GBDI ArboretumDeveloper's Guide[1]

Fabio Jun Takada Chino
Marcos Rodrigues Vieira
GBDI-ICMC-USP

May 9, 2012

[1]This work has been supported by FAPESP (Sao Paulo State Foundation for Research Support)

Abstract

The abstract comes here

Contents

Chapter 1

Introduction to GBDI Arboretum

1.1 What is GBDI Arboretum

1.2 Motivation and Objectives

1.3 A Brief Introduction to Metric Access Methods

The main practice used to accelerate the data retrieval in database management systems is indexing the data through an access method tailored to the domain of the data. The data domains stored in traditional databases - such as numbers and small character strings - have the total ordering property. This property leaded to the development of the current Database Management Systems (DBMS). However, when the data do not have this property, the traditional access methods cannot be used. Data embedded in metric space domains are examples of information that cannot be directly sorted. Metric Access Methods (MAM) have been developed for such data domains, e.g., the M-tree and Slim-tree.

In metric domains the only information available is the objects and the distances between them. The distance function is usually provided by a domain expert, who gathers the particularities of the target domain in order to compare objects.

Formally, given three objects, o_1, o_2 and o_3 pertaining to the domain of objects $\mathfrak{D}$, a distance function $d()$ is said to be metric if it satisfies the following three properties:

1. **symmetry**: $d(o_1, o_2) = d(o_2, o_1)$
2. **non-negativity**: $0 < d(o_1, o_2) < \infty$ if $o_1 \neq o_2$ and $d(o_1, o_1) = 0$
3. **triangle inequality**: $d(o_1, o_2) \leq d(o_1, o_3) + d(o_3, o_2)$

A metric distance function is the foundation to build MAMs, which were developed since the pioneering work of Burkhard and Keller [Burkhard and Keller, 1973]. They are now achieving an efficiency level good enough to be used in practical situations, as is the case of the M-tree [Ciaccia et al., 1997], the Slim-tree [Traina et al., 2000] and the Omni-family members [Santos et al., 2001].

Data in metric domains are retrieved using similarity queries. The two most frequently used similarity queries are defined following:

- **k-Nearest Neighbor Query ($k - NNQ$)**: $kNNQ(k, o_q)$, which asks for the k objects that are the closest to a given query object center o_q, with $o_q \in \mathfrak{D}$ (the object domain). For example, in an image database domain, a typical query could be: "find the 5 nearest images to image1".
- **Range Query (RQ)**: $RQ(o_q, r_q)$, which searches for all the objects whose distance to the query object center o_q is less or equal to the query radius r_q . Using the example previously given, a query could be: "find all the images that are within 10 units of distance from image1".

Calculating distances between complex objects are usually expensive operations. Hence, minimizing these calculations is one of the most important aspects to obtain an efficient answer for the queries. MAMs minimize the number of distance calculations taking advantage of the triangular inequality property of metric distance functions, which allows to prune parts of the tree during the answering process. Metric access methods built on the image features have been successfully used to index images and are suited to answer similarity queries.

1.4 Document Organization

The remaining of this document is structured as follows. In the next chapter, we first give a brief overview of the archtecture of GBDI Arboretum. The user, structure and storage layers are introduce in Chapter 3, **??** and 5, respectively. Chapter 8 explain how to write and to became familiar with the source code of the GBDI Arboretum. Chapter 7 shows how to add a new Metric Access Method in the GBDI Arboretum. Chapter 9 presents the conclusion of this document, and Chapters A and C presents the Utilities and the License of GBDI Arboretum.

Chapter 2

Architecture of GBDI Arboretum

2.1 Overview

2.2 System Requirements

All design decisions and the basic structure of the GBDI Arboretum was based on the system requirements. These requirements are:

1. the library must be portable among various compilers and systems;
2. the library must provide an easy to use framework for MAM developers;
3. the library must be implemented using the object oriented approach;
4. multiple instances of MAMs may exists in the same program at same time without interfering each other;
5. the library must be easy to use and install even in existing programs;
6. the insertion of new objects and metric distance functions must be easy even for users that do not know the details about the MAM implementation;
7. the addition of new MAMs must be easy and all support features, such as query visualization support, must be shared among them;
8. all MAMs must be able to store their data in memory and block devices;
9. the storage support must be extensible to make the integration with DBMSs as simple as possible.

All these requirements were used to create an object oriented framework which is based on three distinct layers, each one responsible for one of the basic functions: the object and metric distance functions implementations, the MAM's implementations and the storage access device implementations.

2.3 Design Philosophy

As told in Section 2.2, the design of GBDI Arboretum is based on its requirements. The first requirement did not affected the design of the library since it is related only to implementation issues. The rest of the requirements were decisive to define how this library will be structured.

The main idea behind the project remains on the encapsulation of each component of the library using the object oriented approach. In other words, each component must be able to manipulate its data without the interference of other components. This concept is applied to classes and group of classes related to the same function. The other important concept used was the polymorphism of the components (the ability of a component to behave according to a given interface regardless of its implementation).

The result of these concepts combined with the system requirements is a structure based on three distinct layers stacked one on top of the other. Each layer provides a set of services to the neighbor layers by using a well defined communication interface that never vary regardless of the implementation of each level. It makes possible to complain with all requirements in a easy to use and understand approach.

