbf{methods} \hfill \begin{center} 11 \end{center}
  \item \textbf{destructor} \hfill \begin{center} 15 \end{center}
\end{itemize}

\textit{Implementation of CommObject for basic C++ data types}

\subsection*{3.1.1.1 \textbf{constructors}}

\textbf{Names}

\begin{itemize}
  \item \textbf{BasicCommObject} (T t)  
    \textit{Conversion constructor}. \quad \begin{center} 11 \end{center}
  \item \textbf{BasicCommObject} (BasicCommObject &another)  
    \textit{Copy constructor}. \quad \begin{center} 11 \end{center}
\end{itemize}
3.1.1.1

BasicCommObject (T t)

Conversion constructor.

Parameters:  

  t  the data to be pre-populated into the object.

3.1.1.2

BasicCommObject (BasicCommObject &another)

Copy constructor.

Parameters:  

  another  a reference to another BasicCommObject of the same type.

3.1.1.2

methods

Names

3.1.1.2.1 virtual void copyToArray (char* array, int offset)

Copy the content of the object to an array.

..................  12

3.1.1.2.2 virtual void
copyFromArray (char* array, int offset)

Copy from an array the content of this object.

3.1.1.2.3 virtual CommObject*
    clone (void* p = NULL)

Clone the object.

3.1.1.2.4 virtual CommObject&
    operator= (CommObject &another)

Assignment operator.

3.1.1.2.5 virtual void
    sendToOstream (ostream &outstream)

Send the content of this object to the ostream.

3.1.1.2.6 virtual int
    getSize ()

get the size of the object when "serialized".

3.1.1.2.7 virtual T&
    getData (void)

Retrieve the data.

3.1.1.2.8 virtual void
    setData (T t)

Set the data.

---

3.1.1.2.1

virtual void copyToArray (char* array, int offset)

Copy the content of the object to an array.

Parameters:

- `target`: a character array.
- `offset`: the offset of the array to start copy to.

---

3.1.1.2.2

virtual void copyFromArray (char* array, int offset)
Copy from an array the content of this object.

Parameters:  

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>target</td>
<td>a character array</td>
</tr>
<tr>
<td>offset</td>
<td>the offset of the array to start copy from</td>
</tr>
</tbody>
</table>

3.1.1.2.3

virtual CommObject* clone (void* p = NULL)

Clone the object.

Return Value:  

- the address of clone.

Parameters:  

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
<td>the memory that the object is suppose to clone into</td>
</tr>
</tbody>
</table>

3.1.1.2.4

virtual CommObject& operator= (CommObject &another)

Assignment operator.

Return Value:  

- a reference to the object itself.

Parameters:  

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>another</td>
<td>a reference to another CommObject</td>
</tr>
</tbody>
</table>
### 3.1.1.2.5

**virtual void sendToOstream (ostream &outstream)**

Send the content of this object to the ostream.

Send the content of this object to the ostream.

**Parameters:**

outstream  a reference to the ostream.

---

### 3.1.1.2.6

**virtual int getSize ()**

get the size of the object when "serialized".

get the size of the object when "serialized".

**Return Value:**

the  size in byte.

---

### 3.1.1.2.7

**virtual T& getData (void)**

Retrive the data.

Retrive the data. Note this is a non-interface method.

**Return Value:**

a  reference to the data.
3.1.1.2.8

virtual void 

Set the data.

Parameters:

\text{t} \quad \text{the data to be set.}

3.1.1.3

destructor

Names

3.1.1.3.1 virtual ~\text{BasicCommObject} (void)

Destructor of the class

3.1.1.3.1

virtual ~\text{BasicCommObject} (void)

Destructor of the class
3.1.2 class Comm

Interface for inter-process communication.

Inheritance

3.1.2 Comm

3.1.5 MPIComm

Public Members

3.1.2.1 constants .............................. 16
3.1.2.2 methods ................................. 18
3.1.2.3 destructor .............................. 27

Interface for inter-process communication. This interface defines a set of common methods for inter-process communication.

3.1.2.1 constants

Names

3.1.2.1.1 static char*

MPICOMM A character string to identify MPI, this will be initialized to "MPICOMM" ........................ 17

3.1.2.1.2 static const int

ANY_NODE An integer to represent that we don’t care which processor is involved .............................. 17

3.1.2.1.3 static const int
3.1.2.1.1

static char* MPICOMM

A character string to identify MPI, this will be initialized to "MPICOMM"

3.1.2.1.2

static const int ANY_NODE

An integer to represent that we don’t care with processor is involved

3.1.2.1.3

static const int PROC_ZERO

An integer to represent the super hero processor - the leading processor
3.1.2.2 methods

Names
3.1.2.2.1 static Comm*

    getComm (int* argc, char*** argv,
              CGMTimers* timers=NULL,
              char* selector=NULL)

    To retrieve a specific comm. .... 19

3.1.2.2.2 virtual int

    getNumberOfProcessors (void)

    Get the number of processors in
    the current comm. ............ 20

3.1.2.2.3 virtual int

    getMyId (void)

    Get the id of the current processor.
    ....................... 20

3.1.2.2.4 virtual CGMTimers*

    getTimers (void)

    Get the timers that are associated
    with the comm ............ 21

3.1.2.2.5 virtual void

    synchronize (void)

    Synchronization among all proces-
    sors. ...................... 21

3.1.2.2.6 virtual int

    send (int target, CommObjectList &data, int tag)

    Send data. .................. 22

3.1.2.2.7 virtual int

    receive (int source, CommObjectList &data,
             int tag, int* actualSource)

    Receive data. ................ 22

3.1.2.2.8 virtual int

    oneToAllBCast (int source,
                   CommObjectList &data)

    Broadcast data. ............. 22

3.1.2.2.9 virtual int

    allToOneGather (int target,
                   CommObjectList &data)

    Gather data. ................ 23

3.1.2.2.10 virtual int

    hRelation (CommObjectList &data, int* ns)

    Perform an h-Relation. ........ 23

3.1.2.2.11 virtual int

    allToAllBCast (CommObjectList &data)

    All to all broadcast. ........ 24

3.1.2.2.12 virtual int

    allToAllBCast2 (CommObjectList &data)
A faster version of all to all broadcast. .......................... 24

3.1.2.2.13 virtual int arrayBalancing (CommObjectList &data, int expectedN=-1)
Balancing an array. ............... 25

3.1.2.2.14 virtual int arrayBalancing2 (CommObjectList &data, int expectedN=-1)
Balancing an array. ............... 25

3.1.2.2.15 virtual void partitionCGM (int* sizes)
Partition the CGM environment into groups. ...................... 26

3.1.2.2.16 virtual void partitionCGM (int groupId)
Partition the CGM environment into groups. ...................... 26

3.1.2.2.17 virtual void unPartitionCGM (void)
Undo the previous partition operation. ......................... 26

3.1.2.2.18 virtual void dispose (void) Dispose the comm ............ 27

3.1.2.1

static Comm* getComm (int* argc, char*** argv, CGM-Timers* timers=NULL, char* selector=NULL)

To retrieve a specific comm.

To retrieve a specific comm. This is a static method.

Return Value: a pointer to the selected comm
Parameters:  

- **argc**: a pointer to the command line argument
- **argv**: a pointer to the command line argument
- **timers**: a pointer to a CGMTimer (optional, default to NULL)
- **selector**: a character string to represent the desired comm (optional, the default comm will be selected if missing)

### 3.1.2.2.2

```cpp
virtual int getNumberOfProcessors (void)
```

Get the number of processors in the current comm.

Get the number of processors in the current comm. Note that the number of processors changes if you call partitionCGM and unPartitionCGM. Do not cache this value if you need to use the above mentioned methods.

**Return Value:** the number of processor in the current comm

**See Also:** partitionCGM unPartitionCGM

### 3.1.2.2.3

```cpp
virtual int getId (void)
```

Get the id of the current processor.

Get the id of the current processor. Note that the number of processors changes if you call partitionCGM and unPartitionCGM. Do not cache this value if you need to use the above mentioned methods.
3 API

Return Value: the id.
See Also: partitionCGM unPartitionCGM

### 3.1.2.2.4

virtual CGMTimers* **getTimers** (void)

Get the timers that are associated with the comm

Return Value: the timers.

### 3.1.2.2.5

virtual void **synchronize** (void)

Synchronization among all processors.

Synchronization among all processors. Note that this is a collective method. That means all processors in the current comm must call this method at the same time.

### 3.1.2.2.6

virtual int **send** (int target, CommObjectList &data, int tag)

Send data.
Send data.

Return Value: the number of CommObject’s sent.
Parameters: target the id of the target processor.
            data the data to be sent.
            tag a tag to label the message.
See Also: receive (→3.1.2.2.7, page 22)

### 3.1.2.2.7

```
virtual int receive (int source, CommObjectList &data,
                    int tag, int* actualSource)
```

Receive data.

Receive data.

Return Value: the number of CommObject’s received.
Parameters: source the id of the source processor.
            data the data to be received.
            tag a tag to label the message.
            sctualSource the processor that actually sent the data.
See Also: send (→3.1.2.2.6, page 22)

### 3.1.2.2.8

```
virtual int oneToAllBCast (int source, CommObjectList &data)
```

Broadcast data.
Broadcast data. Note that this is a collective method. That means all processors in the current comm must call this method at the same time.

Return Value: the number of CommObject’s sent/received.
Parameters: source the id of the source processor, data the data to be broadcast.

3.1.2.2.9

virtual int allToOneGather (int target, CommObjectList &data)

Gather data.

Gather data. Note that this is a collective method. That means all processors in the current comm must call this method at the same time.

Return Value: the number of CommObject’s sent/received.
Parameters: target the id of the target processor, data the data to be sent.

3.1.2.2.10

virtual int hRelation (CommObjectList &data, int* ns)

Perform an h-Relation.

Perform an h-Relation. Note that this is a collective method. That means all processors in the current comm must call this method at the same time.
3 API

Return Value: the number of CommObject’s received.
Parameters: data the data to be routed.
            ns the number of CommObjects to be sent to each processor.

3.1.2.2.11
virtual int allToAllBCast (CommObjectList &data)

All to all broadcast. Applicable only in environment with scalability of 1 or less, but will require $O(\log p)$ communication round. Note that this is a collective method. That means all processors in the current comm must call this method at the same time.

Return Value: the total number of CommObject’s received by each processor.
Parameters: data the data to be broadcast.

3.1.2.2.12
virtual int allToAllBCast2 (CommObjectList &data)

A faster version of all to all broadcast. Applicable only in environment with scalability of 0.5 or less. Note that this is a collective method. That means all processors in the current comm must call this method at the same time.

Return Value: the total number of CommObject’s received by each processor.
Parameters: data the data to be broadcast.
virtual int arrayBalancing (CommObjectList &data, int expectedN=-1)

Balancing an array.

Balancing an array. Note that this is a collective method. That means all processors in the current comm must call this method at the same time.

Return Value: the final number of CommObject’s at each processor.

Parameters: data the data to be balanced.
            expectedN the desired number of CommObjects eventually at each processor (optional, default to as even as possible).

virtual int arrayBalancing2 (CommObjectList &data, int expectedN=-1)

Balancing an array.

Balancing an array. Note that this is a collective method. That means all processors in the current comm must call this method at the same time. This is a simpler version of arrayBalancing that is applicable only in environment with scalability of 0.5 or less.

Return Value: the final number of CommObject’s at each processor.

Parameters: data the data to be balanced.
            expectedN the desired number of CommObjects eventually at each processor (optional, default to as even as possible).
3.1.2.2.15

virtual void partitionCGM (int* sizes)

Partition the CGM environment into groups.

Partition the CGM environment into groups. Note that this is a collective method. That means all processors in the current comm must call this method at the same time.

Parameters:

sizes an integer array to represent how many processors each new group should have.

3.1.2.2.16

virtual void partitionCGM (int groupId)

Partition the CGM environment into groups.

Partition the CGM environment into groups. Note that this is a collective method. That means all processors in the current comm must call this method at the same time.

Parameters:

groupId an integer to represent the group that the processor wish to belong to.

3.1.2.2.17

virtual void unPartitionCGM (void)

Undo the previous partition operation.
Undo the previous partition operation. Note that this is a collective method. That means all processors in the current comm must call this method at the same time.

```
3.1.2.2.18

virtual void dispose (void)
```

*Dispose the comm*

Dispose the comm

```
3.1.2.3
destructor
```

*Names*

3.1.2.3.1 virtual ~Comm (void)  Destructor of the class  ............  27

```
3.1.2.3.1

virtual ~Comm (void)
```

*Destructor of the class*

Destructor of the class
3.1.3

class CommObject

Interface for CGM algorithm data.

Inheritance

3.1.3
CommObject

-> 3.4.4
Response

-> 3.4.2
Request

-> 3.1.6
SimpleCommObject

-> 3.1.1
BasicCommObject

Public Members

3.1.3.1  methods ........................................ 28
3.1.3.2  destructor ....................................... 31

Interface for CGM algorithm data. This interface defines a set of methods that are common to all data that have to be sent across the processor boundary. Any data that need to be sent to other processors must be sub-classed from this interface.

3.1.3.1
methods
Names

3.1.3.1.1 virtual void copyToArray (char* target, int offset)
Copy the content of the object to an array. .......................... 29

3.1.3.1.2 virtual void copyFromArray (char* target, int offset)
Copy from an array the content of this object. ..................... 30

3.1.3.1.3 virtual CommObject* clone (void* p = NULL)
Clone the object. ...................... 30

3.1.3.1.4 virtual CommObject& operator= (CommObject &another)
Assignment operator. .......... 30

3.1.3.1.5 virtual void sendToOstream (ostream& outstream)
Send the content of this object to the ostream. ..................... 31

3.1.3.1.6 virtual int getSize ()
get the size of the object when "serialized". ....................... 31

virtual void copyToArray (char* target, int offset)

Copy the content of the object to an array.

Parameters:

- **target**: a character array.
- **offset**: the offset of the array to start copy to.
3.1.3.1.2

virtual void copyFromArray (char* target, int offset)

Copy from an array the content of this object.

Parameters:

- **target**: a character array.
- **offset**: the offset of the array to start copy from.

3.1.3.1.3

virtual CommObject* clone (void* p = NULL)

Clone the object.

Return Value: the address of clone.

Parameters:

- **p**: the memory that the object is suppose to clone into.

3.1.3.1.4

virtual CommObject& operator= (CommObject &another)

Assignment operator.

Assignment operator.
Return Value: a reference to the object itself.
Parameters: another a reference to another CommObject.

3.1.3.1.5

virtual void sendToOstream (ostream& outstream)

Send the content of this object to the ostream.

Parameters: outstream a reference to the ostream.

3.1.3.1.6

virtual int getSize ()

get the size of the object when "serialized”.

Return Value: the size in byte.

3.1.3.2

destructor

Names
3.1.3.2.1 virtual “CommObject (void)

Destructor of the class

........... 32
3.1.3.2.1

virtual ~CommObject (void)

Destructor of the class

Destructor of the class

3.1.4

class CommObjectList : public ObjList<CommObject>

A container to encapsulate a list of CommObject’s.

Inheritance

Public Members

3.1.4.1 constructors ......................... 33
3.1.4.2 methods ......................... 34
3.1.4.3 destructor ......................... 34

A container to encapsulate a list of CommObject’s. This container will manage all the memory location it owns. It will never claim ownership on any pointer passed to it.
3.1.4.1 constructors

Names

3.1.4.1.1 CommObjectList (CommObject* sample, int n=0, CommObject** data=NULL)
Create a CommObjectList and initialize it.

3.1.4.1.2 CommObjectList (CommObjectList &another)
Copy constructor.

3.1.4.1.1 CommObjectList (CommObject* sample, int n=0, CommObject** data=NULL)

Create a CommObjectList and initialize it.

Parameters:

- **sample**: a template to create new object, cannot be NULL.
- **n**: the size of the container, cannot be less than zero (optional, default to zero).
- **data**: the data to initialize the content (optional, default to NULL).

3.1.4.1.2 CommObjectList (CommObjectList &another)

Copy constructor.
Copy constructor.

