Contents

Abstract **1**

1 Introduction **2**

1.1 Problem Definition 2

1.1.1 Archival 3

1.1.2 Compression 3

1.2 Proposed Solution 4

1.2.1 Functional Objectives 4

1.2.2 Technical Objectives 5

1.3 Benefits & Features 5

2 Competitive Analysis **7**

2.1 Competitors 7

2.1.1 WinZip 8

2.1.2 Ark 10

3 System Requirements **12**

3.1 Hardware 12

3.2 Software 12

4 Required Resources **14**

4.1 Build Tools 14

4.1.1 BeIDE 14

4.1.2 GCC 16

4.2 Test Tools 17

4.2.1 Debugging Macros 17

4.2.2 BeDebugger 19

4.2.3 Other Debugging Methods 22

4.2.4 Stress Tests . . . 23
4.3 Documentation Tools . . . 24
4.3.1 CoolCat . . . 24
4.3.2 LaTeX . . . 24

5 Design & Implementation 26
5.1 Design Goals . . . 26
5.2 Concept Diagram . . . 27
5.2.1 Interface . . . 27
5.2.2 Archiver . . . 28
5.2.3 Add-Ons . . . 29
5.2.4 Other components . . . 30
5.3 FileType rules . . . 30
5.4 Compile Timestamp . . . 31
5.5 Add-ons & Workers . . . 31

6 Usage Guidelines 32
6.1 Installation . . . 32
6.2 Interface . . . 32
6.2.1 Welcome window . . . 32
6.2.2 Archive window . . . 33
6.3 Performing Actions . . . 40
6.3.1 Creating archives . . . 40
6.3.2 Deleting archives . . . 42
6.3.3 Searching archives . . . 42
6.3.4 Testing archives . . . 44
6.3.5 Extracting archives . . . 44
6.3.6 Archive information . . . 45
6.3.7 Adding files . . . 46
6.3.8 Viewing files . . . 47
6.3.9 Creating folders . . . 48
6.3.10 Editing comments . . . 48
6.4 Tools . . . 49
6.4.1 QuickCreate archive . . . 49
6.4.2 File Splitter . . . 51
6.4.3 File Joiner . . . 53

6.5 Preferences 54
6.5.1 Restoring default options 55
6.5.2 Extract panel 56
6.5.3 Add panel 56
6.5.4 State panel 56
6.5.5 Paths 57
6.5.6 Recent panel 58
6.5.7 Interface panel 58
6.5.8 Miscellaneous 59

7 Source Code **61**

8 Results **65**

9 Conclusion & Future Scope **67**

10 References **68**

List of Figures

2. Competitive Analysis **7**

2.1 *WinZip* 8

2.2 *Ark* 10

4. Required Resources **14**

4.1 *BeIDE: Project window* 15

4.2 *BeIDE: Code window* 16

4.3 *BeDebugger: Running Teams window* 20

4.4 *BeDebugger: Team window* 21

4.5 *BeDebugger: Thread window* 22

5. Design & Implementation **26**

5.1 *Concept Diagram* 28

6. Usage Guidelines **32**

6.1 *Welcome Window* 33

6.2 *Archive Window* 34

6.3 *Context Menu* 36

6.4 *Toolbar* 37

6.5 *Infobar* 37

6.6 *Menubar* 38

6.7 *Add File Panel* 41

6.8 *Search Window* 42

6.9 *Archive Info Window* 45

6.10 *Comments Window* 48

6.11 *Invoking Tracker Add* 49

6.12 *QuickCreate window* 50
6.13 *File Splitter* 51
6.14 *File Joiner* 53
6.15 *Preferences window* 55

8. Results **65**
8.1 *Archive Window* 65
8.2 *Search Window* 66
8.3 *File Splitter* 66

List of Acronyms

Abbreviation	Expansion
API	Application Programming Interface
ARK	Archiver, add-on.
BFS	BeOS File System
CRC	Cyclic Redundancy Check
ELF	Executable and Link Format
FTP	File Transfer Protocol
GUI	Graphical User Interface
UTF	Unicode Type Face

Abstract

This project entitled Beezer is a file archiver program designed to handle many different archive formats under one uniform and consistent interface. In many ways, Beezer is similar to the commercial program - *WinZip* which has evolved over the years as the *de facto* standard in file archiving. Beezer is a program that facilitates the archival and compression of digital data. Beezer is a GUI-based archive management system replete with all the bells and whistles of any modern archiving tool.

The Beezer project has progressively evolved as a comprehensive tool that in many ways even exceeds the utility of other commercially available archivers. Beezer was initially designed and developed for the Be operating system (BeOS). Later Beezer was licensed by yellowTAB and was then redesigned with multi-language support for the commercial operating system - Zeta. Beezer was written from scratch using C/C++ and uses a dynamic add-ons based system for supporting different archive formats. This makes Beezer completely independent of the archive formats it supports and hence allows addition and upgrading of formats extremely simple, fast and without any further modifications to Beezer. Beezer makes extensive use of multi-threading to achieve higher levels of responsiveness by utilising the operating system's fine grain switching.