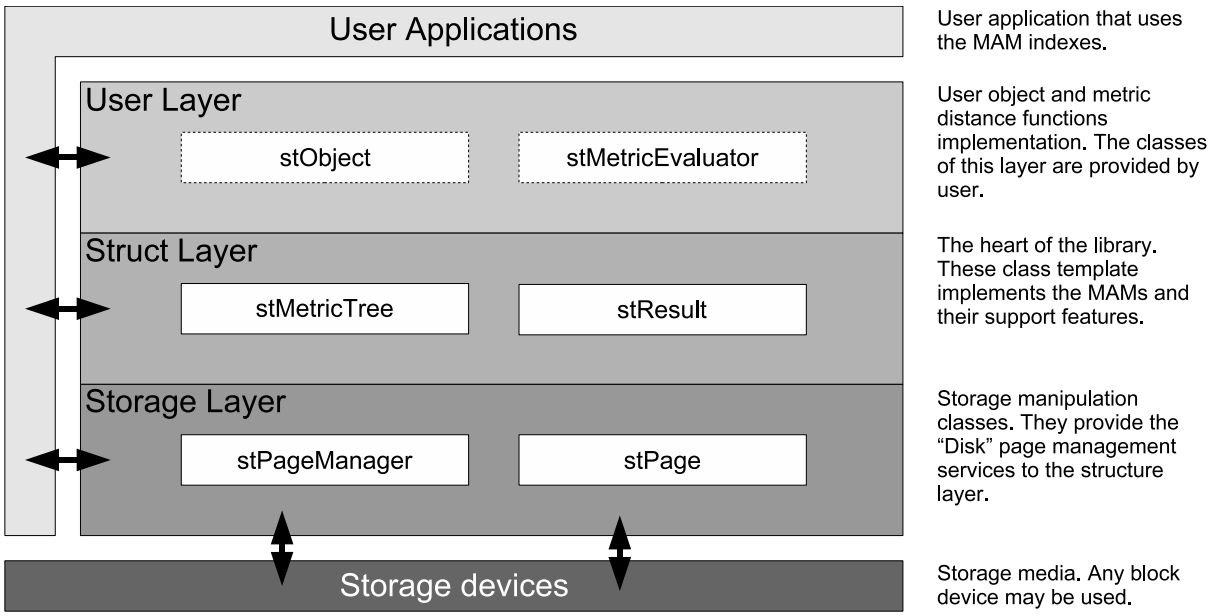

Figure 2.1: The schema of the library layers and the main classes/interfaces of each layer. The dashed white boxes represent the interfaces while solid white boxes represent the classes.

2.4 The three layers

The GBDI Arboretum is divided in three distinct layers, the User Layer, the Structure Layer and the Storage Layer, stacked one on top of the other. Each layer has a fixed communication interface which is used by other layers to request services to this layer. These interface is defined by a set of classes and interfaces which never varies regardless of the layer implementation. Figure 2.1 shows the schema of the three layers, user application and storage devices. It also shows the main classes/interfaces for each level.

The first layer is the **User Layer** which has the knowledge of the object's nature and its metric distance functions. Since the nature of the objects may vary for each object domain, this layer is implemented by users, following a well defined interface for the objects and their metrics. This layer can only communicates with the Structure Layer and the host application.

The second layer is the **Structure Layer** which implements the MAMs. It uses the User Layer classes provided by users to abstract both the object and the distance function. To store the MAM's data, it uses the storage services provided by the Storage Layer. Since this layer must support previously unknown object types, this layer is implemented with class templates and some regular classes.

The last layer is the **Storage Layer** which provides the access to storage devices. The classes of this layer may be compared to "device drivers" since they encapsulate the storage devices to the Structure Layer. This version comes with support for both memory and disk devices but more devices may be added in future version.

These layers are summarized in the following sections and are better described by the Chapters 3, 4 and 5.

2.4.1 User Layer

The User Layer abstracts the objects and their metric distance functions. It defines two class interfaces (not a class hierarchy), stObject and stMetricEvaluator, which are used by the Struct Layer to represent the indexed objects and their related metric distance function respectively.

Usually the classes of this layer are implemented by users following the definition of the stObject and stMetricEvaluator interfaces but this library provides a set of class templates which implement those interfaces for simple array and string types and a few commonly used distance functions such LEdit [**?**] and the Lp family [] distance functions. See Chapter 3 for further details.

2.4.2 Structure Layer

The Structure Layer is the heart of the GBDI Arboretum. It contains the the implementations of all MAMs and their support classes. It uses the user Layer Classes to abstracts the objects and the metric distance functions and stores its data in the disk and/or memory using the services provided by the Storage Layer.

This layer defines two main class templates, the stMetricTree and the stResult. The first is the base class for all MAMs and the later is the class that encapsulates a query result. The use of the base class stMetricTree allows the applications to use all MAMs with the same basic interface. In other words, a MAM can be replaced by other without major changes in the source code.

Since this layer does not need to "know" the implementation of the object and the metric distance function, the addition of new objects and functions does not require modifications in this layer. See Chapter 4 for further details.

2.4.3 Storage Layer

The Storage Layer provides storage services to the Structure Layer. It abstracts all storage devices as a set of services that are accessible using the interface defined by the class stPageManager. So, it looks like a set of device drivers for the Structure Layer.

The idea of the stPageManager is simple, it must provide to the Structure Layer all methods necessary to manipulate a file of disk pages (the class stPage). By definition, this layer can not interact with the User Layer directly. See Chapter 5 for further details.

2.5 Advantages and Limitations of this architecture

All architectures has its advantages and limitations and GBDI Arboretum is not an exception to this. This section will advocate the advantages and also expose the known limitations of the project of this library. The intent of expose the limitations is to allow other developers to help to improve the project and mitigate or even eliminate these limitations.

2.5.1 Advantages

1. user objects and metric distance functions may be added without the knowledge of the MAM and disk access implementations;
2. existing programs may use this library with minimum efforts to install the library to existing objects and distance functions;
3. MAM developers may use this framework to create their MAM implementations without need to implement everything from the roots;
4. the developers can take advantage of some development tools which are integrated to the framework such as query visualization tools and statistic support;
5. MAM developers must provide only one generic implementation of their structures. It is not necessary to create one version for each storage device, object and metric distance function combination;
6. some commonly used functions to perform queries are ready to use;
7. the MAMs implemented by this library can be fairly compared since they use the same base for distance functions and disk access;
8. each combination of object and metric distance function has its own object code (use of templates) but this approach can be easily replaced by a class hierarchy.

2.5.2 Known Limitations

1. The use of templates makes the code a little messy and hard for beginner developers;
2. the use of distinct classes for the object and the distance functions is a quite unusual for object oriented designs;
3. the use of classes add a little penalty to efficiency (class management overhead);
4. the user layer is a quite difficult to understand and implement.