**Parameters:** another CommObjectList to copy from.

### 3.1.4.2 methods

#### Names

3.1.4.2.1 int **getObjectSize** (void)

*Retrieve the size of one CommObject stored in the container.*

---

3.1.4.2.1 int **getObjectSize** (void)

*Retrieve the size of one CommObject stored in the container.*

**Return Value:** the size of one CommObject.

### 3.1.4.3 destructor

#### Names

3.1.4.3.1 ~**CommObjectList** (void)

*Destructor of the class*
3.1.4.3.1

~CommObjectList (void)

Destructor of the class

3.1.5

class MPIComm : public Comm

Implementation of Comm using MPI

Inheritance

Comm

MPIComm

Public Members

3.1.5.1 methods

Implementation of Comm using MPI
Names

3.1.5.1.1

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>getComm (int* argc, char*** argv, CGMTimers* timers=NULL)</td>
<td>To retrieve the comm.</td>
<td>37</td>
</tr>
<tr>
<td>getNumberOfProcessors (void)</td>
<td>Get the number of processors in the current comm.</td>
<td>38</td>
</tr>
<tr>
<td>getMyId (void)</td>
<td>Get the id of the current processor.</td>
<td>38</td>
</tr>
<tr>
<td>getTimers (void)</td>
<td>Get the timers that are associated with the comm</td>
<td>39</td>
</tr>
<tr>
<td>synchronize (void)</td>
<td>Synchronization among all processors.</td>
<td>39</td>
</tr>
<tr>
<td>send (int target, CommObjectList &amp;data, int tag)</td>
<td>Send data.</td>
<td>39</td>
</tr>
<tr>
<td>receive (int source, CommObjectList &amp;data, int tag, int* actualSource)</td>
<td>Receive data.</td>
<td>40</td>
</tr>
<tr>
<td>oneToAllBCast (int source, CommObjectList &amp;data)</td>
<td>Broadcast data.</td>
<td>40</td>
</tr>
<tr>
<td>allToOneGather (int target, CommObjectList &amp;data)</td>
<td>Gather data.</td>
<td>41</td>
</tr>
<tr>
<td>hRelation (CommObjectList &amp;data, int* ns)</td>
<td>Perform an h-Relation.</td>
<td>41</td>
</tr>
<tr>
<td>allToAllBCast (CommObjectList &amp;data)</td>
<td>All to all broadcast.</td>
<td>42</td>
</tr>
<tr>
<td>allToAllBCast2 (CommObjectList &amp;data)</td>
<td>A faster version of all to all broadcast.</td>
<td>42</td>
</tr>
<tr>
<td>arrayBalancing (CommObjectList &amp;data, int expectedN=-1)</td>
<td>Balancing an array.</td>
<td>43</td>
</tr>
<tr>
<td>arrayBalancing2 (CommObjectList &amp;data, int expectedN=-1)</td>
<td></td>
<td>43</td>
</tr>
</tbody>
</table>
3.1.5.1.15 virtual void

partitionCGM (int* sizes)

Partition the CGM environment into groups.

3.1.5.1.16 virtual void

partitionCGM (int groupId)

Partition the CGM environment into groups.

3.1.5.1.17 virtual void

unPartitionCGM (void)

Undo the previous partition operation.

3.1.5.1.18 virtual void

dispose (void)

Dispose the comm

To retrieve the comm. This is a static method.

Return Value: a pointer to the comm

Parameters:
argc a pointer to the command line argument
argv a pointer to the command line argument
timers a pointer to a CGMTimer (optional, default to NULL)
3.1.5.1.2

virtual int getNumberOfProcessors (void)

Get the number of processors in the current comm.

Get the number of processors in the current comm. Note that the number
of processors changes if you call partitionCGM and unPartitionCGM. Do not
cache this value if you need to use the above mentioned methods.

Return Value: the number of processor in the current comm
See Also: partitionCGM unPartitionCGM

3.1.5.1.3

virtual int getMyId (void)

Get the id of the current processor.

Get the id of the current processor. Note that the number of processors
changes if you call partitionCGM and unPartitionCGM. Do not cache this value
if you need to use the above mentioned methods.

Return Value: the id.
See Also: partitionCGM unPartitionCGM

3.1.5.1.4

virtual CGMTimers* getTimers (void)

Get the timers that are associated with the comm
Get the timers that are associated with the comm.

Return Value: the timers.

3.1.5.1.5

`virtual void synchronize(void)`

Synchronization among all processors.

Synchronization among all processors. Note that this is a collective method. That means all processors in the current comm must call this method at the same time.

3.1.5.1.6

`virtual int send(int target, CommObjectList &data, int tag)`

Send data.

Send data.

Return Value: the number of CommObject’s sent.

Parameters:

- `target` the id of the target processor.
- `data` the data to be sent.
- `tag` a tag to label the message.

See Also: receive (→3.1.5.1.7, page 40)
## 3.1.5.1.7

virtual int receive (int source, CommObjectList &data, int tag, int* actualSource)

*Receive data.*

**Return Value:**
the number of CommObject’s received.

**Parameters:**
- **source**: the id of the source processor.
- **data**: the data to be received.
- **tag**: a tag to label the message.
- **actualSource**: the processor that actually sent the data.

**See Also:**
send (→ 3.1.5.1.6, page 39)

## 3.1.5.1.8

virtual int oneToAllBCast (int source, CommObjectList &data)

*Broadcast data.*

Broadcast data. Note that this is a collective method. That means all processors in the current comm must call this method at the same time.

**Return Value:**
the number of CommObject’s sent/received.

**Parameters:**
- **source**: the id of the source processor.
- **data**: the data to be broadcast.
3.1.5.1.9

virtual int allToOneGather (int target, CommObjectList &data)

Gather data.

Gather data. Note that this is a collective method. That means all processors in the current comm must call this method at the same time.

Return Value: the number of CommObject’s sent/received.

Parameters:
- target the id of the target processor.
- data the data to be sent.

3.1.5.1.10

virtual int hRelation (CommObjectList &data, int* ns)

Perform an h-Relation.

Perform an h-Relation. Note that this is a collective method. That means all processors in the current comm must call this method at the same time.

Return Value: the number of CommObject’s received.

Parameters:
- data the data to be routed.
- ns the number of CommObjects to be sent to each processor.
3.1.5.1.11

virtual int allToAllBCast (CommObjectList &data)

*All to all broadcast.*

All to all broadcast. Applicable only in environment with scalability of 1 or less, but will require $O(\log p)$ communication round. Note that this is a collective method. That means all processors in the current comm must call this method at the same time.

*Return Value:* the total number of CommObject’s received by each processor.

*Parameters:* data the data to be broadcast.

3.1.5.1.12

virtual int allToAllBCast2 (CommObjectList &data)

*A faster version of all to all broadcast.*

A faster version of all to all broadcast. Applicable only in environment with scalability of 0.5 or less. Note that this is a collective method. That means all processors in the current comm must call this method at the same time.

*Return Value:* the total number of CommObject’s received by each processor.

*Parameters:* data the data to be broadcast.

3.1.5.1.13

virtual int arrayBalancing (CommObjectList &data, int expectedN=-1)

This page has been automatically generated with DOC++

DOC++ is ©1995 by Roland Wunderling
Malte Zöckler
Balancing an array. Note that this is a collective method. That means all processors in the current comm must call this method at the same time.

Return Value: the final number of CommObject’s at each processor.

Parameters:
- **data**: the data to be balanced.
- **expectedN**: the desired number of CommObjects eventually at each processor (optional, default to as even as possible).

```cpp
virtual int arrayBalancing2 (CommObjectList &data, int expectedN=-1)
```

Balancing an array. Note that this is a collective method. That means all processors in the current comm must call this method at the same time. This is a simpler version of arrayBalancing that is applicable only in environment with scalability of 0.5 or less.

Return Value: the final number of CommObject’s at each processor.

Parameters:
- **data**: the data to be balanced.
- **expectedN**: the desired number of CommObjects eventually at each processor (optional, default to as even as possible).
Partition the CGM environment into groups.

Partition the CGM environment into groups. Note that this is a collective method. That means all processors in the current comm must call this method at the same time.

Parameters:  sizes an integer array to represent how many processors each new group should have.

Partition the CGM environment into groups.

Partition the CGM environment into groups. Note that this is a collective method. That means all processors in the current comm must call this method at the same time.

Parameters:  groupId an integer to represent the group that the processor wish to belong to.

Undo the previous partition operation.
Undo the previous partition operation. Note that this is a collective method. That means all processors in the current comm must call this method at the same time.

### 3.1.5.1.18

```cpp
virtual void dispose (void)
```

Dispose the comm

---

Implementation of CommObject for Simple C++ data types

### Inheritance

```
CommObject
```

```
SimpleCommObject
```

---

### Public Members

3.1.6.1 `constructors` ................................. 46
3.1.6.2 `methods` ................................. 47
3.1.6.3 `destructor` ................................. 50

Implementation of CommObject for Simple C++ data types

---
3.1.6.1 constructors

Names

3.1.6.1.1 SimpleCommObject (T t)

*Conversion constructor.*

3.1.6.1.2 SimpleCommObject (SimpleCommObject &another)

*Copy constructor.*

- **Parameters:**
  - *t* the data to be pre-populated into the object.
  - *another* a reference to another SimpleCommObject of the same type.
### 3.1.6.2 methods

**Names**

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>virtual void copyToArray (char* array, int offset)</code></td>
<td>Copy the content of the object to an array.</td>
</tr>
<tr>
<td><code>virtual void copyFromArray (char* array, int offset)</code></td>
<td>Copy from an array the content of this object.</td>
</tr>
<tr>
<td><code>virtual CommObject* clone (void* p = NULL)</code></td>
<td>Clone the object.</td>
</tr>
<tr>
<td><code>virtual CommObject&amp; operator= (CommObject &amp;another)</code></td>
<td>Assignment operator.</td>
</tr>
<tr>
<td><code>virtual void sendToOstream (ostream &amp;outstream)</code></td>
<td>Send the content of this object to the ostream.</td>
</tr>
<tr>
<td><code>virtual int getSize ()</code></td>
<td>Get the size of the object when &quot;serialized&quot;.</td>
</tr>
<tr>
<td><code>virtual T&amp; getData (void)</code></td>
<td>Retrieve the data.</td>
</tr>
<tr>
<td><code>virtual void setData (T t)</code></td>
<td>Set the data.</td>
</tr>
</tbody>
</table>

---

### 3.1.6.2.1 virtual void `copyToArray (char* array, int offset)`

*Copy the content of the object to an array.*
Copy the content of the object to an array.

**Parameters:**
- `target` a character array.
- `offset` the offset of the array to start copy to.

### 3.1.6.2.2

```cpp
virtual void copyFromArray (char* array, int offset)
```

*Copy from an array the content of this object.*

Copy from an array the content of this object.

**Parameters:**
- `target` a character array.
- `offset` the offset of the array to start copy from.

### 3.1.6.2.3

```cpp
virtual CommObject* clone (void* p = NULL)
```

*Clone the object.*

Clone the object.

**Return Value:** the address of clone.

**Parameters:**
- `p` the memory that the object is suppose to clone into.
3.1.6.2.4

virtual CommObject& operator= (CommObject &another)

Assignment operator.

Return Value: a reference to the object itself.
Parameters: another a reference to another CommObject.

3.1.6.2.5

virtual void sendToOstream (ostream &outstream)

Send the content of this object to the ostream.

Parameters: outstream a reference to the ostream.

3.1.6.2.6

virtual int getSize ()

get the size of the object when "serialized".

Return Value: the size in byte.
3.1.6.2.7

virtual T& getData (void)

Retrive the data.

Retrive the data. Note this is a non-interface method.

Return Value: a reference to the data.

3.1.6.2.8

virtual void setData (T t)

Set the data.

Set the data. Note this is a non-interface method.

Parameters: t the data to be set.

3.1.6.3

destructor

Names

3.1.6.3.1virtual ~SimpleCommObject (void)

Destructor of the class ........... 51
3.1.6.3.1

virtual ~SimpleCommObject (void)

Destructor of the class

Destructor of the class

3.2

Parallel Prefix Sum

Names
3.2.1 class ParallelPrefixSummer

Calculate prefix sum .......... 51

3.2.1

class ParallelPrefixSummer

Calculate prefix sum

Public Members
3.2.1.1 constructor ......................... 52
3.2.1.2 method ......................... 52
3.2.1.3 destructor ......................... 53

Calculate prefix sum
3.2.1.1 constructor

Names
3.2.1.1.1 ParallelPrefixSummer (Comm* c, Operator* oper)

Constructor of the class.

Parameters:
- c: a pointer to the comm.
- oper: a pointer to the operator.

3.2.1.2 method

Names
3.2.1.2.1 virtual void calculatePrefixSum (CommObjectList &result, CommObjectList &data)

Preform the prefix sum calculation.

................. 53
3.2.1.2.1

virtual void calculatePrefixSum (CommObjectList &result, CommObjectList &data)

Preform the prefix sum calculation.

Parameters:
result the result.
data the data.

3.2.1.3

destructor

Names
3.2.1.3.1 virtual ~ParallelPrefixSummer (void)

Destructor of the class

3.2.1.3.1

virtual ~ParallelPrefixSummer (void)

Destructor of the class
3.3 Parallel Sorting

Names

3.3.1 class ParallelSorter Sort data .......................... 54

3.3.1 class ParallelSorter

Public Members

3.3.1.1 constructors ................................. 54

3.3.1.2 methods ............................... 55

3.3.1.3 destructor ............................... 59

Sort data

3.3.1.1 constructors

Names

3.3.1.1.1 ParallelSorter (Sorter* s, Comm* com)
Constructor of the class. ....... 55

3.3.1.1.2 ParallelSorter (Comparator* c, Comm* com)
Constructor of the class. ....... 55
3.3.1.1.1

**ParallelSorter (Sorter* s, Comm* com)**

*Constructor of the class.*

Parameters:
- `s` a pointer to a sorter.
- `com` a pointer to the comm.

3.3.1.1.2

**ParallelSorter (Comparator* c, Comm* com)**

*Constructor of the class.*

Parameters:
- `c` a pointer to a comparator.
- `com` a pointer to the comm.

3.3.1.2

**methods**

Names

3.3.1.2.1 static void **setDefaultScalability** (double s)

*Set the default scalability for all parallel sorters.*

56

3.3.1.2.2 static double
getDefaultScalability (void)
Get the default scalability for all parallel sorters. .......... 57

3.3.1.2.3 void setScalability (double s)
Set the scalability for this parallel sorter. ................. 57

3.3.1.2.4 double getScalability (void)
Get the scalability for this parallel sorter. ................. 57

3.3.1.2.5 void setComparator (Comparator* c)
Change the comparator. ........ 58

3.3.1.2.6 void sort (CommObjectList &data)
Sort the data. ................. 58

3.3.1.2.7 void sort2 (CommObjectList &data)
Sort the data. ................. 58

3.3.1.2.1

static void setDefaultScalability (double s)

Set the default scalability for all parallel sorters.

Parameters: s the scalability - the only valid values are 0.5 and 1.0, if any other value is passed in, the result is undefined.

3.3.1.2.2

static double getDefaultScalability (void)

Get the default scalability for all parallel sorters.
Get the default scalability for all parallel sorters.

Return Value: the default scalability.

### 3.3.1.2.3

```c
void setScalability (double s)
```

Set the scalability for this parallel sorter.

Set the scalability for this parallel sorter.

Return Value: the newly set scalability.

Parameters:
- `s` the scalability - the only valid values are 0.5 and 1.0, if any other value is passed in, the result is undefined.

### 3.3.1.2.4

```c
double getScalability (void)
```

Get the scalability for this parallel sorter.

Get the scalability for this parallel sorter.

Return Value: the scalability.
3.3.1.2.5

void setComparator (Comparator* c)

Change the comparator.

Parameters: comp the new comparator.

3.3.1.2.6

void sort (CommObjectList &data)

Sort the data.

Parameters: data the data to be sorted.

3.3.1.2.7

void sort2 (CommObjectList &data)

Sort the data.

Sort the data. This is a simpler version of sort that is applicable only in environment with scalability of 0.5 or less.

Parameters: data the data to be sorted.
3.3.1.3

**destructor**

Names

3.3.1.3.1  `~ParallelSorter () Destructor of the class ........... 59`

3.3.1.3.1  `~ParallelSorter ()`  

*Destructor of the class*

**RequestSystems**

Names

3.4  class  **BasicRequest**  : public Request  

*Implement a basic request ........ 60*

3.4  class  **Request**  : public CommObject  

*Interface for an enquiry request. ........ 65*

3.4  class  **RequestSystem**  A class to send requests and responses. ................. 69

3.4  class  **Response**  : public CommObject  

*Interface for an response. ........ 71*

3.4  template<T, int N>  class  **SimpleRequest**  : public Request  

*A templated class to implement a data carried request. ........... 75*

3.4  template<T, int N>  class
SimpleResponse: public Response
A templated class to implement a data carried response 81

3.4.1

class BasicRequest: public Request

Implement a basic request

Inheritance

3.1.3 CommObject
3.4.2 Request
3.4.1 BasicRequest

Public Members
3.4.1.1 constructors ........................................ 60
3.4.1.2 methods ............................................... 62
3.4.1.3 destructor ............................................ 64

Implement a basic request

3.4.1.1

constructors
### 3.4.1.1.1 BasicRequest (int q, int r)

*Basic constructor.*

**Parameters:**
- `q` the question asker.
- `r` the response provider.

### 3.4.1.1.2 BasicRequest (Request &another)

*Copy constructor.*

**Parameters:**
- `another` a reference to another BasicRequest of the same type.
3.4.1.2 methods

Names

3.4.1.2.1 virtual void copyToArray (char* target, int offset)
   Copy the content of the object to an array. ...................... 62

3.4.1.2.2 virtual void copyFromArray (char* target, int offset)
   Copy from an array the content of this object. ................... 63

3.4.1.2.3 virtual CommObject* clone (void* p = NULL)
   Clone the object. ..................... 63

3.4.1.2.4 virtual CommObject& operator= (CommObject& another)
   Assignment operator. ............... 63

3.4.1.2.5 virtual void sendToOstream (ostream& outstream)
   Send the content of this object to the ostream. .................... 64

3.4.1.2.6 virtual int getSize ()
   get the size of the object when "serialized". ...................... 64

---

3.4.1.2.1 virtual void copyToArray (char* target, int offset)

Copy the content of the object to an array.

Parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>target</td>
<td>a character array.</td>
</tr>
<tr>
<td>offset</td>
<td>the offset of the array to start copy to.</td>
</tr>
</tbody>
</table>
### 3.4.1.2.2

```cpp
virtual void copyFromArray (char* target, int offset)
```

*Copy from an array the content of this object.*

Copy from an array the content of this object.

**Parameters:**
- `target` a character array.
- `offset` the offset of the array to start copy from.

### 3.4.1.2.3

```cpp
virtual CommObject* clone (void* p = NULL)
```

*Clone the object.*

Clone the object.

**Return Value:** the address of clone.

**Parameters:**
- `p` the memory that the object is suppose to clone into.

### 3.4.1.2.4

```cpp
virtual CommObject& operator= (CommObject &another)
```

*Assignment operator.*

Assignment operator.
Return Value: a reference to the object itself.
Parameters: another a reference to another CommObject.

3.4.1.2.5

virtual void sendToOstream (ostream& outstream)

Send the content of this object to the ostream.

Send the content of this object to the ostream.
Parameters: outstream a reference to the ostream.

3.4.1.2.6

virtual int getSize ()

get the size of the object when "serialized".

get the size of the object when "serialized".
Return Value: the size in byte.

3.4.1.3
destructor

Names
3.4.1.3.1virtual ~BasicRequest (void)

Destructor of the class ........... 65
3.4.1.3.1

virtual ~BasicRequest (void)

Destructor of the class

3.4.2

class Request : public CommObject

Interface for an enquiry request.

Inheritance

CommObject

Request

3.4.5 SimpleRequest

3.4.1 BasicRequest

Public Members

3.4.2.1 methods ........................................... 66
3.4.2.2 destructor ........................................... 68

Interface for an enquiry request. This interface defines an enquiry request. That is, it represents a request for information. This can be useful if a node in a tree wants to know who its grandparent is. The node can send a request to its parent and ask it to send back its parent as a response.
3.4.2.1  

methods

Names

3.4.2.1.1 int  getQuestionAsker (void)  

Get the question asker (request sender).  66

3.4.2.1.2 int  getResponseProvider (void)  

Get the response provider (request receiver).  67

3.4.2.1.3 int  getConsolidationFactor (void)  

Get the consolidation factor (how many requests have been consolidated into this request).  67

3.4.2.1.4 void  setQuestionAsker (int q)  

Set the question asker (request sender).  67

3.4.2.1.5 void  setResponseProvider (int r)  

Set the response provider (request receiver).  68

3.4.2.1.1  

int  getQuestionAsker (void)  

Get the question asker (request sender).

Return Value:  an integer to represent the question asker.
3.4.2.1.2

int getResponseProvider (void)

Get the response provider (request receiver).

Get the response provider (request receiver).

Return Value: an integer to represent the response provider.

3.4.2.1.3

int getConsolidationFactor (void)

Get the consolidation factor (how many requests have been consolidated into this request).

Get the consolidation factor (how many requests have been consolidated into this request).

Return Value: the consolidation factor.

3.4.2.1.4

void setQuestionAsker (int q)

Set the question asker (request sender).

Set the question asker (request sender).
Parameters: 

q an integer to represent the question asker.

3.4.2.1.5

void setResponseProvider (int r)

Set the response provider (request receiver).

Parameters: 

r an integer to represent the response provider.

3.4.2.2

destructor

Names

3.4.2.2.1 virtual ~Request (void) Destructor of the class 68

3.4.2.2.1

virtual ~Request (void)

Destructor of the class
3.4.3  

**class RequestSystem**

*A class to send requests and responses.*

**Public Members**

3.4.3.1  **methods**

A class to send requests and responses. This request system provides a way to send message to a CommObject. It will process the requests through a consolidation step. This is to prevent a CommObject to receive too many requests in a communication round, and thus violating the CGM requirement that each h-relation is limited to a size of h=O(n/p).

3.4.3.1  **methods**

**Names**

3.4.3.1.1  **static void sendRequests** (CommObjectList &requests, CommObjectList &consolidatedRequests, int* lastIndices, Comm* comm, Operator* op = NULL) 

*Route a set of requests to the destination.*

3.4.3.1.2  **static void sendResponses** (CommObjectList &consolidatedResponses, CommObjectList &responses, CommObjectList &requests, int* lastIndices, Comm* comm) 

*Route a set of responses back to the original request senders.*
3.4.3.1.1

static void sendRequests (CommObjectList &requests, CommObjectList &consolidatedRequests, int* lastIndices, Comm* comm, Operator* op = NULL)

ROUTE a set of requests to the destination.

Route a set of requests to the destination. The response providers should process the requests through consolidatedRequests. The response providers should be sorted by some order (that the integer key of sorting should be used in the response provider field of the requests. The consolidated requests are guaranteed to be sorted by the response provider field and in the processor where the response providers reside. Also noted that in the consolidated requested, the question asker is meaningless for end user but is important to the system if response is to be routed back, therefore make sure this is not changed while being processed. Also the original requests will be needed to expand the responses, so it is very important that they are not changed.

Parameters:

- **requests** the original requests.
- **consolidatedRequests** the consolidated requests.
- **lastIndices** an integer array to represent the (unique) last index of request receiver in each processor. This is needed for the system to route the requests.
- **comm** a pointer to the comm.
- **op** a user provided operator that can be used to consolidate the data carried by the requests. (Optional, default to NULL)
3.4.3.1.2

static void sendResponses(CommObjectList &consolidatedResponses, CommObjectList &responses, CommObjectList &requests, int* lastIndices, Comm* comm)

Route a set of responses back to the original request senders.

Route a set of responses back to the original request senders. The user should form the responses in the consolidated responses using information available in the consolidated requests. The question askers (response receiver) should be sorted by the index that is used to represent the question askers in the requests. After the responses are sent, the question askers should process the responses that is guaranteed to be expanded back into a one-to-one relationship with the request (i.e. each request will get exactly one response).

Parameters:

- consolidatedResponses: the consolidated responses
- responses: the expanded responses
- requests: the original requests. This is needed for expanding the consolidated responses.
- lastIndices: an integer array to represent the (unique) last index of response receiver in each processor. This is needed for the system to route the responses.
- comm: a pointer to the comm.

3.4.4

class Response : public CommObject

Interface for an response.
Inheritance

Public Members

3.4.4.1 methods ............................................. 72
3.4.4.2 destructor ............................................. 75

Interface for an response. This interface defines a response.

3.4.4.1 methods

Names

3.4.4.1.1 int getQuestionAsker (void)
Get the question asker (response receiver). .................... 73

3.4.4.1.2 int getResponseProvider (void)
Get the response provider (response sender). .................... 73

3.4.4.1.3 void setRequest (Request* r)
Set the question asker and the response provider according to the information in the request. ...... 74

3.4.4.1.4 void setQuestionAsker (int q)
Set the question asker (response receiver). ..................... 74

3.4.4.1.5 void setResponseProvider (int r)
3.4.4.1.1

int getQuestionAsker (void)

Get the question asker (response receiver).

Get the question asker (response receiver).

Return Value: an integer to represent the question asker.

3.4.4.1.2

int getResponseProvider (void)

Get the response provider (response sender).

Get the response provider (response sender).

Return Value: an integer to represent the response provider.

3.4.4.1.3

void setRequest (Request* r)

Set the question asker and the response provider according to the information in the request.
Set the question asker and the response provider according to the information in the request.

**Parameters:**
- \( r \) a pointer to the request.

### 3.4.4.1.4

```c
void setQuestionAsker (int q)
```

*Set the question asker (response receiver).*

Set the question asker (response receiver).

**Parameters:**
- \( q \) an integer to represent the question asker.

### 3.4.4.1.5

```c
void setResponseProvider (int r)
```

*Set the response provider (response sender).*

Set the response provider (response sender).

**Parameters:**
- \( r \) an integer to represent the response provider.
3.4.4.2 destructor

Names
3.4.4.2.1 virtual `Response (void) Destructor of the class

3.4.4.2.1 virtual `Response (void)

Destructor of the class

3.4.5 template<class T, int N> class SimpleRequest : public Request

A templated class to implement a data carried request.

Inheritance

3.1.3 CommObject
3.4.2 Request
3.4.5 SimpleRequest
Public Members

### 3.4.5.1 constructors

- **SimpleRequest** (int q, int r)
  - *Basic constructor*  
  - Parameters: 
    - `q` the question asker. 
    - `r` the response provider.

### 3.4.5.2 methods

### 3.4.5.3 destructor

A templated class to implement a data carried request. Note that during the request consolidation process, the data carried by some requests will be lost. This may still be useful to implement a action setting requests to simulate a CRCW PRAM algorithm, if the write conflict is resolved by letting a random process to win.
3.4.5.1.2

**SimpleRequest** (Request &another)

*Copy constructor.*

Copy constructor.

**Parameters:**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>another</td>
<td>a reference to another SimpleRequest of the same type.</td>
</tr>
</tbody>
</table>

### 3.4.5.2 methods

#### Names

3.4.5.2.1 virtual void

**copyToArray** (char* target, int offset)

*Copy the content of the object to an array.* .......................... 78

3.4.5.2.2 virtual void

**copyFromArray** (char* target, int offset)

*Copy from an array the content of this object.* .......................... 78

3.4.5.2.3 virtual CommObject*

**clone** (void* p = NULL)

*Clone the object.* .......................... 79

3.4.5.2.4 virtual CommObject&

**operator=** (CommObject &another)

*Assignment operator.* .......................... 79

3.4.5.2.5 virtual void

**sendToOstream** (ostream& outstream)

*Send the content of this object to the ostream.* .......................... 79

3.4.5.2.6 virtual int **getSize** ()

*get the size of the object when "serialized".* .......................... 80
3.4.5.2.7T  \textbf{getData} (\text{int} \; i) \quad \textit{get the data carried by the request.} \quad \ldots \quad 80

3.4.5.2.8void  \textbf{setData} (\text{int} \; i, \; \text{T} \; d) \quad \textit{set the data to be carried by the request.} \quad \ldots \quad 80

\begin{center}
\begin{tabular}{|p{5cm}|p{10cm}|}
\hline
\textbf{3.4.5.2.1} & \vspace{0.5em}
\end{tabular}
\end{center}

\begin{center}
\begin{tabular}{|p{5cm}|p{10cm}|}
\hline
\textbf{virtual} & \textbf{void \texttt{copyToArray}} (\text{char*} \; \text{target}, \; \text{int} \; \text{offset}) \\vspace{0.5em}
\end{tabular}
\end{center}

\textit{Copy the content of the object to an array.}

Copy the content of the object to an array.

\textbf{Parameters:} \hspace{1cm} \begin{tabular}{ll}
\text{target} & \text{a character array.} \\
\text{offset} & \text{the offset of the array to start copy to.} \\
\end{tabular}

\begin{center}
\begin{tabular}{|p{5cm}|p{10cm}|}
\hline
\textbf{3.4.5.2.2} & \vspace{0.5em}
\end{tabular}
\end{center}

\begin{center}
\begin{tabular}{|p{5cm}|p{10cm}|}
\hline
\textbf{virtual} & \textbf{void \texttt{copyFromArray}} (\text{char*} \; \text{target}, \; \text{int} \; \text{offset}) \\vspace{0.5em}
\end{tabular}
\end{center}

\textit{Copy from an array the content of this object.}

Copy from an array the content of this object.

\textbf{Parameters:} \hspace{1cm} \begin{tabular}{ll}
\text{target} & \text{a character array.} \\
\text{offset} & \text{the offset of the array to start copy from.} \\
\end{tabular}
3.4.5.2.3

virtual CommObject* clone (void* p = NULL)

Clone the object.

Return Value: the address of clone.
Parameters: p the memory that the object is suppose to clone into.

3.4.5.2.4

virtual CommObject& operator= (CommObject &another)

Assignment operator.

Return Value: a reference to the object itself.
Parameters: another a reference to another CommObject.

3.4.5.2.5

virtual void sendToOstream (ostream& outstream)

Send the content of this object to the ostream.

Send the content of this object to the ostream.
Parameters: outstream a reference to the ostream.

3.4.5.2.6

virtual int getSize ()

get the size of the object when "serialized".

Return Value: the size in byte.

3.4.5.2.7

T getData (int i)

get the data carried by the request.

Return Value: the data indexed by i.
Parameters: i the data index.

3.4.5.2.8

void setData (int i, T d)

set the data to be carried by the request.
set the data to be carried by the request.

**Parameters:**

- `i` the data index.
- `d` the data to be set.

### 3.4.5.3

**Destructor**

**Names**

3.4.5.3.1 virtual ~`SimpleRequest` (void)

*Destructor of the class*

3.4.6

**Template**

```cpp
template<class T, int N> class SimpleResponse : public Response
```

*A templated class to implement a data carried response*
Inheritance

Public Members

3.4.6.1 constructors ........................................ 82
3.4.6.2 .......................................................... 84
3.4.6.3 destructor ................................................ 87

A templated class to implement a data carried response

3.4.6.1 constructors

Names

3.4.6.1.1 SimpleResponse (int q, int r)

   Basic constructor ......................... 83

3.4.6.1.2 SimpleResponse (Request &r)

   Quick constructor ......................... 83

3.4.6.1.3 SimpleResponse (Response &another)

   Copy constructor ......................... 83

3.4.6.1.1 SimpleResponse (int q, int r)

   Basic constructor
Basic constructor

Parameters:
- q  the question asker.
- r  the response provider.

### 3.4.6.1.2 SimpleResponse (Request &r)

Quick constructor

Quick constructor

Parameters:
- r  the original request.

### 3.4.6.1.3 SimpleResponse (Response &another)

Copy constructor.

Copy constructor.

Parameters:
- another  a reference to another SimpleResponse of the same type.
3.4.6.2

Names

3.4.6.2.1 virtual void copyToArray (char* target, int offset)  
Copy the content of the object to an array.  

3.4.6.2.2 virtual void copyFromArray (char* target, int offset)  
Copy from an array the content of this object.  

3.4.6.2.3 virtual CommObject* clone (void* p = NULL)  
Clone the object.  

3.4.6.2.4 virtual CommObject& operator= (CommObject &another)  
Assignment operator.  

3.4.6.2.5 virtual void sendToOstream (ostream& outstream)  
Send the content of this object to the ostream.  

3.4.6.2.6 virtual int getSize ()  
get the size of the object when "serialized".  

3.4.6.2.7 T getData (int i)  
get the data carried by the response.  

3.4.6.2.8 void setData (int i, T d)  
set the data to be carried by the response.  

3.4.6.2.1 virtual void copyToArray (char* target, int offset)  
Copy the content of the object to an array.
Copy the content of the object to an array.

**Parameters:**
- `target` a character array.
- `offset` the offset of the array to start copy to.

### 3.4.6.2.2

```cpp
virtual void copyFromArray (char* target, int offset)
```

*Copy from an array the content of this object.*

Copy from an array the content of this object.

**Parameters:**
- `target` a character array.
- `offset` the offset of the array to start copy from.

### 3.4.6.2.3

```cpp
virtual CommObject* clone (void* p = NULL)
```

*Clone the object.*

Clone the object.

**Return Value:**
- the address of clone.

**Parameters:**
- `p` the memory that the object is suppose to clone into.
### 3.4.6.2.4

```cpp
virtual CommObject& operator= (CommObject &another)
```

Assignment operator.