2.6 conclusion

This chapter describes the design issues, requirements and the basic concepts of the architecture of the GBDI Arboretum. It explains how the library is organized in layers and why it is organized in this way. The existence and the arrangement of the three layer is explained as the main aspects of their interaction.

Additionally, there is a section that explains the advantages and limitations of GBDI Arboretum which provide additional information to the understanding of the architecture and their design principles.

Chapter 3

The User Layer

3.1 Overview

3.2 Functionality Description

3.3 Layer abstraction roles

3.3.1 The Object

3.3.2 The Metric Distance Function

3.3.3 The Metric Evaluator

3.4 User Layer interfaces

3.4.1 The stObject interface

3.4.2 The stMetricEvaluator interface

3.5 Serialization and unserialization principles

3.5.1 Serializing an Object

3.5.2 Unserializing an Object

3.5.3 Implementation Hints and Tips

3.6 Integration with user applications

3.7 Conclusion

Chapter 4

The Structure Layer

Chapter 5

The Storage Layer

5.1 Overview

This chapter describes the Storage layer and its classes in detail. It explores the Storage Layer classes implementation in deep, providing to developers all the information required to develop new media support and to integrate this library to existing DBMSs.

5.2 Functionality Description

The Storage Layer, as its name suggests, must provide the storage services to the Structure Layer. Since the Structure only needs "pages"[1] which can be identified by a given ID, it is almost possible to think of Storage Layer as a simple page repository.

The services provided by this layer are:

1. Allocation o new pages;
2. Reading and writing of pages;
3. Recycling of unused pages.

The Structure Layer uses the classes of this layer through the stPage and stPageManager interfaces. The first abstracts a single page while the later abstracts the storage services (a page repository). The details about these classes and interfaces are described in details in the next sections.

5.3 Layer abstraction roles

This layer defines four abstractions roles. Two of then are related to the interface with the Structure Layer (Page and Page Managers) while the other two are concerned to the integration with existing DBMSs (Page Servers and Page Clients). They are described in the following sections.

5.3.1 The Page

A page is a raw array of bytes with a given size which is stored somewhere and can be identified by an ID which is an integer number greater than zero (zero is used to indicate an invalid ID).

When stored, a page may contain a header but this information is never visible to the Structure Layer which can only sees the page's usable space. In the same way, the contents of a page is unknow to it, so, page managers can not make any assumptions about the page contents.

The basic interface of the Page Managers are defined by the class stPageManager which is described in details in the Section and by the on-line reference guide [**?**].

5.3.2 The Page Manager

The most important idea behind the Page Manager abstraction is the idea of a file of pages. In other words, a repository of pages with a given size. The Structure Layer classes are not concerned about how the page managers work provided that they are able to perform the services described in Section

[1] This document will use "pages" to refer to an array of bytes with fixed or variable length that can stored in a given media, usually disks but not restricted to.

5.2. This allows the Page Managers to store their pages in any kind of storage media without affect the implementation of the other layers.

Most MAMs are designed to use only fixed size pages but is is not granted by the storage layer that all pages of a Page Manager will have the same amount of useful space. So, it is possible to create a Page Manager that holds variable size pages but this practice is not recommended due to efficiency issues. To the Structure Layer classes, each Page Manager appears to be a private single file that can not be used by more than one instance at once.

The basic interface of the Page Managers are defined by the class stPageManager which is described in details in the Section and by the on-line reference guide [**?**].

5.3.3 The Page Server and the Page Client

As told in Section 5.3.2, the Structure Layer see each Page Manager as a single file that can be used by only one structure at once. In fact, the interface of Page Managers is not designed to allow multiple instances of MAMs to share a single instance of a Page Manager. Since each Manager is associated to a single "page file", it is a quite difficult to implement Page Managers that share the same physical repository.

To allow avoid this "limitation" of Page Managers, the Storage Layer has the concept of Page Servers and Page Clients. The Page Client is a standard Page Manager implementation that is not associated to a physical repository but to a Page Server.

The Page Server is a special page repository that can accept "connections" from various Page Clients. These Page Clients forward the services requested by the Structure Layer classes to their Page Servers and format the server response to the format expected by the Structure Layer. Each client must be associated to a single segment in the server (a logical "file") which is identified by a positive integer number.

This approach allows the implementation of Page Servers that can be built on top of a DBMS storage module and provide all services required to the Storage Layer to work with the minimum efforts possible. There is no need to modify the implementation of other classes of the library.

The implementation of this abstraction is very effective to keep the simplicity of the Storage Layer and to add the minimum overhead (the implementation of the Page Client is almost weightless).

5.4 Storage Layer Classes

5.4.1 The stPage class

This class is the basic implementation of the Page concept. All Page implementations must implement all methods defined by this class.

It works as a small file that the structure can use to read and write information. It also allows the Structure Layer to format the contents of this pages as a memory segment by using a cast to a data structure. All page managers uses this class or one of its specializations to communicate with the Structure Layer.

void Clear()

This method sets all useful bytes of the page to zero. It is mainly used for debug purposes.

void Copy(stPage * page)

This method copies the contents of a page to another. The page ID is not copied because each stPage instance is associated with an specific physical page. Both source and destination pages must have the same size.

stSize GetPageSize()

This method returns the size of the page in bytes.

stByte * GetData()

This method provides access to the page usable data. This method must be used with special care to avoid buffer overflows.

stPageID GetPageID()

This method returns the page ID. It may return zero if this instance is not associated to any valid physical page.

void SetPageID(stpageID pageID)

This method sets the page ID of this instance. It is used by Page Managers only.

void Write(stByte * buff, stSize size, stSize offset)

This method writes the content of a buffer with given size.

5.4.2 The stLockablePage class

This class is a specialization of stPage that allows the Page Managers to lock portions of the page data to the Structure Layer. This class makes possible to add a small header to the stored page. Since it adds only a few methods which are not required by the Structure Layer, this class will not be detailed in this documentation. See the on-line reference guide [**?**] for more details about the implementation.