**Return Value:** a reference to the object itself.

**Parameters:**
- `another` a reference to another CommObject.

### 3.4.6.2.5

```cpp
virtual void sendToOstream (ostream &outstream)
```

Send the content of this object to the ostream.

Send the content of this object to the ostream.

**Parameters:**
- `outstream` a reference to the ostream.

### 3.4.6.2.6

```cpp
virtual int getSize ()
```

get the size of the object when "serialized".

get the size of the object when "serialized".

**Return Value:** the size in byte.
3.4.6.2.7

T getData (int i)

get the data carried by the response.

Return Value: the data indexed by i.
Parameters: i the data index.

3.4.6.2.8

void setData (int i, T d)

set the data to be carried by the response.

Parameters: i the data index.
           d the data to be set.

3.4.6.3

destructor

Names
3.4.6.3.1virtual hSimpleResponse (void)

Destructor of the class 88
3.4.6.3.1

virtual ~SimpleResponse (void)

Destructor of the class

Destructor of the class

3.5

Utilities

Names
3.5.1 Random Number Generator  .....................  88
3.5.2 Sorting  .................................  92
3.5.3 Timers  .................................  109
3.5.4 Miscellaneous Utilities  .....................  127

3.5.1

Random Number Generator

Names
3.5.1.1 class Random  A random number generator. ...  88

3.5.1.1

class Random

A random number generator.
A random number generator. There is nothing specific to the parallel computing and/or CGM in this class. It is provided as a convenience for the user of this library.

### 3.5.1.1.1 constructors

**Names**

<table>
<thead>
<tr>
<th>3.5.1.1.1.1</th>
<th>Random (void)</th>
<th>Default constructor.</th>
<th>89</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.5.1.1.1.2</td>
<td>Random (int myId)</td>
<td>Constructor of the class.</td>
<td>90</td>
</tr>
</tbody>
</table>

**Random (void)**

Default constructor.

Default constructor. It use the current time to seed the random number generator.
**3.5.1.1.1.2**

**Random** (int myId)

Constructor of the class.

Constructor of the class. It is suggested to use the processor id for this argument. Doing this will ensure the random numbers generated at each processor are different.

**Parameters:**

myId any integer, processor id preferred

---

**3.5.1.2**

**methods**

**Names**

3.5.1.2.1 int **getInteger** (void) *Generate an integer random number in the range of 0 to RAND_MAX ........................* 90

3.5.1.2.2 int **getInteger** (int range) *Generate an integer random number in the range of 0 to range-1.* 91

3.5.1.2.3 double **getDouble** (void) *Generates a double random number in the range of [0.0, 1.0).* 91

**3.5.1.2.1**

int **getInteger** (void)

*Generate an integer random number in the range of 0 to RAND_MAX*
Generate an integer random number in the range of 0 to RAND_MAX.

3.5.1.2.2

int getInteger (int range)

Generate an integer random number in the range of 0 to range-1.

Parameters:

- range: the range of the random number

3.5.1.2.3

double getDouble (void)

Generates a double random number in the range of [0.0, 1.0).

3.5.1.3

destructor

Names

3.5.1.3.1virtual ~Random (void) Destructor of the class ......... 92
3.5.1.3.1

virtual ~Random (void)

Destructor of the class

Destructor of the class

3.5.2

Sorting

Names

3.5.2.1 class Comparator Interface for the comparator. ... 92
3.5.2.2 class HeapSorter : public Sorter Implementation of the sorter using heap sort. ................. 95
3.5.2.3 class IntegerComparator : public Comparator Interface for the integer comparator. .................. 98
3.5.2.4 class IntegerSorter : public Sorter Implementation of the sorter using integer sort. ................. 100
3.5.2.5 class ReverseComparator : public Comparator A "reverse" comparator. ......... 103
3.5.2.6 class Sorter Interface for the sorter. ........... 106

3.5.2.1

class Comparator

Interface for the comparator.
Inheritance

3.5.2.1 Comparator

3.5.2.5 ReverseComparator

3.5.2.3 IntegerComparator

Public Members

3.5.2.1.1 methods ........................................... 93
3.5.2.1.2 destructor ........................................... 94

Interface for the comparator. It defines a set of common methods for comparing two objects. There is nothing specific to the parallel computing and/or CGM in this class. It is provided as a convenience for the user of this library.

3.5.2.1.1 methods

Names

3.5.2.1.1.1virtual bool isComparable (CommObject* a)

Determine if the given object is comparable using this comparator.

........................................... 94

3.5.2.1.2virtual int compare (CommObject* a, CommObject* b)

Compare the two given objects.

........................................... 94
### 3.5.2.1.1

**virtual bool isComparable (CommObject* a)**

Determine if the given object is comparable using this comparator.

**Return Value:** true if a is comparable using this comparator, false otherwise

**Parameters:**
- a: a pointer to a CommObject

### 3.5.2.1.2

**virtual int compare (CommObject* a, CommObject* b)**

Compare the two given objects.

Compare the two given objects. It throws an exception if either of the objects is not comparable using this comparator.

**Return Value:** -1 if a<b, 0 if a=b and 1 if a>b

**Parameters:**
- a: a pointer to the first CommObject
- b: a pointer to the second CommObject

### 3.5.2.1.2

**destructor**

Names

3.5.2.1.2.1 **virtual ~Comparator (void)**

Destructor of the class  

This page has been automatically generated with DOC++

DOC++ is ©1999 by Roland Wunderling
Malte Zöckler
3.5.2.1.2.1

virtual ~Comparator (void)

Destructor of the class

Destructor of the class

3.5.2.2

class HeapSorter : public Sorter

Implementation of the sorter using heap sort.

Inheritance

3.5.2.6

Sorter

3.5.2.2

HeapSorter

Public Members

3.5.2.2.1 constructors ........................................ 96
3.5.2.2.2 methods .............................................. 96
3.5.2.2.3 destructor ............................................. 98

Implementation of the sorter using heap sort. There is nothing specific to the parallel computing and/or CGM in this class. It is provided as a convenience for the user of this library.
3.5.2.2.1

constructors

Names
3.5.2.2.1.1 HeapSorter (Comparator* c)

Constructor of the class.

HeapSorter (Comparator* c)

Constructor of the class.

Parameters:

- c a pointer to a comparator

3.5.2.2.2

methods

Names
3.5.2.2.2.1 virtual void sort (CommObjectList &data)

Sort the array of objects.

3.5.2.2.2.2 virtual Comparator* getComparator (void)

Retrieve the comparator used by the sorter.

3.5.2.2.2.3 virtual void setComparator (Comparator* c)

Set the comparator used by the sorter.
3.5.2.2.2.1

virtual void sort (CommObjectList &data)

Sort the array of objects.

Parameters:

data  the data to be sorted

3.5.2.2.2.2

virtual Comparator* getComparator (void)

Retrieve the comparator used by the sorter.

Return Value:

the  comparator

3.5.2.2.2.3

virtual void setComparator (Comparator* c)

Set the comparator used by the sorter.

Parameters:

c  the comparator
3.5.2.2.3

destructor

Names

3.5.2.3.1
virtual ~HeapSorter (void)

Destructor of the class

3.5.2.3

class IntegerComparator : public Comparator

Interface for the integer comparator.

Inheritance

3.5.2.1
Comparador

3.5.2.3
IntegerComparator
Public Members

3.5.2.3.1  method .............................................. 99

Interface for the integer comparator. It defines one additional methods to provide
d facility for integer sorting. There is nothing specific to the parallel
computing and/or CGM in this class. It is provided as a convenience for the
user of this library.

3.5.2.3.1

method

Names

3.5.2.3.1.1virtual int getIntegerKeyValue (CommObject* object)

Retrive the integer key value of the
given object.  ....................... 99

3.5.2.3.1.1

virtual int getIntegerKeyValue (CommObject* object)

Retrive the integer key value of the given object.

Return Value:  the  integer key value
Parameters:  object  a pointer to a CommObject
### 3.5.2.4

**class IntegerSorter : public Sorter**

*Implementation of the sorter using integer sort.*

#### Inheritance

- **3.5.2.6** Sorter
  - **3.5.2.4** IntegerSorter

#### Public Members

| 3.5.2.4.1 | constructors | ......................................... | 100 |
| 3.5.2.4.2 | methods      | ......................................... | 101 |
| 3.5.2.4.3 | destructor   | ......................................... | 102 |

*Implementation of the sorter using integer sort. There is nothing specific to the parallel computing and/or CGM in this class. It is provided as a convenience for the user of this library.*

#### 3.5.2.4.1 constructors

**Names**

| 3.5.2.4.1.1 | IntegerSorter (Comparator* c) |
|             | Constructor of the class.      | 101 |
3.5.2.4.1.1

**IntegerSorter** (Comparator* c)

Constructor of the class.

Constructor of the class. An exception will be thrown if the comparator is NOT an IntegerComparator.

**Parameters:**

- c a pointer to a comparator

3.5.2.4.2

**Methods**

**Names**

3.5.2.4.2.1 virtual void **sort** (CommObjectList &data)  
*Sorts the array of objects.* ...... 102

3.5.2.4.2.2 virtual Comparator* **getComparator** (void)  
*Retrieves the comparator used by the sorter.* ....................... 102

3.5.2.4.2.3 virtual void **setComparator** (Comparator* c)  
*Set the comparator used by the sorter.* .............................. 102

3.5.2.4.2.1

virtual void **sort** (CommObjectList &data)  
*Sorts the array of objects.*
Sorts the array of objects.

**Parameters:**

- **data** the data to be sorted

---

### 3.5.2.4.2.2

**virtual Comparator* getComparator (void)**

*Retrieves the comparator used by the sorter.*

Retrieves the comparator used by the sorter.

**Return Value:** the comparator

---

### 3.5.2.4.2.3

**virtual void setComparator (Comparator* c)**

*Set the comparator used by the sorter.*

Set the comparator used by the sorter.

**Parameters:**

- **c** the comparator

---

### 3.5.2.4.3

**destructor**
Names

3.5.2.4.3.1 virtual ~IntegerSorter (void)

Destructor of the class

3.5.2.4.3.1

virtual ~IntegerSorter (void)

Destructor of the class

3.5.2.5

class ReverseComparator : public Comparator

A ”reverse” comparator.

Inheritance

3.5.2.1
Comparator

3.5.2.5
ReverseComparator

Public Members

3.5.2.5.1 constructors ........................................... 104
3.5.2.5.2 methods ............................................... 104
3.5.2.5.3 destructor ............................................. 106
A "reverse" comparator. This comparator will reverse the result of the original comparator. In addition to just for fun, it is needed for the in place heap sort. There is nothing specific to the parallel computing and/or CGM in this class. It is provided as a convenience for the user of this library.

See Also: HeapSorter (§3.5.2.2, page 95)

### 3.5.2.5.1 constructors

#### Names

<table>
<thead>
<tr>
<th>Section</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.5.2.5.1.1</td>
<td>ReverseComparator (Comparator* c)</td>
<td>Constructor of the class.</td>
</tr>
</tbody>
</table>

Constructor of the class.

**Parameters:**

- **c** a pointer to a comparator

### 3.5.2.5.2 methods

#### Names

<table>
<thead>
<tr>
<th>Section</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.5.2.5.2.1</td>
<td>virtual bool</td>
</tr>
</tbody>
</table>
isComparable (CommObject* a)

Determines if the given object is comparable using this comparator.

Parameters:
- a: a pointer to a CommObject

Return Value:
- true if a is comparable using this comparator, false otherwise

compare (CommObject* a, CommObject* b)

Compare the two objects. It throws an exception if either of the objects is not comparable using this comparator.

Parameters:
- a: a pointer to the first CommObject
- b: a pointer to the second CommObject

Return Value:
- -1 if a < b, 0 if a = b and 1 if a > b
3.5.2.5.3

destructor

Names
3.5.2.5.3.1 virtual ~ReverseComparator (void)

Destructor of the class

3.5.2.6

class Sorter

Interface for the sorter.

Inheritance

3.5.2.6
Sorter

3.5.2.4
IntegerSorter

3.5.2.2
HeapSorter
Public Members

3.5.2.6.1 methods ........................................ 107
3.5.2.6.2 destructor ....................................... 108

Interface for the sorter. It defines a set of common methods for sorting an
array of objects. There is nothing specific to the parallel computing and/or
CGM in this class. It is provided as a convenience for the user of this library.

3.5.2.6.1

methods

Names

3.5.2.6.1.1 virtual void

sort (CommObjectList &data)
Sorts the array of objects. ....... 107

3.5.2.6.1.2 virtual Comparator*

getComparator (void)
Retrieves the comparator used by
the sorter. ....................... 108

3.5.2.6.1.3 virtual void

setComparator (Comparator* c)
Set the comparator used by the
sorter. ......................... 108

3.5.2.6.1.1

virtual void sort (CommObjectList &data)

Sorts the array of objects.

This page has been automatically generated with DOC++

DOC++ is ©1995 by Roland Wunderling
Malte Zöckler

107
3 API

Parameters: data the data to be sorted

3.5.2.6.1.2

virtual Comparator* getComparator (void)

Retrives the comparator used by the sorter.

Retrives the comparator used by the sorter.

Return Value: the comparator

3.5.2.6.1.3

virtual void setComparator (Comparator* c)

Set the comparator used by the sorter.

Set the comparator used by the sorter.

Parameters: c the comparator

3.5.2.6.2
destructor

Names
3.5.2.6.2.1virtual ~Sorter (void) Destructor of the class 109
3.5.2.6.2.1

virtual ~Sorter (void)

Destructor of the class

Destructor of the class

3.5.3

Timers

Names

3.5.3.1 class CGMTimers Combines a set of timers. ........ 109
3.5.3.2 class CPUTimer: public Timer
  Implementation of the timer using
  the UNIX CPU ticks. ............ 116
3.5.3.3 class Timer Interface for the timer. ............ 120
3.5.3.4 class UnixTimer: public Timer
  Implementation of the timer using
  the UNIX built-in timing facility.
  123

3.5.3.1

class CGMTimers

Combines a set of timers.

Public Members

3.5.3.1.1 constructors ................................. 110
3.5.3.1.2 methods ................................. 111
3.5.3.1.3 destructor ................................. 116
Combines a set of timers. This class provides methods to allow the user accessing them easily.

### 3.5.3.1.1 Constructors

#### Names

<table>
<thead>
<tr>
<th>3.5.3.1.1.1</th>
<th><code>CGMTimers (void)</code></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Default constructor</code></td>
<td>110</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>3.5.3.1.1.2</th>
<th><code>CGMTimers (CGMTimers &amp;source)</code></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Copy constructor.</code></td>
<td>110</td>
</tr>
</tbody>
</table>

#### 3.5.3.1.1.1 CGMTimers (void)

*Default constructor*

Default constructor

#### 3.5.3.1.1.2 CGMTimers (CGMTimers &source)

*Copy constructor.*

Copy constructor.

**Parameters:**

- `source` a reference to another CGMTimer
3.5.3.1.2 methods

Names

3.5.3.1.2.1 void timersStart (void)  
Start the appropriate timers. ... 112

3.5.3.1.2.2 void timersStop (void)  
Stop all the timers .......... 112

3.5.3.1.2.3 void timersReset (void)  
Stop and reset all the timers ... 112

3.5.3.1.2.4 void switchToComputation (void)  
Switch the timers to time computational activities ............... 113

3.5.3.1.2.5 void switchToCommunication (void)  
Switch the timers to time communication activities ............... 113

3.5.3.1.2.6 double getTotalElapsedSeconds (void)  
Return the total elapsed time as timed by the Unix Wall Clock. 113

3.5.3.1.2.7 double getComputationElapsedSeconds (void)  
Return the total computational time as timed by the Unix Wall Clock. ............... 114

3.5.3.1.2.8 double getCommunicationElapsedSeconds (void)  
Return the total communication time as timed by the Unix Wall Clock. ............... 114

3.5.3.1.2.9 double getTotalCPUTicksInSeconds (void)  
Return the total elapsed time as timed by the Unix CPU ticks. ... 114

3.5.3.1.2.10 double getComputationCPUTicksInSeconds (void)  
Return the total computational time as timed by the Unix CPU ticks. ............... 115

3.5.3.1.2.11 double getCommunicationCPUTicksInSeconds (void)  
Return the total communication time as timed by the Unix CPU ticks. ............... 115

3.5.3.1.2.12 bool areTimersRunning (void)
3.5.3.1.2.1

void timersStart (void)

Start the appropriate timers.