5.4.3 The stPageManager class

5.4.4 The stDiskPageManager class

It is an specialization of the stPageManager that stores its data in a single file on the file system. See the on-line reference guide [**?**] for more details about the implementation.

5.4.5 The stMemoryPageManager class

It is an specialization of the stPageManager that stores its data in private page repository in memory. See the on-line reference guide [**?**] for more details about the implementation.

5.4.6 The stClientPageManager

This class is an specialization of the stPageManager that can connect to a Page Server end request its services. Its interface is almost the same of the stPageManager except by the methods Connect() and Disconnect(). The first is used to connect to a Page Server and other to disconnect from it.

This class is not fully implemented by GBDI Arboretum yet.

int Connect(stPageServer * server, int segmentID)

This method creates a connection between this Page Client and a Page Server. The parameter connectionID identifies the segment to be used by this client. If this parameter is negative, a new segment will be created.

Only one page client can be associated to a segment in Page Sever at same time. The segment access control is done by the Page Server.

This method returns the segment ID or a negative value for fail.

void Disconnect()

This method disconnects this Page Client from the Page Server. All associated resources will be released.

5.4.7 The stPageServer class

The class stPageserver is a pure abstract class that is the base for all Page Servers implementations. Its interface provides all services that a Page Client may need to perform its job. The implementation of Page Servers must be thread-safe.

The interface of this class is not fully specified and may change in future versions of GBDI Arboretum, so it is not fully implemented by GBDI Arboretum yet.

int CreateNewSegment()

This method creates a new segment with only one page, the header page. It returns the ID of the segment or a negative value for failure.

bool DisposeSegment(int segmentID)

This method disposes a given segment. All associated pages are disposed too.

bool DisposePage (int segmentID, stPage *page)

This method marks a page as available for reuse. It may fail the given page is not associated to the given segmentID.

It returns true for success or false otherwise.

stPageID GetHeaderPageID (int segmentID)

Returns the page ID of the header page associated with a given segment. This method is used by the Page Clients to determine the header page ID of its segment.

stPage * GetNewPage (int segmentID)

This method allocates a new page and associate it with the given segment ID. It may returns NULL if the page can not be created or the segment does not exist.

stPage * GetPage (int segmentID, stPageID pageid)

This method reads a page from the Server. The read page must be protected for writing and reading. It may return NULL if the segmentID or the pageID is invalid.

stSize GetPageCount (int segmentID)

This method returns the number of pages associated with a given segment.

void ReleasePage (int segmentID, stPage *page)

This method reports to the Server that the given page will not be used for reading and writing. All existing locks that protect the page may be released and the instance of stPage can be disposed or recycled.

void WritePage (int segmentID, stPage *page)

This method updates the page content in the Server. It does not release the page, so the method ReleasePage() must be called to release all locks added to the page.

5.4.8 The extensions of stPageServer class

Since all implementations Page Servers must be thread-safe, this library does not provide functional implementations of them (except for file support for some systems).

The Page Servers were designed to be used as a integration class to existing DBMSs, so their implementations must be provided by users.

5.5 Integrating the Storage Layer to existing DBMSs

There is two ways to integrate the Storage Layer to an existing DBMS. The first one is to create a Page Manager that can interact with the DBMS' storage module and the other is to create a Page Server that also interacts with the DBMS' storage module.

Unfortunately it's impossible to suggest the best approach for each DBMS. The choice of the best approach to be taken depends on the DBMS' storage module. In general, the first approach is easier to implement but may add some undesirable complexity to the manipulation of multiple instances of age Managers. The second approach is a bit more complex but can take full advantage of the DBMS' storage module in a more appropriated fashion.

The integration of of this library to a existing DBMS depends on the implementation of an appropriated user layer too. See Section **??** for further details.

5.6 Conclusion

Chapter 6

GBDI Arboretum and user applications

Chapter 7

Adding a new Metric Access Method

Chapter 8

Writing GBDI Arboretum source code

8.1 Overview

The GBDI Arboretum is developed by many different persons with distinct coding styles, so it is necessary to follow a standard to write its code in order to keep it as uniform as possible. These standard rules are intend to add a "personality" to the library implementation regardless of the author of each piece of code, making it easier to read and more predictable for both developers and users, specially the beginners. This chapter describes all standard rules that must be applied to the source code of the GBDI Arboretum.

This chapter also contains rules to write a portable code, instructions about the use of Doxygen, a semi-automated documentation tool used in this project and some generic implementation hints and tips.

8.2 Naming convention

8.2.1 File names

All files in this library must start with "st" followed by a significant name[1]. The significant name must reflect what is implemented/declared inside. They may be composed by one or more English words, all starting with the first letter capitalized. Take the file "stBasicObjects.h" as an example. It contains the declarations of all basic objects that are implemented by GBDI Arboretum.

Another important part of the file names is their extensions. In GBDI Arboretum, there are only 3 possible file extensions that may be used in the implementation. They describe the file content type, so they must be chosen properly.

The extension ".h" is reserved for headers only. As headers, these files can contain only data types, structures, macros and class declarations. Function implementations are not allowed inside them except to in-line methods and template function implementations.

Each header file that contains templates may be associated to a ".cc" file with the same name. These ".cc" files can contain template implementations only. Finally, the ".cpp" files, which are also associated with a header file with the same name, contain the implementation of functions and methods.

A complete example of this convention may be found in the file "stUtil.h" and its companions, "stUtil.cc" and "stUtil.cpp". The "Util" word comes from what is inside, in this case utility classes, functions and macros. Since it contains templates, it is associated with a ".cc" file that contains the implementations of these templates. It is also associated with a ".cpp" file because it contains non-template classes and functions declarations.

The only exceptions to these rules are the files "CNSCache.h", "CStorage.h" and "CStorage.cpp" which were implemented prior to the beginning of this project. These files are not a formal part of the library and may be removed in future versions.