See Also: switchToComputation switchToCommunication

3.5.3.1.2.2

void timersStop (void)

Stop all the timers

3.5.3.1.2.3

void timersReset (void)

Stop and reset all the timers
3.5.3.1.2.4

```c
void switchToComputation (void)
```

*Switch the timers to time computational activities*

Switch the timers to time computational activities

3.5.3.1.2.5

```c
void switchToCommunication (void)
```

*Switch the timers to time communication activities*

Switch the timers to time communication activities

3.5.3.1.2.6

```c
double getTotalElapsedSeconds (void)
```

*Return the total elapsed time as timed by the Unix Wall Clock.*

Return the total elapsed time as timed by the Unix Wall Clock.

**Return Value:** the elapsed time in seconds
3.5.3.1.2.7

double getComputationElapsedTime (void)

Return the total computational time as timed by the Unix Wall Clock.

Return Value: the computational time in seconds

3.5.3.1.2.8

double getCommunicationElapsedTime (void)

Return the total communication time as timed by the Unix Wall Clock.

Return Value: the communication time in seconds

3.5.3.1.2.9

double getTotalCPUTicksInSeconds (void)

Return the total elapsed time as timed by the Unix CPU ticks.

Return Value: the elapsed time in seconds
3.5.3.1.2.10  
\begin{verbatim}
double getComputationCPUTicksInSeconds (void)
\end{verbatim}

*Return the total computational time as timed by the Unix CPU ticks.*

Return the total computational time as timed by the Unix CPU ticks.

**Return Value:** the computational time in seconds

3.5.3.1.2.11  
\begin{verbatim}
double getCommunicationCPUTicksInSeconds (void)
\end{verbatim}

*Return the total communication time as timed by the Unix CPU ticks.*

Return the total communication time as timed by the Unix CPU ticks.

**Return Value:** the communication time in seconds

3.5.3.1.2.12  
\begin{verbatim}
bool areTimersRunning (void)
\end{verbatim}

*Return a boolean to indicate if the timers are running.*

Return a boolean to indicate if the timers are running.

**Return Value:** true if the timers are running, false otherwise.
3.5.3.1.3

destructor

Names

3.5.3.1.3.1 ~CGMTimers (void)

Destructor of the class

class CPUTimer : public Timer

Implementation of the timer using the UNIX CPU ticks.

Inheritance

3.5.3.2

3.5.3.3

Timer

CPUTimer
3.5.3.2.1 constructors

Names

3.5.3.2.1.1 CPUTimer (void) Default constructor ............ 117
3.5.3.2.1.2 CPUTimer (CPUTimer &source)

Copy constructor. ............ 118

3.5.3.2.1.1 CPUTimer (void)

Default constructor

Implementation of the timer using the UNIX CPU ticks. There is nothing
specific to the parallel computing and/or CGM in this class. It is provided as a
convenience for the user of this library.
3.5.3.2.1.2

CPUTimer (CPUTimer &source)

Copy constructor.

Parameters:  source  a reference to another CPUPtimer

3.5.3.2.2

methods

Names
3.5.3.2.2.1virtual  void  start (void)  Start the timer  .................  118
3.5.3.2.2.2virtual  void  stop (void)  Stop the timer  .................  119
3.5.3.2.2.3virtual  void  reset (void)  Stop and reset the timer  .......  119
3.5.3.2.2.4virtual  double  getTime (void)  Return the current value of the
timer.  .........................  119

3.5.3.2.2.1

virtual  void  start (void)

Start the timer

Start the timer
3.5.3.2.2

virtual void **stop** (void)

*Stop the timer*

Stop the timer

3.5.3.2.3

virtual void **reset** (void)

*Stop and reset the timer*

Stop and reset the timer

3.5.3.2.4

virtual double **getTime** (void)

*Return the current value of the timer.*

Return the current value of the timer.

**Return Value:** the current value of the timer in seconds.
3.5.3.2.3
destructor

Names
3.5.3.2.3.1 virtual ~CPUTimer (void)

Destructor of the class

Destructor of the class

3.5.3.3
class Timer

Interface for the timer.

Inheritance

3.5.3.3
Timer

3.5.3.4
UnixTimer

3.5.3.2
CPUTimer
Public Members

3.5.3.3.1 methods ........................................ 121
3.5.3.3.2 destructor ....................................... 123

Interface for the timer. This interface defines a set of common methods for timing a task. There is nothing specific to the parallel computing and/or CGM in this class. It is provided as a convenience for the user of this library.

### 3.5.3.3.1

#### methods

<table>
<thead>
<tr>
<th>Names</th>
<th>3.5.3.3.1.1virtual</th>
<th>void</th>
<th>start (void)</th>
<th>Start the timer</th>
<th>.................</th>
<th>121</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3.5.3.3.1.2virtual</td>
<td>void</td>
<td>stop (void)</td>
<td>Stop the timer</td>
<td>.................</td>
<td>122</td>
</tr>
<tr>
<td></td>
<td>3.5.3.3.1.3virtual</td>
<td>void</td>
<td>reset (void)</td>
<td>Stop and reset the timer</td>
<td>...........</td>
<td>122</td>
</tr>
<tr>
<td></td>
<td>3.5.3.3.1.4virtual</td>
<td>double</td>
<td>getTime (void)</td>
<td>Return the current value of the timer.</td>
<td>.................</td>
<td>122</td>
</tr>
</tbody>
</table>

#### 3.5.3.3.1.1

<table>
<thead>
<tr>
<th>virtual</th>
<th>void</th>
<th>start (void)</th>
</tr>
</thead>
</table>

Start the timer

Start the timer
3.5.3.3.1.2

virtual void **stop** (void)

*Stop the timer*

Stop the timer

3.5.3.3.1.3

virtual void **reset** (void)

*Stop and reset the timer*

Stop and reset the timer

3.5.3.3.1.4

virtual double **get**Time (void)

*Return the current value of the timer.*

Return the current value of the timer.

Return Value: the current value of the timer in seconds.
3.5.3.3.2

destructor

Names
3.5.3.3.2.1

virtual ~Timer (void) Destructor of the class

3.5.3.4

class UnixTimer : public Timer

Implementation of the timer using the UNIX built-in timing facility.

Inheritance

3.5.3.3

Timer

3.5.3.4

UnixTimer
Implementation of the timer using the UNIX built-in timing facility. There is nothing specific to the parallel computing and/or CGM in this class. It is provided as a convenience for the user of this library.

### 3.5.3.4.1 constructors

**Names**

3.5.3.4.1.1 `UnixTimer` (void) *Default constructor* ............ 124
3.5.3.4.1.2 `UnixTimer` (UnixTimer &source) *Copy constructor* .......... 125

#### 3.5.3.4.1.1 UnixTimer (void)

*Default constructor*

Default constructor
3.5.3.4.1.2

UnixTimer (UnixTimer &source)

Copy constructor.

Copy constructor.

Parameters: source a reference to another UnixTimer

3.5.3.4.2

methods

Names

3.5.3.4.2.1 virtual void start (void) Start the timer ............... 125
3.5.3.4.2.2 virtual void stop (void) Stop the timer .................. 126
3.5.3.4.2.3 virtual void reset (void) Stop and reset the timer ....... 126
3.5.3.4.2.4 virtual double getTime (void) Return the current value of the timer. ...................... 126

3.5.3.4.2.1

virtual void start (void)

Start the timer
3.5.3.4.2.2

virtual void stop (void)

Stop the timer

3.5.3.4.2.3

virtual void reset (void)

Stop and reset the timer

3.5.3.4.2.4

virtual double getTime (void)

Return the current value of the timer.

Return Value: the current value of the timer in seconds.
3.5.3.4.3
destructor

Names
3.5.3.4.3.1 virtual ~UnixTimer (void)

Destructor of the class

3.5.4
Miscellaneous Utilities

Names
3.5.4.1 class Debug Debug utilities. .................. 128
3.5.4.2 class GeneralUtilities General utilities for use by the users of this library ............. 130
3.5.4.3 template<class T> class ObjList A container to encapsulate a Obj-list of objects. ................. 136
3.5.4.4 class Operator Interface for the operator to preform prefix sum. .................. 144
3.5.4.1 class Debug

Debug utilities. This class provides facilities for the user to debug their application.

Public Members
3.5.4.1.1 methods ........................................... 128

Debug utilities. This class provides facilities for the user to debug their application.

Names
3.5.4.1.1.1 static void  
setDebugStatus (bool status)  
   Enable or disable the debug facilities.  ....................... 129

3.5.4.1.1.2 static bool  
getDebugStatus (void)  
   Query if the debug facilities have been enabled.  ............ 129

3.5.4.1.1.3 static void  
printArray (CommObjectList &data,  
    char* label=NULL, int procId=-1)  
   Print the array if and only if the debug facilities are enabled.  .... 129

3.5.4.1.1.4 static void  
printArray (int* data, int n, char* label=NULL,  
    int procId=-1)  
   Print the array if and only if the debug facilities are enabled.  .... 130
3.5.4.1.1

static void setDebugStatus (bool status)

Enable or disable the debug facilities.

Enable or disable the debug facilities.

Parameters:

status: true to enable the debug facilities, false to disable.

3.5.4.1.2

static bool getDebugStatus (void)

Query if the debug facilities have been enabled.

Query if the debug facilities have been enabled.

Return Value:

true if the debug facilities is enabled, false otherwise.

3.5.4.1.3

static void printArray (CommObjectList &data, char* label=NULL, int procId=-1)

Print the array if and only if the debug facilities are enabled.

Print the array if and only if the debug facilities are enabled.
Parameters:

- **data**: the data to be printed.
- **label**: an optional label to identify the data.
- **procId**: the processor id (optional, default to not printed)

```c
static void printArray (int* data, int n, char* label=NULL, int procId=-1)
```

Print the array if and only if the debug facilities are enabled.

Print the array if and only if the debug facilities are enabled.

Parameters:

- **data**: the data to be printed.
- **n**: the number of data to be printed.
- **label**: an optional label to identify the data.
- **procId**: the processor id (optional, default to not printed)

```c
3.5.4.2
```

```c
class GeneralUtilities
```

*General utilities for use by the users of this library*

### Public Members

<table>
<thead>
<tr>
<th>3.5.4.2.1</th>
<th><strong>methods</strong></th>
</tr>
</thead>
</table>

*General utilities for use by the users of this library*
### 3.5.4.2.1 Methods

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Names</td>
<td></td>
</tr>
<tr>
<td>3.5.4.2.1.1 static void</td>
<td><code>printTimers</code> (CGMTimers* timers, int id=-1) &lt;br&gt;Print out the timer value.</td>
</tr>
<tr>
<td>3.5.4.2.1.2 static void</td>
<td><code>printAverageTimes</code> (CGMTimers* timers, Comm* comm, int factor = 1) &lt;br&gt;Print out the timer value averaging across all processors.</td>
</tr>
<tr>
<td>3.5.4.2.1.3 static void</td>
<td><code>writeDataToFile</code> (CommObjectList &amp;data, char* filename, char* suffix=&quot;txt&quot;, int id=-1) &lt;br&gt;Write data to file.</td>
</tr>
<tr>
<td>3.5.4.2.1.4 static double</td>
<td><code>getAverage</code> (double value, Comm* comm) &lt;br&gt;Calculate the average provided by each processor.</td>
</tr>
<tr>
<td>3.5.4.2.1.5 template&lt;class T&gt; static T</td>
<td><code>getTotal</code>(T value, Comm* comm) &lt;br&gt;Calculate the total provided by each processor (Note, used only on scalar type such as int, long, float, double)</td>
</tr>
<tr>
<td>3.5.4.2.1.6 template&lt;class T&gt; static T</td>
<td><code>getMaximum</code>(T value, Comm* comm) &lt;br&gt;Calculate the maximum provided by each processor (Note, used only on scalar type such as int, long, float, double, char)</td>
</tr>
<tr>
<td>3.5.4.2.1.7 template&lt;class T&gt; static T</td>
<td><code>getMinimum</code>(T value, Comm* comm) &lt;br&gt;Calculate the minimum provided by each processor (Note, used only on scalar type such as int, long, float, double, char)</td>
</tr>
<tr>
<td>3.5.4.2.1.8 static bool</td>
<td></td>
</tr>
</tbody>
</table>
getAND (bool value, Comm* comm)

Calculate the AND value of a boolean value provided by each processor

3.5.4.2.1.9 static bool

getOR (bool value, Comm* comm)

Calculate the OR value of a boolean value provided by each processor

3.5.4.2.1.1

static void printTimers (CGMTimers* timers, int id=-1)

Print out the timer value.

Parameters:

- timers: a pointer to the CGMTimers used to time the events.
- id: the processor id (optional, default to not printing the id).

3.5.4.2.1.2

static void printAverageTimes (CGMTimers* timers,
Comm* comm, int factor = 1)

Print out the timer value averaging across all processors.

Print out the timer value averaging across all processors.
Parameters:  

- **timers**: a pointer to the CGMTimers used to time the events.
- **comm**: a pointer to the Comm.
- **factor**: a scale down factor (optional, default to 1)

### 3.5.4.2.1.3

```c
static void writeDataToFile (CommObjectList &data,  
                            char* filename, char* suffix="txt", int id=-1)
```

*Write data to file.*

Write data to file. The file name will be "<filename>.<id>.<suffix>" or "<filename>.<suffix>".

**Parameters:**

- **data**: the data to be written out to file.
- **filename**: the file name (include path if desired).
- **suffix**: the suffix of the file (optional, default to "txt").
- **id**: the processor id, will be used as part of file name (optional, default to not using id in filename).

### 3.5.4.2.1.4

```c
static double getAverage (double value, Comm* comm)
```

*Calculate the average provided by each processor*

Calculate the average provided by each processor.
Return Value: the average
Parameters: value the input number
            comm the Comm

3.5.4.2.1.5

```cpp
template<class T> static T getTotal (T value, Comm* comm)
```

Calculate the total provided by each processor (Note, used only on scalar type such as int, long, float, double)

Return Value: the total
Parameters: value the input number
            comm the Comm

3.5.4.2.1.6

```cpp
template<class T> static T getMaximum (T value, Comm* comm)
```

Calculate the maximum provided by each processor (Note, used only on scalar type such as int, long, float, double, char)

Return Value: the maximum
Parameters: value the input number
            comm the Comm
3.5.4.2.1.7

```cpp
template<class T>
static T getMinimum (T value, Comm* comm)
```

*Calculate the minimum provided by each processor (Note, used only on scalar type such as int, long, float, double, char)*

Calculate the minimum provided by each processor (Note, used only on scalar type such as int, long, float, double, char)

**Return Value:** the minimum  
**Parameters:**  
- `value` the input number  
- `comm` the Comm

3.5.4.2.1.8

```cpp
static bool getAND (bool value, Comm* comm)
```

*Calculate the AND value of a boolean value provided by each processor*

Calculate the AND value of a boolean value provided by each processor

**Return Value:** the AND value  
**Parameters:**  
- `value` the input number  
- `comm` the Comm

3.5.4.2.1.9

```cpp
static bool getOR (bool value, Comm* comm)
```

*Calculate the OR value of a boolean value provided by each processor*

Calculate the OR value of a boolean value provided by each processor

**Return Value:** the OR value  
**Parameters:**  
- `value` the input number  
- `comm` the Comm
Calculate the OR value of a boolean value provided by each processor

**Return Value:**  
the OR value

**Parameters:**  
value the input number  
comm the Comm

### 3.5.4.3

```
template<class T> class ObjList
```

*A container to encapsulate a Objlist of objects.*

#### Inheritance

- **3.5.4.3**  
  ObjList
- **3.1.4**  
  CommObjectList

#### Public Members

- **3.5.4.3.1**  
  constructors  
  --------------------------  
  137
- **3.5.4.3.2**  
  methods  
  --------------------------  
  138
- **3.5.4.3.3**  
  destructor  
  --------------------------  
  143

*A container to encapsulate a Objlist of objects. This container will manage all the memory location it owns. It will never claim ownership on any pointer passed to it.*
3.5.4.3.1

constructors

Names
3.5.4.3.1.1  **ObjList** (T* defaultData, int n=0, T** data=NULL)

*Create a ObjList and initialize it.*

3.5.4.3.1.2  **ObjList** (ObjList<T>& another)

*Copy constructor.*

3.5.4.3.1.1  **ObjList** (T* defaultData, int n=0, T** data=NULL)

*Create a ObjList and initialize it.*

Create a ObjList and initialize it.

**Parameters:**
- **defaultData**  a template to create new object, cannot be NULL.
- **n**  the size of the container, cannot be less than zero (optional, default to zero).
- **data**  the data to initialize the content (optional, default to NULL).

3.5.4.3.1.2  **ObjList** (ObjList<T>& another)

*Copy constructor.*
Copy constructor.

**Parameters:**

another ObjList to copy from.