8.2.2 Class and data type names

Class and data type names follow the same rules applied to file names. They must start with "st" followed by as significant name which may be composed by one or more English words, all starting with the first

[1]The starting "st" of all file names has no special mean in current version of GBDI Arboretum but was kept due to historical reasons. In the very beginning of this project, it was intented to hold only the Slim-Tree implementation and from that comes the "st".

letter capitalized (e.g.:'"stMyLittleClass"). The same rule must be applied to global structure and data type names and those which are public or protected members of classes. There is no rules defined for private data type and structure names because they are not accessible outside its class.

The only exceptions to these rules are the classes defined in the files "CNSCache.h", "CStorage.h" and "CStorage.cpp" which are not a formal part of the library and may be removed in future versions.

8.2.3 Method, function and parameter names

All method names must be composed by one or more English words, all starting with the first letter capitalized (e.g.: "MyLittleFunction(...)"). This name must reflect the functionality of the method and may be overloaded only, and only if the overloaded methods perform the same function with little variations.

Method parameter names have no formal rule but it's recommended to create them by using one or more English words concatenated. The first word starts with the first character in lower case while others begin with the first character capitalized (e.g.: "myLittleParam") as Java method and variable name convention. This recommendation may become a rule in future revisions of this library.

Global functions must begin with "st" followed by a name that follows the same rules applied to method names, in other words, "st" followed by one or more English words with the first letter capitalized.

8.2.4 Variable, fields and constant names

The only restriction applied to internal variable names is that they must be compose by English words. Each developer may define them as his/her wish. Class and structure field names follow the same rules applied to class method names regardless of their scopes (see Section 8.2.3).

A local constant name has all of its words capitalized and may contain "_" character. The global constant names have the same formation rule except that they must always begin with "st". There is no rule to govern global variable names because they are not allowed in this project.

The macro names follow the same rules applied to other names depending of its nature. Macro functions names follow the same name rules applied to global function names while constant macro names follow the same rules applied to constant names and so on.

Header protection macros must begin with "__" (two underscores) followed by the header name with all letters capitalized and the dots replaced by a "_" (e.g.: "__STUTIL_H").

There are a few exceptions to macro naming which are the conditional compilation macros. They start with "__st" (two underscores followed by "st") followed by capitalized English words and ends with "__" (two underscores). See Section 8.5 for further details about the conditional compiling.

8.2.5 Template names

Template names follow the same rules of class and global function depending on its nature. The template parameters have the same naming rules of field names (see Section 8.2.4).

There are two special template parameter names that must always remain the same regardless of the class used. They are **ObjectType** and **EvaluatorType** which are used for the object type and the metric evaluator type respectively. More fixed template names may be added in future versions of this document.

8.2.6 General recommendations

This section contains additional recommendations that may be applied to naming conventions. They are not formal rules but their use are encouraged by the authors of the present document because they can make the code much easier to read and understand.

- avoid ambiguous abbreviations;
- use more than one word to describe the name if possible (do not be afraid of long names);
- do not exaggerate with name lengths, something as "ThisMethodSortsTheEntriesUsingQuickSort()" is beyond the common sense;
- use the same initial word or words to name related classes and types.

8.3 Coding style

8.3.1 Indentation

The indentation is the most simple but powerful mean to get code readability of a source code. GBDI Arboretum uses three blank spaces to indent each block level. Tab characters are not allowed and must be replaced by three spaces when found. For example:

```
level 1
level 1
   level 2
   level 2
      level 3
level 1
```

8.3.2 Blocks

The beginning of each block ("{") must be placed at the end of the same line of the statement that requires its use. Additionally, after the end of each block ("}"), a brief comment about what it ends is required except when it is followed by the **else** keyword. For example:

```
void MyFunc(...){
   while (...){
      if (...){
         switch (...){
            case 0:
               ...;
               break;
            case 1:
               ...;
               break;
            default:
               ...;
         }//end switch
      }else{
         ...
      }//end if
   }//end while
} //end MyFunc
```

8.3.3 Comments

Internal comments

There is no restrictions to the internal function/method comments except that they must be written in English. Developers are encouraged to put as many comments as possible.

Class, data type, methods and function descriptions

The GBDI Arboretum uses a semi-automated documentation tool, called **Doxygen**, to generates its reference guide. Because of that, these comment blocks are formated according the specification of this tool. See Section 8.7.3 for further details.

Copyright and licence notices

Each file must contain the following comment near the beginning of the file.

```
// Copyright (c) 2002-2003 GBDI-ICMC-USP
```

End block comment

At the end of each block, a comment about what the "}" ends must be added. See Section **??** for more details.

Method and function implementation separators

To make the code easier to read, each function or method must be separated by a comment line composed by a single line comment ("//") followed by at least seventy "-" characters. This separator is also used to mark the beginning of the implementation of each class methods.

In the later case, the separator must be followed by a single line comment with the name of the class followed by a another separator. This class implementation delimiter is used to mark the beginning of a class declaration too. For example:

```
//---------------------------------------------------------------------
// Class ClassName
//---------------------------------------------------------------------

void ClassName::Func(){
   ...
}//end ClassName::Func
//---------------------------------------------------------------------

void ClassName::Func2(){
   ...
}//end ClassName::Func2
```

8.4 C/C++ Libraries and Portability Issues

The code of GBDI Arboretum must be portable among various compilers and system. So, it is necessary to take additional attention to a set of small but important details while writing. The most important one is the set of libraries and functions that may be used in the implementation.

GBDI Arboretum is supposed to be compiled in any ANSI C++ compiler. Because of that, the only libraries allowed are the standard ANSI C library and the standard ANSI C++ library. It's important to remind that some popular functions are not defined by ANSI (various of them are POSIX functions). So, if you are not sure about a functions or class, check its documentation.

Another issue related to ANSI standards is the use of some non-standard keywords and data types (compiler extensions) such as Borland C++ **__property**, GNU GCC **__attribute()__** and others. Almost all compilers provide such kind of extension. They all must be avoided at any costs regardless of the "comfort" they provide unless their use are protected by a compiler specific macro. See Section 8.4.3 for further details.