## 3.5.4.3.2 methods

### Names

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>getSize()</code></td>
<td>Retrieve the size of the ObjList.</td>
<td>139</td>
</tr>
<tr>
<td><code>setSize(int n)</code></td>
<td>Set the size of the ObjList.</td>
<td>139</td>
</tr>
<tr>
<td><code>expand(int delta=1)</code></td>
<td>Expand the size of the ObjList.</td>
<td>139</td>
</tr>
<tr>
<td><code>expand(T* data)</code></td>
<td>Expand the size of the ObjList by one and populate the new entry with the given data.</td>
<td>140</td>
</tr>
<tr>
<td><code>shrink(int delta=1)</code></td>
<td>Shrink the size of the ObjList.</td>
<td>140</td>
</tr>
<tr>
<td><code>operator[](int i)</code></td>
<td>Index operator.</td>
<td>140</td>
</tr>
<tr>
<td><code>getDefaultData()</code></td>
<td>Retrieve the default object.</td>
<td>141</td>
</tr>
<tr>
<td><code>getData(int i)</code></td>
<td>Retrieve the object.</td>
<td>141</td>
</tr>
<tr>
<td><code>setData(int i, T* data)</code></td>
<td>Set the object.</td>
<td>142</td>
</tr>
<tr>
<td><code>swap(int first, int second)</code></td>
<td>Swap the data stored in the container.</td>
<td>142</td>
</tr>
<tr>
<td><code>ObjList&lt;T&gt; &amp;operator=(ObjList&lt;T&gt; &amp;another)</code></td>
<td>Assignment operator.</td>
<td>142</td>
</tr>
<tr>
<td><code>contains(T* element)</code></td>
<td>Check if the ObjList contains a certain element.</td>
<td>143</td>
</tr>
<tr>
<td><code>sendToOstream(ostream &amp;outstream)</code></td>
<td>Send the content of the ObjList to the ostream.</td>
<td>143</td>
</tr>
</tbody>
</table>
3.5.4.3.2.1

int getSize ()

Retrive the size of the ObjList.

Return Value: the size.

3.5.4.3.2.2

void setSize (int n)

Set the size of the ObjList.

Set the size of the ObjList. The content of the ObjList will be expanded or shrunked accordingly.

Parameters: n the new size, cannot be less than zero.

3.5.4.3.2.3

int expand (int delta=1)

Expand the size of the ObjList.

Expand the size of the ObjList. The content of the ObjList will be expanded accordingly.

Return Value: the new size of the ObjList.
Parameters: delta the number of new data to be inserted to the ObjList (optional, default to 1).
3.5.4.3.2.4

```c
int expand (T* data)
```

*Expand the size of the ObjList by one and populate the new entry with the given data.*

Expand the size of the ObjList by one and populate the new entry with the given data.

**Return Value:** the new size of the ObjList.

**Parameters:**
- `data` the data to be inserted to the ObjList.

3.5.4.3.2.5

```c
int shrink (int delta=1)
```

*Shrink the size of the ObjList.*

Shrink the size of the ObjList. The content of the ObjList will be shrunk accordingly.

**Return Value:** the new size of the ObjList.

**Parameters:**
- `delta` the number of new data to be removed from the ObjList.

3.5.4.3.2.6

```c
T* operator[](int i)
```

*Index operator.*

This page has been automatically generated with DOC++

DOC++ is ©1995 by Roland Wunderling
Malte Zöckler
Index operator.

**Return Value:** a pointer to the object stored at the index i (first object is indexed as 0).

**Parameters:** i the index.

### 3.5.4.3.2.7

```c
T* getDefaultData (void)
```

*Retrieve the default object*

Retrieve the default object

**Return Value:** a pointer to the default object.

### 3.5.4.3.2.8

```c
T* getData (int i)
```

*Retrieve the object.*

Retrieve the object.

**Return Value:** a pointer to the object stored at the index i (first object is indexed as 0).

**Parameters:** i the index.
3.5.4.3.2.9

void setData (int i, T* data)

Set the object.

Set the object.

Parameters:

- **i** the index.
- **data** a pointer to the object to be stored at the index i (first object is indexed as 0).

3.5.4.3.2.10

void swap (int first, int second)

Swap the data stored in the container.

Swap the data stored in the container.

Parameters:

- **first** the index of the first object to be swapped.
- **second** the index of the second object to be swapped.

3.5.4.3.2.11

ObjList<T> & operator= (ObjList<T> &another)

Assignment operator.

Assignment operator.
Return Value: a reference to the object itself.
Parameters: another a reference to another ObjList.

3.5.4.3.2.12

bool contains (T* element)

*Check if the Objlist contains a certain element.*

Check if the Objlist contains a certain element.
Parameters: element the element to look for.

3.5.4.3.2.13

void sendToOstream (ostream &outstream)

*Send the content of the ObjList to the ostream.*

Send the content of the ObjList to the ostream.
Parameters: outstream a reference to the ostream.

3.5.4.3.3

destructor

Names
3.5.4.3.3.1 ~ObjList (void) Destructor of the class 144
3.5.4.3.3.1

~ObjList (void)

Destructor of the class

Destructor of the class

3.5.4.4

class Operator

Interface for the operator to preform prefix sum.

Public Members
3.5.4.4.1 method .............................................. 144
3.5.4.4.2 destructor .............................................. 145

Interface for the operator to preform prefix sum. This interface defines the operation for preforming prefix sum.

See Also: ParallelPrefixSummer (→3.2.1, page 51)

3.5.4.4.1

method

Names
3.5.4.4.1.1virtual void 

 operate (CommObject* result, CommObject* first, 
 CommObject* second)
Perform the operation on two CommObject’s. ................. 145
3.5.4.4.1.1

virtual void operate (CommObject* result, CommObject* first, CommObject* second)

*Perform the operation on two CommObject’s.*

Perform the operation on two CommObject’s.

**Parameters:**
- result: a pointer to the result.
- first: a pointer to the first operand.
- second: a pointer to the second operand.

3.5.4.4.2

destructor

**Names**

3.5.4.4.2.1 virtual ~Operator (void) *Destructor of the class*  

3.5.4.4.2.1

virtual ~Operator (void)

*Destructor of the class*
4 graphlib API

Names
4.1 List Ranking ........................................ 146
4.2 Graphs .................................................... 155

4.1 List Ranking

Names
4.1.1 class ListRanker An implementation of the list ranking algorithm ............... 146
4.1.2 class Node A class to implement a node ... 148

4.1.1 class ListRanker

An implementation of the list ranking algorithm

Public Members
4.1.1.1 constants ............................................. 147
4.1.1.2 method ............................................... 148

An implementation of the list ranking algorithm

This page has been automatically generated with DOC++

DOC++ is ©1995 by Roland Wunderling
Malte Zöckler
4.1.1.1 constants

Names

4.1.1.1.1 static const bool RANDOMIZED
A boolean to cause the algorithm to perform a randomized list ranking

4.1.1.1.2 static const bool DETERMINISTIC
A boolean to cause the algorithm to perform a deterministic list ranking

4.1.1.1.1 static const bool RANDOMIZED

A boolean to cause the algorithm to perform a randomized list ranking

4.1.1.1.2 static const bool DETERMINISTIC

A boolean to cause the algorithm to perform a deterministic list ranking
4.1.2 method

Names

4.1.2.1 void rankTheList (ObjList<Node> &nodes,
   bool randomized, Comm* comm)
   To perform a list ranking operation.

Parameters:
   nodes a reference to a list of nodes
   randomized a boolean to indicate if we want to perform a randomized list ranking or not
   comm a point to the select comm

4.1.2 class Node

A class to implement a node
Inheritance

4.1.2 Node

4.2.2.1 EulerNode

Public Members

4.1.2.1 constructors ........................................ 149
4.1.2.2 methods ................................................ 150

A class to implement a node

4.1.2.1 constructors

Names

4.1.2.1.1 Node (void) Default constructor ............... 149
4.1.2.1.2 Node (Node &another) Copy constructor. ............ 150

4.1.2.1.1 Node (void)

Default constructor

Default constructor
4.1.2.1.2

Node (Node &another)

Copy constructor.

Parameters:  
another  a reference to another node

4.1.2.2

methods

Names

4.1.2.2.1 int  getIndex (void)  Retrieve the index of the node. . . 151
4.1.2.2.2 int  getNext (void)  Retrieve the index of the next node. ................. 151
4.1.2.2.3 int  getRank (void)  Retrieve the rank of the node, meaningful only after the list ranking algorithm has been invoked. 152
4.1.2.2.4 int  getTail (void)  Retrieve the index of the last node in the chain, meaningful only after the list ranking algorithm has been invoked. ................. 152
4.1.2.2.5 void  setIndex (int i)  Set the index of the node. ...... 152
4.1.2.2.6 void  setNext (int n)  Set the index of the next node. . . 153
4.1.2.2.7 void  setRank (int r)  Set the rank of the node, should only be used by the list ranking algorithm. ................. 153
4.1.2.2.8 void  setTail (int t)  Set the index of the last node in the chain, should only be used by the list ranking algorithm. ...... 153
4.1.2.2.9 virtual Node*
4 graphlib API

4.1.2.2.10

clone (void* p = NULL)

Clone the node. .................... 154

4.1.2.2.11

virtual Node&

operator= (Node &another)

Copy the content of a node. .... 154

virtual int getSize ()

get the size of the object when "serialised". .................... 154

---

4.1.2.2.1

int getIndex (void)

Retrieve the index of the node.

Return Value: the index

4.1.2.2.2

int getNext (void)

Retrieve the index of the next node.

Return Value: the index of the next node
4.1.2.2.3

```c
int getRank (void)
```

*Retrieve the rank of the node, meaningful only after the list ranking algorithm has been invoked.*

Retrieve the rank of the node, meaningful only after the list ranking algorithm has been invoked.

**Return Value:** the rank of the node

---

### 4.1.2.2.4

```c
int getTail (void)
```

*Retrieve the index of the last node in the chain, meaningful only after the list ranking algorithm has been invoked.*

Retrieve the index of the last node in the chain, meaningful only after the list ranking algorithm has been invoked.

**Return Value:** the index of the last node in the chain

---

### 4.1.2.2.5

```c
void setIndex (int i)
```

*Set the index of the node.*

Set the index of the node.

**Parameters:**

- `i` the new index
4.1.2.2.6

void setNext (int n)

Set the index of the next node.

Set the index of the next node.

Parameters: n the new index for the next node

4.1.2.2.7

void setRank (int r)

Set the rank of the node, should only be used by the list ranking algorithm.

Set the rank of the node, should only be used by the list ranking algorithm.

Parameters: r the new rank of the node

4.1.2.2.8

void setTail (int t)

Set the index of the last node in the chain, should only be used by the list ranking algorithm.

Set the index of the last node in the chain, should only be used by the list ranking algorithm.

Parameters: r the index of the last node in the chain
4.1.2.2.9

virtual Node* clone (void* p = NULL)

Clone the node.

Return Value: a pointer to the newly cloned node
Parameters: p the memory that the object is suppose to clone into.

4.1.2.2.10

virtual Node& operator= (Node &another)

Copy the content of a node.

Copy the content of a node.

Return Value: a reference to the node itself
Parameters: another a reference to another node

4.1.2.2.11

virtual int getSize ()

get the size of the object when "serialized".

get the size of the object when "serialized".

Return Value: the size in byte.
4.2

Graphs

Names
4.2.1 General ........................................ 155
4.2.2 Euler Tour ..................................... 178
4.2.3 Connected Components and Spanning Forest 184
4.2.4 Bipartite Graph Detection ..................... 186

4.2.1

General

Names
4.2.1.1 class Edge  
An implementation of an edge  ..  155
4.2.1.2 class Graph 
An implementation of a graph  ..  163
4.2.1.3 class Vertex  
Implementation of a vertex  .....  169

4.2.1.1

class Edge

An implementation of an edge

Public Members
4.2.1.1.1 constructors .................................. 156
4.2.1.1.2 methods .................................... 158
4.2.1.1.3 destructor .................................. 162

An implementation of an edge
4.2.1.1.1

constructors

Names

4.2.1.1.1.1 \textbf{Edge} (void) \hspace{1cm} \textit{Default constructor} \hspace{1cm} 156

4.2.1.1.1.2 \textbf{Edge} (int i, Vertex &first, Vertex &second, bool b=UNDIRECTED) \hspace{1cm} \textit{Basic constructor.} \hspace{1cm} 156

4.2.1.1.1.3 \textbf{Edge} (int i, int first, int second, bool b=UNDIRECTED) \hspace{1cm} \textit{Simplified constructor.} \hspace{1cm} 157

4.2.1.1.1.4 \textbf{Edge} (Edge &another) \hspace{1cm} \textit{Copy constructor.} \hspace{1cm} 157

4.2.1.1.1.1 \textbf{Edge} (void)

\textit{Default constructor}

Default constructor

\begin{verbatim}
4.2.1.1.1.2
\textbf{Edge} (int i, Vertex &first, Vertex &second, bool b=UNDIRECTED)
\textit{Basic constructor.}

Basic constructor.
\end{verbatim}
Parameters:

- **i**: the index of the edge
- **first**: the first vertex
- **second**: the second vertex
- **b**: not supported yet, should always be false or ignored

### 4.2.1.1.3

**Edge (int i, int first, int second, bool b=UNDIRECTED)**

Simplified constructor.

Parameters:

- **i**: the index of the edge
- **first**: the first vertex
- **second**: the second vertex
- **b**: not supported yet, should always be false or ignored

### 4.2.1.1.4

**Edge (Edge &another)**

Copy constructor.

Parameters:

- **another**: a reference to another edge
### 4.2.1.1.2

**Methods**

**Names**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>int getId()</code></td>
<td>Retrieve the index of the edge.</td>
<td>158</td>
</tr>
<tr>
<td><code>void setId(int i)</code></td>
<td>Set the index of the edge.</td>
<td>159</td>
</tr>
<tr>
<td><code>int getFirstEndPoint()</code></td>
<td>Retrieve the first end point.</td>
<td>159</td>
</tr>
<tr>
<td><code>int getSecondEndPoint()</code></td>
<td>Retrieve the second end point.</td>
<td>160</td>
</tr>
<tr>
<td><code>void setEndPoints(Vertex &amp;first, Vertex &amp;second)</code></td>
<td>Set the end points.</td>
<td>160</td>
</tr>
<tr>
<td><code>void setEndPoints(int first, int second)</code></td>
<td>Set the end points.</td>
<td>160</td>
</tr>
<tr>
<td><code>virtual Edge&amp; operator=(Edge &amp;another)</code></td>
<td>Copy the content of an edge.</td>
<td>161</td>
</tr>
<tr>
<td><code>bool operator==(Edge &amp;another)</code></td>
<td>Compare two edges.</td>
<td>161</td>
</tr>
<tr>
<td><code>virtual Edge* clone(void* p = NULL)</code></td>
<td>Clone the edge.</td>
<td>161</td>
</tr>
<tr>
<td><code>int getSize()</code></td>
<td>Get the size of the object when &quot;serialized&quot;.</td>
<td>162</td>
</tr>
<tr>
<td><code>void sendToOstream(ostream &amp;outstream)</code></td>
<td>Print the edge.</td>
<td>162</td>
</tr>
</tbody>
</table>

---

#### 4.2.1.1.2.1

**int getId() (void)**

Retrieve the index of the edge.
Retrieve the index of the edge.

**Return Value:** the index

### 4.2.1.1.2.2

```c
void setId (int i)
```

*Set the index of the edge.*

Set the index of the edge.

**Parameters:**

```c
i  the new index
```

### 4.2.1.1.2.3

```c
int getFirstEndPoint (void)
```

*Retrieve the first end point.*

Retrieve the first end point.

**Return Value:** the index of the first end point

### 4.2.1.1.2.4

```c
int getSecondEndPoint (void)
```

*Retrieve the second end point.*
Retrieve the second end point.

**Return Value:** the index of the second end point

---

### 4.2.1.1.2.5

```cpp
void setEndPoints (Vertex &first, Vertex &second)
```

*Set the end points.*

Set the end points.

**Parameters:**

- `first` the first end point
- `second` the second end point

---

### 4.2.1.1.2.6

```cpp
void setEndPoints (int first, int second)
```

*Set the end points.*

Set the end points.

**Parameters:**

- `first` the first end point
- `second` the second end point
4.2.1.2.7

virtual Edge& operator= (Edge &another)

Copy the content of an edge.

Copy the content of an edge.

Return Value: a reference to the edge itself
Parameters: another a reference to another edge

4.2.1.2.8

bool operator==(Edge &another)

Compare two edges.

Compare two edges.

Return Value: true if the edges are the same, false otherwise
Parameters: another a reference to another edge to be compared

4.2.1.2.9

virtual Edge* clone (void* p = NULL)

Clone the edge.