8.4.1 Headers

Some standard C library headers may contain different sets of functions. This make a simple operation, like include a standard header, a source of problems for some compilers. To minimize that, there is a file called "stCommon.h", which "mitigates" this problem by use a set of compiler specific headers to include the definitions of the most used functions and types (memory management (memcpy(), memset() and others), basic math functions and system functions). It is more complete version of the "stdlib.h" header.

8.4.2 Data types

The data types also require a little attention. Do not rely on your knowledge about data type sizes because they may vary from system to system and even from compiler to compiler (**int** type is a good

example, it may vary fom 16 to 64 bits). Always use the **sizeof** operator. Any assumptions about numeric formats must not be considered too since the the target system may store than in an unknown way (remember the little endian[2] and big endian case[3]).

8.4.3 System specific implementations

The use of system specific functions must be avoided and restrict to the minimum. When they are used, they must be protected by a system specific macro and a default standard implementation must be provided too. The same rule must be applied to assembly code[4]. For example:

```
#ifdef __WIN32__
   // Windows specific code
   ...
#ifdef __GNUC
   // GCC specific code
   ...
#else
   // Generic code
   ...
#endif //__GNUC
#else
   // Generic code
   ...
#endif //__WIN32__
```

These macros are compiler specific but their use are harmless regardless of the compiler C/C++ compiler (they are predefined or not defined) and may be freely used.

8.5 Optional features and Conditional Compiling

Some features of GBDI Arboretum may degrade the performance of the implemented MAMs (such as visualization and statistics support) or may lead to undesirable behavior in user applications (extra debug features). Because of these "side effects", they must be implemented as optional features that are activated or deactivated during the compilation time.

In the current version, there are only 3 features which are considered optional. The pieces of code responsible for each of these features are protected by these macros:

- **__stDEBUG__**: Extra Debug information. This feature enables the debug features hidden inside the code. They may provide extra checking and some console verbose messages. They add a little performance degradation but they may mess console application interfaces with undesirable messages.
- **__stSTATISTICS__**: Performance statistics. This feature enables the support for statistics. Some of them are always available because they are very cheap while other will not be compiled when this feature is not enabled.
- **__stVIEW__**: Visualization Module. This macro enables the support to **MAMView** visualization tool. This feature is very expensive and should not be enabled when it is not required. This module is under development and may be used by MAM developers to track down the behavior of the structure.

The following example shows how these macros must be used:

[2]Little endian is a integer enconding schema where the less significant byte is stored first. Intel processors use this format.

[3]Big endian is a integer enconding schema where the most significant byte is stored first. Motorola processors use this format.

[4]The modern compilers are able to create very optimized assembly codes, as good as any assembly programmer can make. The only exceptions to this are related to some processor's SIMD extensions that are not properly used by the compilers

```
class stSample{
...
   #ifdef __stDEBUG__
   void PrintMyFields();
   #endif //__stDEBUG__
...
};//end stSample
...
#ifdef __stDEBUG__
void stSample::PrintMyFields(){
   ...
}//end stSample::PrintMyFields
#endif //__stDEBUG__
...
```

8.6 Source Directory Tree

The GBDI Arboretum source code is divided in six main directories that may contain subdirectories. Each main directory contain a vital part of the source which groups files by their roles.

docs

The documentation directory. All related documentation is stored in this directory, including the source of this document and the configuration file required to generate the semi-automated documentation (see section 8.7.3 for more details).

include

The header directory. It contains the sub-directory "arboretum" which contains all headers and template implementations. It may be added to the compiler include search path to compile the library and programs which use it.

lib

The library binary and project directory. It contain the library binaries and the projects to compile them.

sample

The sample program directory. It contain source code to sample programs written to exemplify the use of the library. Each program is stored in a single subdirectory. Some of them may be system specific.

src

The source code directory. It contains the implementations of the non-template classes and functions of the library.

test

The test program directory. It contains the source codes of test programs and special developer tools. This is the most volatile directory in the library and may change at the will of developers. Each program is kept on its own subdirectory.

8.7 Development support tools

8.7.1 Compilers

The developer of this library is free to choose his/her preferred compiler provided that it is an ANSI C/C++ compliant compiler. Developers are encouraged to use a wide variety of compilers and help to assure the portability of GBDI Arboretum by detecting and fixing eventual portability flaws.

Current supported compilers are Borland C++ Compiler, GNU GCC and Microsoft Visual C++ Compiler but the support to other compilers are welcome.

8.7.2 Concurrent Versions System - CVS

The source code and the documentation of GBDI Arboretum are controlled by **CVS**, a simple but powerful open source distributed version system. See the **CVS** Internet site (http://www.cvshome.org) for more information about CVS.

The only restriction about the use of CVS remains about the commentary of each modification. They must be posted in English and must reflect the difference between versions as much as possible. The use of CVS is mandatory to all developers.

8.7.3 Automated Documentation

This library uses an open source semi-automated documentation tool called **Doxygen** to generate its reference guide. It very is similar to Sun's **Javadoc** tool but capable to perform the same task for C/C++ projects. It generates the reference guide by extracting special C/C++ commentary blocks which are used as regular internal documentation too.

The format of this documentation blocks and usage instructions of this tool may be found at Doxygen manual, found in its Internet site (http://www.doxygen.org). The use of this tool is mandatory to all GBDI Arboretum components and files.

The configuration files to generate the reference guide using Doxygen can be found in the directory "doc".

Doxygen comment block

Doxygen defines two forms of comment blocks, one which starts wiht "/*!" and other which starts with "/**". In this library, the later format is preferred. Because of this choice, all Doxygen commands starts with "@".

All classes, structures, functions and data types must have a Doxygen documentation block (always in English) which must appear once in the whole source code, preferred in the header files. All files must have a Doxygen comment block to describe their contents.