Clone the edge.
Return Value: a pointer to the newly cloned edge
Parameters: p the memory that the object is suppose to clone into.

4.2.1.2.10

virtual int getSize ()

get the size of the object when "serialized".

get the size of the object when "serialized".
Return Value: the size in byte.

4.2.1.2.11

virtual void sendToOstream (ostream &outstream)

Print the edge.

Print the edge.
Parameters: outstream the stream that the edge is to be printed on

4.2.1.3

destructor

Names
4.2.1.3.1virtual ~Edge (void) Destructor of the class .......... 163
4.2.1.3.1

virtual ~Edge (void)

Destructor of the class

4.2.1.2

class Graph

An implementation of a graph

Public Members
4.2.1.2.1 constructors ........................................ 163
4.2.1.2.2 methods .............................................. 164

An implementation of a graph

4.2.1.2.1 constructors

Names
4.2.1.2.1.1 Graph (void)  Default constructor ............... 164
4.2.1.2.1.2 Graph (ObjList<Vertex> &v, ObjList<Edge> &e)  Basic constructor. ............... 164
4.2.1.2.1.1

Graph (void)

Default constructor

4.2.1.2.1.2

Graph (ObjList<Vertex> &v, ObjList<Edge> &e)

Basic constructor.

Parameters:
  v  the vertices
  e  the edges

4.2.1.2.2

methods

Names

4.2.1.2.2.1 ObjList<Vertex> *
  getVertices (void) Retrieve the vertices.  ..........  165

4.2.1.2.2.2 ObjList<Edge> *
  getEdges (void) Retrieve the edges.  ...............  165

4.2.1.2.2.3 Vertex*
  getVertex (int index)
  Retrieve a vertex.  ...............  166

4.2.1.2.2.4 Edge*
  getEdge (int index)
  Retrieve an edge.  ...............  166

4.2.1.2.2.5 int
  getNumberOfVertices (void)
Count the number of vertices. . . 167

4.2.1.2.2.6 int getNumberOfEdges (void)

Count the number of edges. . . 167

4.2.1.2.2.7 void addVertex (Vertex* v)

Add a new vertex to the graph. . 167

4.2.1.2.2.8 void addEdge (Edge* e)

Add a new edge to the graph. . . 168

4.2.1.2.2.9 bool contains (Vertex* v)

Check if the graph contains a certain vertex. ..................... 168

4.2.1.2.2.10 bool contains (Edge* e)

Check if the graph contains a certain edge. ......................... 168

4.2.1.2.2.1

ObjList<Vertex>* getVertices (void)

Retrieve the vertices.

Retrieve the vertices.

Return Value: a pointer to the list of vertices

4.2.1.2.2.2

ObjList<Edge>* getEdges (void)

Retrieve the edges.

Retrieve the edges.
Return Value: a pointer to the list of edges

### 4.2.1.2.2.3

<table>
<thead>
<tr>
<th>Vertex* getVertex (int index)</th>
</tr>
</thead>
</table>

Retrieves a vertex.

Parameters: index the index of the vertex

Return Value: a pointer to the vertex

### 4.2.1.2.2.4

<table>
<thead>
<tr>
<th>Edge* getEdge (int index)</th>
</tr>
</thead>
</table>

Retrieves an edge.

Parameters: index the index of the edge

Return Value: a pointer to the edge

### 4.2.1.2.2.5

<table>
<thead>
<tr>
<th>int getNumberOfVertices (void)</th>
</tr>
</thead>
</table>

Counts the number of vertices.
Count the number of vertices.

Return Value: the number of vertices

4.2.1.2.2.6

```c
int getNumberOfEdges (void)
```

Count the number of edges.

Return Value: the number of edges

4.2.1.2.2.7

```c
void addVertex (Vertex* v)
```

Add a new vertex to the graph.

Add a new vertex to the graph.

Parameters: v the new vertex
4.2.1.2.2.8

void addEdge (Edge* e)

Add a new edge to the graph.

Parameters:   

   e    the new edge

4.2.1.2.2.9

bool contains (Vertex* v)

Check if the graph contains a certain vertex.

Parameters:   

   v    the vertex

Return Value:   

   true   if the graph contains v, false otherwise

4.2.1.2.2.10

bool contains (Edge* e)

Check if the graph contains a certain edge.

Parameters:   

   e    the edge

Return Value:   

   true   if the graph contains e, false otherwise
4.2.1.3

class Vertex

Implementation of a vertex

Public Members
4.2.1.3.1 constants .................................. 169
4.2.1.3.2 constructors ............................... 171
4.2.1.3.3 methods ................................... 172
4.2.1.3.4 destructor .................................. 178

Implementation of a vertex

4.2.1.3.1

constants

Names
4.2.1.3.1.1 static int const
BIPARTITE_GROUP_A
An integer to indicate that the vertex in group A of a bipartite graph .......................... 170

4.2.1.3.1.2 static int const
BIPARTITE_GROUP_B
An integer to indicate that the vertex in group B of a bipartite graph .......................... 170

4.2.1.3.1.3 static int const
BIPARTITE_GROUP_UNKNOWN
An integer to indicate that we do not know which group the vertex belongs to .................. 170
4.2.1.3.1.1

static int const BIPARTITE_GROUP_A

An integer to indicate that the vertex in group A of a bipartite graph

4.2.1.3.1.2

static int const BIPARTITE_GROUP_B

An integer to indicate that the vertex in group B of a bipartite graph

4.2.1.3.1.3

static int const BIPARTITE_GROUP_UNKNOWN

An integer to indicate that we do not know which group the vertex belongs to
### 4.2.1.3.2

**constructors**

#### Names

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>4.2.1.3.2.1</td>
<td><strong>Vertex</strong> (void)</td>
<td>Default constructor</td>
<td>. . . . . . . . . . . 171</td>
<td></td>
</tr>
<tr>
<td>4.2.1.3.2.2</td>
<td><strong>Vertex</strong> (int i)</td>
<td>Basic constructor</td>
<td>. . . . . . . . . . . 171</td>
<td></td>
</tr>
<tr>
<td>4.2.1.3.2.3</td>
<td><strong>Vertex</strong> (Vertex &amp;another)</td>
<td>Copy constructor</td>
<td>. . . . . . . . . . . 172</td>
<td></td>
</tr>
</tbody>
</table>

#### 4.2.1.3.2.1

**Vertex** (void)

*Default constructor*

Default constructor

#### 4.2.1.3.2.2

**Vertex** (int i)

*Basic constructor*

Basic constructor

**Parameters:**

i the index of the vertex
### 4.2.1.3.2.3

**Vertex** (Vertex &another)

*Copy constructor*

Copy constructor

**Parameters:**

- `another` a reference to another vertex

### 4.2.1.3.3

**methods**

#### Names

| 4.2.1.3.3.1 | `getId` (void) | Retrieve the index of the vertex. |
| 4.2.1.3.3.2 | `getComponentId` (void) |
| 4.2.1.3.3.3 | `getBipartiteGroup` (void) |
| 4.2.1.3.3.4 | `setId` (int i) |
| 4.2.1.3.3.5 | `setComponentId` (int c) |
| 4.2.1.3.3.6 | `setBipartiteGroup` (int b) |

-four virtual Vertex&
4.2.1.3.3.8 virtual Vertex&
operator=(int another)
Setup the vertex from an integer representing the ID of the vertex.

4.2.1.3.3.9 bool
operator==(Vertex &another)
Compare two vertices.

4.2.1.3.3.10 virtual Vertex*
clone (void* p = NULL)
Clone the vertex.

4.2.1.3.3.11 virtual int getSize()
get the size of the object when "serialized".

4.2.1.3.3.12 virtual void
sendToOstream (ostream &outstream)
Print the vertex.

4.2.1.3.3.1
int getId (void)
Retrieve the index of the vertex.

Return Value: the index

4.2.1.3.3.2
int getComponentId (void)
Retrieve the component ID, meaningful only after the connected component algorithm has been invoked.
Retrieve the component ID, meaningful only after the connected component algorithm has been invoked.

**Return Value:** the component ID

---

**4.2.1.3.3.3**

```c
int getBipartiteGroup (void)
```

*Retrieve the bipartite group, meaningful only after the bipartite graph detection algorithm has been invoked AND the graph is really a bipartite graph.*

---

Retrieve the bipartite group, meaningful only after the bipartite graph detection algorithm has been invoked AND the graph is really a bipartite graph.

**Return Value:** the bipartite group

---

**4.2.1.3.3.4**

```c
void setId (int i)
```

*Set the index of the vertex.*

---

Set the index of the vertex.

**Parameters:**

- `i` the new index
### 4.2.1.3.3.5

```cpp
template

void setComponentId (int c)
```

*Set the component ID, should only be used by the connected components algorithm.*

Set the component ID, should only be used by the connected components algorithm.

**Parameters:**

- `c`  
  the component ID

### 4.2.1.3.3.6

```cpp
template

void setBipartiteGroup (int b)
```

*Set the bipartite group, should only be used by the biparatite graph detection algorithm.*

Set the bipartite group, should only be used by the biparatite graph detection algorithm.

**Parameters:**

- `b`  
  the bipartite group

### 4.2.1.3.3.7

```cpp
virtual Vertex& operator= (Vertex &another)
```

*Copy the content of a vertex.*

Copy the content of a vertex.
Return Value: a reference to the vertex itself
Parameters: another a reference to another vertex

### 4.2.1.3.3.8

```cpp
virtual Vertex& operator=(int another)
```

Setup the vertex from an integer representing the ID of the vertex.

Setup the vertex from an integer representing the ID of the vertex.

Return Value: a reference to the vertex itself
Parameters: i the index of the vertex

### 4.2.1.3.3.9

```cpp
bool operator==(Vertex& another)
```

Compare two vertices.

Compare two vertices.

Return Value: true if the vertices are the same, false otherwise
Parameters: another a reference to another vertex to be compared
4.2.1.3.3.10

virtual Vertex* clone (void* p = NULL)

Clone the vertex.

Return Value: a pointer to the newly cloned vertex
Parameters: p the memory that the object is suppose to clone into.

4.2.1.3.3.11

virtual int getSize ()

get the size of the object when "serialized".

Return Value: the size in byte.

4.2.1.3.3.12

virtual void sendToOstream (ostream &outstream)

Print the vertex.

Parameters: outstream the stream that the vertex is to be printed on
4.2.1.3.4

destructor

Names
4.2.1.3.4.1 virtual ~\texttt{Vertex} (void) Destructor of the class 178

4.2.2

Euler Tour

Names
4.2.2.1 class \texttt{EulerNode} : public Node A node that is used in the Euler Tour algorithm 178
4.2.2.2 class \texttt{EulerTourer} An implementation of the Euler Tour algorithm 182

4.2.2.1
class \texttt{EulerNode} : public Node

A node that is used in the Euler Tour algorithm
Inheritance

Public Members

4.2.2.1.1 constructors ........................................ 179
4.2.2.1.2 methods ............................................. 180

A node that is used in the Euler Tour algorithm

### 4.2.2.1.1

#### constructors

Names

4.2.2.1.1.1 **EulerNode** (void) *Default constructor* .......... 179
4.2.2.1.1.2 **EulerNode** (EulerNode &another)

*Copy constructor.* .......... 180

#### 4.2.2.1.1.1

**EulerNode** (void)

*Default constructor*

Default constructor
4.2.2.1.2

**EulerNode** (EulerNode &another)

Copy constructor.

Copy constructor.

**Parameters:**

another a reference to another euler node

### 4.2.2.1.2 methods

#### Names

4.2.2.1.2.1 virtual Node* `clone` (void* p = NULL)

*Clone the euler node.*  
.........  181

4.2.2.1.2.2 virtual Node& `operator=` (Node &another)

*Copy the content of an euler node.*  
......................  181

4.2.2.1.2.3 virtual int `getSize` ()

*get the size of the object when "serialized".*  
......................  181

4.2.2.1.2.4 int `getVertexIndex` (int i)

*Retrieve the index of the given vertex.*  
......................  182

4.2.2.1.2.5 void `setVertexIndex` (int i, int v)

*Set the index of the given vertex.*  
  182
4.2.2.1.2.1

virtual Node* clone (void* p = NULL)

Clone the euler node.

Return Value: a pointer to the newly cloned euler node
Parameters: p the memory that the object is suppose to clone into.

4.2.2.1.2.2

virtual Node& operator= (Node &another)

Copy the contenet of an euler node.

Return Value: a reference to the euler node itself
Parameters: another a reference to another euler node

4.2.2.1.2.3

virtual int getSize ()

get the size of the object when "serialized".

Return Value: the size in byte.
4.2.1.2.4

int getVertexIndex (int i)

Retrieve the index of the given vertex.

Return Value: the index of the given vertex
Parameters: i an integer to represent the end point, 0 = first end point, 1 = the second end point

4.2.1.2.5

void setVertexIndex (int i, int v)

Set the index of the given vertex.

Parameters: i an integer to represent the end point, 0 = first end point, 1 = the second end point
v the new index

4.2.2

class EulerTourer

An implementation of the Euler Tour algorithm
An implementation of the Euler Tour algorithm

4.2.2.2.1 method

Names

4.2.2.2.1.1 void getEulerTour (ObjList<Vertex> &r, ObjList<Vertex> &v, ObjList<Edge> &e, ObjList<EulerNode> &eulerNodes, Comm* comm)

Find the Euler Tour of a forest.

Find the Euler Tour of a forest.

Parameters:

- **r**: a reference to a list of vertices that represent the roots of the forest
- **v**: a reference to a list of vertices
- **e**: a reference to a list of edges
- **eulerNodes**: a reference to a list of eulernodes that is used to hold the result of the algorithm@comm a pointer to the selected comm

Find the Euler Tour of a forest.
4.2.3

Connected Components and Spanning Forest

Names
4.2.3.1 class ConnectedComponents

An implementation of the connected components and spanning forest algorithms

Public Members
4.2.3.1.1 methods

An implementation of the connected components and spanning forest algorithms

4.2.3.1.1

methods

Names
4.2.3.1.1.1 void findConnectedComponents (Graph &g, Comm* comm)
Find the connected components of a graph.

4.2.3.1.1.1

```cpp
void findConnectedComponents (Graph &g, Comm* comm)
```

Find the connected components of a graph.

Parameters:

- `g` a reference to the graph
- `comm` a pointer to the selected comm

4.2.3.1.1.2

```cpp
void findSpanningForest (Graph &g, ObjList<Vertex> &spanningRoots,
                         ObjList<Edge> &spanningTreeEdges,
                         Comm* comm)
```

Find the spanning forests of a graph.

Find the spanning forests of a graph.
Parameters:

- **g**: a reference to the graph
- **spanningRoots**: a reference to a list of vertices that represents the roots of the resulting spanning forest
- **spanningTreeEdge**: a reference to a list of edges that represent the resulting spanning forest
- **comm**: a pointer to the selected comm

### 4.2.4

**Bipartite Graph Detection**

**Names**

**4.2.4.1 class** BipartiteDetector

*An implementation of the bipartite graph detection algorithm*

**Public Members**

**4.2.4.1.1 method** ........................................ 186

*An implementation of the bipartite graph detection algorithm*

**4.2.4.1.1**

**method**
Names

4.2.4.1.1 bool isBipartiteGraph (Graph &g, Comm* comm)

Determine if a graph is a bipartite graph.

Return Value:

\textbf{true} if the graph is a bipartite graph, false otherwise

Parameters:

g a reference to the graph
comm a pointer to the selected comm
Distribution and Installation

The library is distributed in source code. It can be downloaded from the following URLs:

- http://www.scs.carleton.ca/~cgm/src/cgmlib.src.tar.gz
- http://www.scs.carleton.ca/~cgm/src/cgmlib.src.zip

To install the library, unzip the source code into a library, in the top level, there will be a file named `Makefile.common`, edit the file to make the following macros points to the correct tools/directory/include files/libraries:

- **SHELL** - The shell to run make, usually `/bin/sh`
- **BINDIR** - The directory to hold the generic tools, e.g. `/usr/bin`
- **MPIHOME** - The directory where MPI is installed, e.g. `/usr/local/apps/mpich-1.2.2`
- **MPICFLAGS** - The flags to use for compiling MPI source code, e.g. `-I$(MPIHOME)/include`
- **MPILDFLAGS** - The flags to use for linking MPI object code, e.g. `-L$(MPIHOME)/lib -lpmpich++ -lmpich`
- **CC** - the program used to compile c program, e.g. ‘which gcc’ (using gcc)
- **CLINKER** - the program used to link c object code, e.g. ‘which gcc’ (using gcc)
- **CCC** - the program used to compile c++ program, e.g. ‘which g++’ (using g++)
- **CCLINKER** - the program used to link c++ object code, e.g. ‘which g++’ (using g++)
- **AR** - the program used to create library, e.g. ‘which ar’ (using ar)
- **RANLIB** - the program used to create index for libraries, e.g. ‘which ranlib’ (using ranlib)

You may also need to adjust the Makefile for the install directory if you want to install the library in a common location for other to use:

- **INSTALL** - the location to install the library, e.g. `/usr/local/cgmlib`. Note you must have write permission to the install directory.
After you’ve adjusted Makefile and Makefile.common to fit your system configuration, you can build the library by issuing the following commands in the top level directory:

- make - to build the library
- make apps - to make the applications
- make clean - to clean the library
- make install - to install the library into a predefined location
- make remove - to uninstall the library from the predefined location (will prompt for file removal)
- make removeNoPrompt - to uninstall the library from the predefined location (will NOT prompt for file removal)
This library is distributed under the GNU's LGPL:

GNU LESSER GENERAL PUBLIC LICENSE
Version 2.1, February 1999

Copyright (C) 1991, 1999 Free Software Foundation, Inc.
59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.