Mandatory Doxygen tags

Doxygen does not define any mandatory tag but in this project certain tags are always required for certain blocks. The class, macro and global function documentation blocks must contain the tags "@author" and "@version". All global function, macro and method documentation block must have all parameters and returning values documented (tags "@param", "@return" and "@retval"). All others are optional and may be used without restrictions.

A special attention must be given to the tag "@author". It must contain the name of the author followed by its e-mail. For each block which this tag is required, an "@author" tag is required for each developer who has worked in this component no matter how small his/her contribution was. The order of the authors must respect their importance. This must be done to keep the track of the authors to be blamed for their fails (just kidding!).

Organizing the documentation in modules

To keep the automated documentation well organized, all classes, structures, functions and data types must be assigned to a module defined by the tag "@defgroup ¡group id¿ ¡description¿". A member is added to a group by adding a tag "@ingroup ¡group id¿" to its comment block. The division of the file members in groups is described by table 8.1.

All group definitions are placed in the file "doxygen.dxg". See section 8.7.3 for more details about this file.

The doxygen.dxg file

Some global doxygen delcarations such group definitions and the main page must be kept together to make this management easier. These definitions are placed in a file called "doxygen.dxg" which is a regular text file that contains regular C/C++ comments and Doxygen formated blocks.

It is located in the same directory of the headers file but is not included by any source file. In the final library distribution, this file may be removed in order to save a few bytes in the file system.

Group name	Group identifier	
Exceptions	exceptions	All exception classes.
FastMap Implementation	fastmap	Internal FastMap [] implementation components.
MAMViewer Extractor	mamview	MAMView Extrator module implementation components ex
Standard Types	types	Standard types definitions.
Storage Layer	storage	Storage Layer components.
Structure Layer	struct	Common Structure Layer components. Each MAM has its
Structure Layer - DBM-Tree	dbmtree	DBM-Tree components.
Structure Layer - Dummy-Tree	dummy	Dummy-Tree components.
Structure Layer - Partition-Tree	partition	Partition-Tree components.
Structure Layer - Slim-Tree	slim	Slim-Tree [] components.
User Layer	user	User Layer default implementation classes and interface dec
User Layer - Utilities	userlayerutil	User Layer utility components. These components are used
Utilities	util	Utility components. See Appendix A for further informatio

Table 8.1: List of Doxygen modules and their contents.

8.8 File and class member scopes

8.8.1 Global functions

Global functions must be defined only if they are not specific to a given MAM and may be useful in other parts of the code such as math functions.

8.8.2 Global Constants

Global constants must be avoided. It is better to keep them local to classes unless they are really global to all library members which are rare situations.

8.8.3 Global data types

Global data types and structures may be defined only if they are global to all library members otherwise they must be defined local to classes.

Class member visibility

All methods, data types and constants that are part of the external class interface must put in the public section. The use of public fields are not allowed. The private and protected sections may contain any kind of members.

It's a bit difficult to decide whenever a member must put on the private or protected scope. The general recommendation to that is, when unsure, put it in the protected scope because they will affect the subclasses only.

The use of the **friend** classes and functions must be avoided at any costs but their use is not prohibited.

8.8.4 Class creation recommendations

The following recommendations must be followed to keep the integrity of the class hierarchy and the object oriented concepts.

- do not let the class interface expose the class internal working;
- do not extend a class only because it has similar functions. The inheritance must be placed only if the proper abstraction is met;
- do not create methods that grants direct access to a class field unless it is really necessary or the field is too simple to be malicious explored to change the normal class behavior. In most cases, these situations are generated by a bad design;
- class local constants are created by declaring local enumerations. The same is valid to structures;
- internal data types and constants must be in the protected or private scope.

Interfaces

To make the Reference Guide easier to use, some interfaces, such as stObject and stMetricEvaluator, are materialized as abstract classes because C++ does not have the concept of interfaces[5].

These abstract classes have the same name as their interface name but are not supposed to be used by users and MAM developers except when their documentation states otherwise. This is not the best approach but it is very simple and was adopted due to the inexistence of better solutions.

8.9 Templates

Some classes and functions of GBDI Arboretum must be implemented as class templates because they manipulate User Layer classes (see Chapter 3 for further details) or they are planned to be applied to various data types (like stGenericMatrix). All other functions and classes can not be implemented as templates.

The template declarations must always be in header files while their implementation must be inside a ".cc" file except for in line templates. The tipical template header file seen as follow:

```
#ifndef __STTEMPLATE_H
#define __STTEMPLATE_H
...
// Template declarations
template <class...
...
// Include template implementation
#include <slimtree/stTemplate.cc>
#endif //__STTEMPLATE_H
```

There is two template parameter names that must reamain the same no matter what class they are. These parameters are "ObjectType", for user objects, and "EvaluatorType", for user metric evaluators. See Section 8.2.5 for more details.

8.10 Frequently Asked Questions

This section contains a set of common implementation doubts and their .

Q1: How do I change the memory alignment of a structure without change it for the rest of the code ?

Use the "#pragma pack(n)" before the structure declaration and "#pragma pack()" after the declaration to restore it the default. See Q2 for more details about memory alignment.

Q2: What is the memory alignment and why it is good to set it to 1 in some parts of the code ?

To accelerate the memory access, the compilers aligns structure fields to the size of the processor word or the memory bus size. In other words, a structure composed by 2 chars, which is supposed to have 2 bytes, may be allocated to 2 processor words, one for each field. It generates unused bytes among fields which are undesired in a disk node structure for example. So it's good to set the alignment to 1 when defining a structure taht will be written to disk. See Q1 for more details.

Q3: stDistance is a double. Why I should not use double instead of stDistance ?

This types were defined to allow users to change these data types by modifying its definitions int the file "stTypes.h". So it is not granted that stDistance and other types will always be the their defaults types.

[5]Some languages, such Java, have special structures to define interfaces.

Q4: Why almost all GBDI Arboretumnames begin with "st" ?

This was done to minimize the possibility of naming conflicts among other libraries without the need of namespaces. The "st" was chosen in earlier stages of the project. See Section 8.2 for further details.

Q5: stObject and stMetricEvaluator are supposed to be interfaces. Why there are classes with that name ? Should users use them ?

It is true. They are interfaces. These classes exists as empty examples of these interfaces and to keep the documentation of the interface in the reference guide. They are not supposed to be used.

Q6: Who implements the user layer at all ?

The user layer is supposed to be provided by users. They may add almost all sort of objects and implementations as they wish without changes the MAM implementation provided these classes comply with the user layer interface.

Q7: What should a developer expect from the unknown user layer classes ?

The only thing a MAM developer should expect from the user layer is that it is performing what the specification tells. The correct behavior of the user layer is a user responsibility.

Q8: Why some parts of the library are implemented as templates while others are not ?

Classes which have no relationship with the user layer are not templates. The classes that are manipulates user objects must be templates provided the user classes are unknown to MAM developers. It makes possible for users to incorporate this library to their existing projects without substantial changes to project's class hierarchy.

Q9: I found a function called random() which exists only in Windows compilers. Why is it allowed ?

This is a very useful function which can return random numbers between 0 and a given value. Sice it's easy to implement using macros, this function is provided by the header "stCommon.h" for all compilers and systems.

8.11 Conclusion

This chapter describes the rules which must be used to define the create new software components to GBDI Arboretum. The use of these coding rules is essential to keep the source code uniform, making it easy to read regardless of the authors of each part of the code.

It also contains instructions required to write the optional features which must be enabled or disable during the compilation time, instructions about writing portable code in C/C++ and hints about miscellaneous programming issues.

This chapter does not intent to provide a full software engineering process but set of tools which will help multiple developers to work together.

Chapter 9

Conclusion

Bibliography

[Burkhard and Keller, 1973] Burkhard, W. A. and Keller, R. M. (1973). Some approaches to best-match file searching. *Communications of the ACM*, 16(4):230–236.

[Ciaccia et al., 1997] Ciaccia, P., Patella, M., and Zezula, P. (1997). M-tree: An efficient access method for similarity search in metric spaces. In Jarke, M., editor, *Proceedings of 23rd International Conference on Very Large Data Bases (VLDB)*, pages 426–435, Athens, Greece. Morgan Kaufmann Publishers.

[Santos et al., 2001] Santos, Roberto Figueira, F., Traina, A. J. M., Traina, Caetano, J., and Faloutsos, C. (2001). Similarity search without tears: The OMNI family of all-purpose access methods. In *International Conference on Data Engineering (ICDE)*, pages 623–630, Heidelberg, Germany. IEEE Computer Society.

[Traina et al., 2000] Traina, Caetano, J., Traina, A. J. M., Seeger, B., and Faloutsos, C. (2000). Slim-trees: High performance metric trees minimizing overlap between nodes. In Zaniolo, C., Lockemann, P. C., Scholl, M. H., and Grust, T., editors, *International Conference on Extending Database Technology*, volume 1777 of *Lecture Notes in Computer Science*, pages 51–65, Konstanz, Germany. Springer.

Appendix A

Utilities

The GBDI Arboretum implementation has a set of utility classes that may be used in the implementation of the library and/or applications which uses this library. This appendix brings a brief descriptions of these classes, functions and macros. For further details see the their complete documentation in the GBDI Arboretum Reference Guide.

A.1 Classes

A.1.1 stGenericMatrix class template

This class template implements a generic dynamic n by m matrix which may be defined for any basic C++ data type.

A.1.2 stRPriorityQueue class template

This class template implements reversed priority queue with a fixed maximum number of entries. In other words, the smallest key value has the greater priority. Since it is implemented as a heap, it takes only $O(\log n)$ comparisons to get the minimum value of the queue. This class template is used as local priority queue by the implementation of queries.

A.2 Functions and Macros

Appendix B

Portability issues

B.1 Minimum compiler requirements

B.2 Runtime Libraries

B.2.1 Standard C Library

B.2.2 Standard C++ Library

B.2.3 Other standard functions

Some functions and constants that are not provided by both Standard C++ Library and Standard C Library are allowed since they are very useful or are required to get some information about the system and its properties. Most of these functions and constants are defined by POSIX libraries but are available for a large range of systems and, when not available, are implemented by this library ()

Appendix C

GNU Free Documentation License

Version 1.2, November 2002

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus

accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties–for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and

all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document 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.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation 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. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

```
Copyright (c)  YEAR  YOUR NAME.
Permission is granted to copy, distribute and/or modify this document
under the terms of the GNU Free Documentation License, Version 1.2
or any later version published by the Free Software Foundation;
with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts.
A copy of the license is included in the section entitled "GNU
Free Documentation License".
```

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with...Texts." line with this:

```
with the Invariant Sections being LIST THEIR TITLES, with the
Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.
```

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Appendix D

Credits

D.1 GBDI Arboretum Project Staff

Project Coordinators:
Agma Juci Machado Traina and Caetano Traina Jr.
Architecture Concepts:
Fabio Jun Takada Chino and Enzo Seraphim
Architectural Project:
Fabio Jun Takada Chino, Enzo Seraphim, Adriano Arantes and Josiel...
Project Manager/Leader Programmer:
Fabio Jun Takada Chino
Programmers:
Fabio Jun Takada Chino, Marcos Rodrigues Vieira, Adriano Arantes, Enzo Seraphim and Josiel...
Dummy-Tree Programmers:
Fabio Jun Takada Chino, Marcos Rodrigues Vieira and Adriano Arantes
Slim-Tree Programmers:
Fabio Jun Takada Chino, Marcos Rodrigues Vieira, Adriano Arantes and Josiel...
Partition-Tree Programmer:
Enzo Seraphim
DBM-Tree Programmer:
Marcos Rodrigues Vieira

D.2 Documentation

Reference guide and Internal Documentation:
All staff
Developer's Guide:
Fabio Jun Takada Chino and Marcos Rodrigues Vieira
Documentation Revision:
All staff

D.3 Special Thanks

To FAPESP...