[This is the first released version of the Lesser GPL. It also counts
as the successor of the GNU Library Public License, version 2, hence
the version number 2.1.]

Preamble

The licenses for most software are designed to take away your
freedom to share and change it. By contrast, the GNU General Public
Licenses are intended to guarantee your freedom to share and change
free software--to make sure the software is free for all its users.

This license, the Lesser General Public License, applies to some
specially designated software packages--typically libraries--of the
Free Software Foundation and other authors who decide to use it. You
can use it too, but we suggest you first think carefully about whether
this license or the ordinary General Public License is the better
strategy to use in any particular case, based on the explanations below.

When we speak of free software, we are referring to freedom of use,
not price. Our General Public Licenses are designed to make sure that
you have the freedom to distribute copies of free software (and charge
for this service if you wish); that you receive source code or can get
it if you want it; that you can change the software and use pieces of
it in new free programs; and that you are informed that you can do
these things.

To protect your rights, we need to make restrictions that forbid
distributors to deny you these rights or to ask you to surrender these
rights. These restrictions translate to certain responsibilities for
you if you distribute copies of the library or if you modify it.

For example, if you distribute copies of the library, whether gratis or for a fee, you must give the recipients all the rights that we gave you. You must make sure that they, too, receive or can get the source code. If you link other code with the library, you must provide complete object files to the recipients, so that they can relink them with the library after making changes to the library and recompiling it. And you must show them these terms so they know their rights.

We protect your rights with a two-step method: (1) we copyright the library, and (2) we offer you this license, which gives you legal permission to copy, distribute and/or modify the library.

To protect each distributor, we want to make it very clear that there is no warranty for the free library. Also, if the library is modified by someone else and passed on, the recipients should know that what they have is not the original version, so that the original author's reputation will not be affected by problems that might be introduced by others.

Finally, software patents pose a constant threat to the existence of any free program. We wish to make sure that a company cannot effectively restrict the users of a free program by obtaining a restrictive license from a patent holder. Therefore, we insist that any patent license obtained for a version of the library must be consistent with the full freedom of use specified in this license.

Most GNU software, including some libraries, is covered by the ordinary GNU General Public License. This license, the GNU Lesser General Public License, applies to certain designated libraries, and is quite different from the ordinary General Public License. We use this license for certain libraries in order to permit linking those libraries into non-free programs.

When a program is linked with a library, whether statically or using a shared library, the combination of the two is legally speaking a combined work, a derivative of the original library. The ordinary General Public License therefore permits such linking only if the entire combination fits its criteria of freedom. The Lesser General Public License permits more lax criteria for linking other code with the library.

We call this license the "Lesser" General Public License because it does Less to protect the user's freedom than the ordinary General Public License. It also provides other free software developers Less
of an advantage over competing non-free programs. These disadvantages are the reason we use the ordinary General Public License for many libraries. However, the Lesser license provides advantages in certain special circumstances.

For example, on rare occasions, there may be a special need to encourage the widest possible use of a certain library, so that it becomes a de-facto standard. To achieve this, non-free programs must be allowed to use the library. A more frequent case is that a free library does the same job as widely used non-free libraries. In this case, there is little to gain by limiting the free library to free software only, so we use the Lesser General Public License.

In other cases, permission to use a particular library in non-free programs enables a greater number of people to use a large body of free software. For example, permission to use the GNU C Library in non-free programs enables many more people to use the whole GNU operating system, as well as its variant, the GNU/Linux operating system.

Although the Lesser General Public License is Less protective of the users' freedom, it does ensure that the user of a program that is linked with the Library has the freedom and the wherewithal to run that program using a modified version of the Library.

The precise terms and conditions for copying, distribution and modification follow. Pay close attention to the difference between a "work based on the library" and a "work that uses the library". The former contains code derived from the library, whereas the latter must be combined with the library in order to run.

GNU LESSER GENERAL PUBLIC LICENSE
TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License Agreement applies to any software library or other program which contains a notice placed by the copyright holder or other authorized party saying it may be distributed under the terms of this Lesser General Public License (also called "this License"). Each licensee is addressed as "you".

A "library" means a collection of software functions and/or data prepared so as to be conveniently linked with application programs (which use some of those functions and data) to form executables.

The "Library", below, refers to any such software library or work which has been distributed under these terms. A "work based on the
Library" means either the Library or any derivative work under
copyright law: that is to say, a work containing the Library or a
portion of it, either verbatim or with modifications and/or translated
straightforwardly into another language. (Hereinafter, translation is
included without limitation in the term "modification".)

"Source code" for a work means the preferred form of the work for
making modifications to it. For a library, complete source code means
all the source code for all modules it contains, plus any associated
interface definition files, plus the scripts used to control compilation
and installation of the library.

Activities other than copying, distribution and modification are not
covered by this License; they are outside its scope. The act of
running a program using the Library is not restricted, and output from
such a program is covered only if its contents constitute a work based
on the Library (independent of the use of the Library in a tool for
writing it). Whether that is true depends on what the Library does
and what the program that uses the Library does.

1. You may copy and distribute verbatim copies of the Library's
complete source code as you receive it, in any medium, provided that
you conspicuously and appropriately publish on each copy an
appropriate copyright notice and disclaimer of warranty; keep intact
all the notices that refer to this License and to the absence of any
warranty; and distribute a copy of this License along with the
Library.

You may charge a fee for the physical act of transferring a copy,
and you may at your option offer warranty protection in exchange for a
fee.

2. You may modify your copy or copies of the Library or any portion
of it, thus forming a work based on the Library, and copy and
distribute such modifications or work under the terms of Section 1
above, provided that you also meet all of these conditions:

   a) The modified work must itself be a software library.

   b) You must cause the files modified to carry prominent notices
      stating that you changed the files and the date of any change.

   c) You must cause the whole of the work to be licensed at no
      charge to all third parties under the terms of this License.

   d) If a facility in the modified Library refers to a function or a
table of data to be supplied by an application program that uses the facility, other than as an argument passed when the facility is invoked, then you must make a good faith effort to ensure that, in the event an application does not supply such function or table, the facility still operates, and performs whatever part of its purpose remains meaningful.

(For example, a function in a library to compute square roots has a purpose that is entirely well-defined independent of the application. Therefore, Subsection 2d requires that any application-supplied function or table used by this function must be optional: if the application does not supply it, the square root function must still compute square roots.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Library, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Library, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Library.

In addition, mere aggregation of another work not based on the Library with the Library (or with a work based on the Library) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may opt to apply the terms of the ordinary GNU General Public License instead of this License to a given copy of the Library. To do this, you must alter all the notices that refer to this License, so that they refer to the ordinary GNU General Public License, version 2, instead of to this License. (If a newer version than version 2 of the ordinary GNU General Public License has appeared, then you can specify that version instead if you wish.) Do not make any other change in these notices.

Once this change is made in a given copy, it is irreversible for that copy, so the ordinary GNU General Public License applies to all
subsequent copies and derivative works made from that copy.

This option is useful when you wish to copy part of the code of the Library into a program that is not a library.

4. You may copy and distribute the Library (or a portion or derivative of it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange.

If distribution of object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place satisfies the requirement to distribute the source code, even though third parties are not compelled to copy the source along with the object code.

5. A program that contains no derivative of any portion of the Library, but is designed to work with the Library by being compiled or linked with it, is called a "work that uses the Library". Such a work, in isolation, is not a derivative work of the Library, and therefore falls outside the scope of this License.

However, linking a "work that uses the Library" with the Library creates an executable that is a derivative of the Library (because it contains portions of the Library), rather than a "work that uses the library". The executable is therefore covered by this License. Section 6 states terms for distribution of such executables.

When a "work that uses the Library" uses material from a header file that is part of the Library, the object code for the work may be a derivative work of the Library even though the source code is not. Whether this is true is especially significant if the work can be linked without the Library, or if the work is itself a library. The threshold for this to be true is not precisely defined by law.

If such an object file uses only numerical parameters, data structure layouts and accessors, and small macros and small inline functions (ten lines or less in length), then the use of the object file is unrestricted, regardless of whether it is legally a derivative work. (Executables containing this object code plus portions of the Library will still fall under Section 6.)

Otherwise, if the work is a derivative of the Library, you may distribute the object code for the work under the terms of Section 6.
Any executables containing that work also fall under Section 6, whether or not they are linked directly with the Library itself.

6. As an exception to the Sections above, you may also combine or link a "work that uses the Library" with the Library to produce a work containing portions of the Library, and distribute that work under terms of your choice, provided that the terms permit modification of the work for the customer's own use and reverse engineering for debugging such modifications.

You must give prominent notice with each copy of the work that the Library is used in it and that the Library and its use are covered by this License. You must supply a copy of this License. If the work during execution displays copyright notices, you must include the copyright notice for the Library among them, as well as a reference directing the user to the copy of this License. Also, you must do one of these things:

a) Accompany the work with the complete corresponding machine-readable source code for the Library including whatever changes were used in the work (which must be distributed under Sections 1 and 2 above); and, if the work is an executable linked with the Library, with the complete machine-readable "work that uses the Library", as object code and/or source code, so that the user can modify the Library and then relink to produce a modified executable containing the modified Library. (It is understood that the user who changes the contents of definitions files in the Library will not necessarily be able to recompile the application to use the modified definitions.)

b) Use a suitable shared library mechanism for linking with the Library. A suitable mechanism is one that (1) uses at run time a copy of the library already present on the user's computer system, rather than copying library functions into the executable, and (2) will operate properly with a modified version of the library, if the user installs one, as long as the modified version is interface-compatible with the version that the work was made with.

c) Accompany the work with a written offer, valid for at least three years, to give the same user the materials specified in Subsection 6a, above, for a charge no more than the cost of performing this distribution.

d) If distribution of the work is made by offering access to copy from a designated place, offer equivalent access to copy the above specified materials from the same place.
e) Verify that the user has already received a copy of these materials or that you have already sent this user a copy.

For an executable, the required form of the "work that uses the Library" must include any data and utility programs needed for reproducing the executable from it. However, as a special exception, the materials to be distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

It may happen that this requirement contradicts the license restrictions of other proprietary libraries that do not normally accompany the operating system. Such a contradiction means you cannot use both them and the Library together in an executable that you distribute.

7. You may place library facilities that are a work based on the Library side-by-side in a single library together with other library facilities not covered by this License, and distribute such a combined library, provided that the separate distribution of the work based on the Library and of the other library facilities is otherwise permitted, and provided that you do these two things:

a) Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities. This must be distributed under the terms of the Sections above.

b) Give prominent notice with the combined library of the fact that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

8. You may not copy, modify, sublicense, link with, or distribute the Library except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, link with, or distribute the Library is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

9. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Library or its derivative works. These actions are
prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Library (or any work based on the Library), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Library or works based on it.

10. Each time you redistribute the Library (or any work based on the Library), the recipient automatically receives a license from the original licensor to copy, distribute, link with or modify the Library subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties with this License.

11. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Library at all. For example, if a patent license would not permit royalty-free redistribution of the Library by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Library.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply, and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

12. If the distribution and/or use of the Library is restricted in
certain countries either by patents or by copyrighted interfaces, the 
original copyright holder who places the Library under this License may add 
an explicit geographical distribution limitation excluding those countries, 
so that distribution is permitted only in or among countries not thus 
excluded. In such case, this License incorporates the limitation as if 
written in the body of this License.

13. The Free Software Foundation may publish revised and/or new 
versions of the Lesser General Public License from time to time. 
Such new versions will be similar in spirit to the present version, 
but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Library 
specifies a version number of this License which applies to it and 
"any later version", you have the option of following the terms and 
conditions either of that version or of any later version published by 
the Free Software Foundation. If the Library does not specify a 
license version number, you may choose any version ever published by 
the Free Software Foundation.

14. If you wish to incorporate parts of the Library into other free 
programs whose distribution conditions are incompatible with these, 
write to the author to ask for permission. For software which is 
copyrighted by the Free Software Foundation, write to the Free 
Software Foundation; we sometimes make exceptions for this. Our 
decision will be guided by the two goals of preserving the free status 
of all derivatives of our free software and of promoting the sharing 
and reuse of software generally.

NO WARRANTY

15. BECAUSE THE LIBRARY IS LICENSED FREE OF CHARGE, THERE IS NO 
WARRANTY FOR THE LIBRARY, TO THE EXTENT PERMITTED BY APPLICABLE LAW. 
EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR 
OTHER PARTIES PROVIDE THE LIBRARY "AS IS" WITHOUT WARRANTY OF ANY 
KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE 
IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR 
PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE 
LIBRARY IS WITH YOU. SHOULD THE LIBRARY PROVE DEFECTIVE, YOU ASSUME 
THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN 
WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY 
AND/OR REDISTRIBUTE THE LIBRARY AS PERMITTED ABOVE, BE LIABLE TO YOU 
FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR 
CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE
LIBRARY (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING
RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A
FAILURE OF THE LIBRARY TO OPERATE WITH ANY OTHER SOFTWARE), EVEN IF
SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH
DAMAGES.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Libraries

If you develop a new library, and you want it to be of the greatest
possible use to the public, we recommend making it free software that
everyone can redistribute and change. You can do so by permitting
redistribution under these terms (or, alternatively, under the terms of the
ordinary General Public License).

To apply these terms, attach the following notices to the library. It is
safest to attach them to the start of each source file to most effectively
convey the exclusion of warranty; and each file should have at least the
"copyright" line and a pointer to where the full notice is found.

<one line to give the library's name and a brief idea of what it does.>
Copyright (C) <year> <name of author>

This library is free software; you can redistribute it and/or
modify it under the terms of the GNU Lesser General Public
License as published by the Free Software Foundation; either
version 2.1 of the License, or (at your option) any later version.

This library is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public
License along with this library; if not, write to the Free Software
Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

Also add information on how to contact you by electronic and paper mail.

You should also get your employer (if you work as a programmer) or your
school, if any, to sign a "copyright disclaimer" for the library, if
necessary. Here is a sample; alter the names:

Yoyodyne, Inc., hereby disclaims all copyright interest in the
library 'Frob' (a library for tweaking knobs) written by James Random Hacker.
<signature of Ty Coon>, 1 April 1990
Ty Coon, President of Vice

That's all there is to it!
Bug Report

Send bug report and/or suggestions to cgm@scs.carleton.ca.

Please include in your bug report the following information:

- Your contact (name, email address, etc)
- The environment (type of machine, number of processors, RAM, disk storage, etc)
- Description of the problem (please include error message if available) and how to reproduce the problem.
- Any other additional information that you think may help us to fix the problem.

Please note that due to the wide variation of the applications that this library can apply to, we can only trace/debug the problem that is caused directly by the use of the library. We have no resource to help the user to debug the problem caused by misuse of the library. However, we welcome any suggestion to be incorporated into future release, although we cannot make any commitment.
This library is developed by Albert Chan, under the supervision of Dr. Frank Dehne, as part of his Ph. D thesis. The research is partially funded by the following organizations:

- Natural Sciences and Engineering Research Council of Canada
- Nortel Networks

Ryan Taylor ported the library to run on the Sunfire machine.
In addition, we must acknowledge the following parties:

- Argonne National Laboratory for providing the MPICH implementation
- Roland Wunderling and Malte Zekler for providing DOC++ as the documentation tool for this library
Class Graph

4.1.2 Node .................................................. 148

4.2.2.1 EulerNode .......................................... 178

4.2.1.1 Edge .................................................. 155

4.2.1.2 Graph .................................................. 163

4.2.1.3 Vertex .................................................. 169

4.2.2.2 EulerTourer ........................................... 182

4.2.3.1 ConnectedComponents ................................. 184

4.2.4.1 BipartiteDetector ..................................... 186