Chapter 1

Introduction

Beezer is a full fledged archive management system designed and developed for the BeOS. Beezer was developed from scratch using C++. Beezer makes full use of several technical aspects of the operating system from pervasive multi-threading, to instrinsic locale support. The entire Beezer code spans several thousand lines and like anyother original software work is protected under the Indian Copyright (Amendment) Act 1999.

Beezer was initially designed and developed for the Be operating system (BeOS). During the course of its development Beezer was redesigned with multi-language support for the commercial operating system - Zeta. Though Beezer is available for more than one platform its code is semi-platform independent.

Beezer is a heavily multi-threaded program designed in such a way that the interface remains totally responsive and under the control of the user even during CPU intensive operations. This is also partly due to the way Zeta performs multi-threading. The code was written in such a way so as to grant more control to the end user such as enabling cancelling of operations that are in progress.

Project Beezer also has pre-built tools for miscellaneous file management functions such as splitting and joining of files. There is also full support for drag and drop input such as extracting files from and adding files to archives.

1.1 Problem Definition

The problem of archival and compression of data is an age old one. Ever since the inception of computers, people started to store more and more information. In this digital age, the need for file compression still lives on even with the advent of huge capacity hard-disks. In this problem definition section, we define the processes that are part and parcel of this project.

1.1.1 Archival

Archival in this context refers to the clubbing of several files into a possibly larger file, stored in such a manner that individual files from the archive can be retrieved. Archiving has existed for several years and more recently has been coupled with compression to achieve the effect of compressing and storing many files into a single file called an archive.

A file archiver combines a number of files together into one archive file, or a series of archive files, for easier transportation or storage. Many file archivers use lossless data compression in order to reduce the archive's size.

The most basic archivers just take a list of files and concatenate their contents sequentially into the archive. In addition the archive must also contain some information about at least the names and lengths of the originals, so that proper reconstruction is possible. Most archivers also store meta-data about a file that the operating system provides, such as timestamps, ownership and access control.

The process of making an archive file is called archiving or packing. Reconstructing the original files from the archive is termed unarchiving, unpacking or extracting.

Ubiquitous amongst Unix and compatible operating systems is the tar file format (”tape archive”). Originally intended for transferring files to and from tape, it is still used on disk-based storage to combine files before they are compressed. Some other, less popular Unix oriented formats include ar and shar.

1.1.2 Compression

In computer science and information theory, data compression or squeezing is the process of encoding information using fewer bits (or other information-bearing units) than an unencoded representation would use through use of specific encoding schemes. For example, this page could be encoded with fewer bits if we accept the convention that the word ”compression” be encoded as ”comp”. One popular instance of compression that many computer users are familiar with is the zip file format, which, as well as providing compression, acts as an archiver, storing many files in a single output file.

Compression is important because it helps reduce the consumption of expensive resources, such as disk space or connection bandwidth. However, compression requires information processing power, which can also be expensive. The design of data compression schemes therefore involves trade-offs between various factors including compression capability, any amount of introduced distortion, computational resource requirements, and often other considerations as well.

1.2 Proposed Solution

This section discusses the objectives of project Beezer. It highlights the basic objectives of the program's functionality and also touches upon the technical aims of the program that are not easily perceived by the casual user.

1.2.1 Functional Objectives

- To provide the user with maximum utility, flexibility and ease of use.
- To provide support for as many archive formats as possible
- To build a stable program that remains responsive at all times
- Allow customisation of the program as much as possible
- To make Beezer as seamlessly integrated with the operating system as possible. This involves intricate interaction with standard third party programs such as the *Tracker* and the *Deskbar* (these are similar to the Windows MyComputer And Taskbar)
- To provide complete set of archive management features
- To allow users to test the integration of an archive
- To provide a powerful search function
- To provide on-the-fly language support

1.2.2 Technical Objectives

Apart from the aforementioned general objectives, project Beezer aims at some of the following technical goals:

- To utilize the operating system's unique add-on API, one add-on for each archive format
- Make Beezer as memory efficient as possible by implementing thread-safe global caches of data (such as images) that are safely shared across all windows of the application
- Provide super fast file lookup and archive loading using specially optimised Hash table algorithms, this involves using intelligent caches of previously looked up hash items
- Get across the hurdles of incompatibility among each archive format in such a way that it is possible to mimic native features of archive formats

1.3 Benefits & Features

This section gives a run-down of Beezer's most useful features:

- Opens archives without extracting them
- Displays archives in a neat and easy to use hierarchial tree-view, this feature is not there even in commercial archives such as *WinZip*
- Supports all major archive formats such as: zip, tar, gzip, bzip2, tar.gzip, tar.bzip2, lha (lzh), rar, arj and 7zip
- Can work with multiple archives at the same time
- Fully integrated with the *Tracker* making it seem as though it is an integral part of the operating system itself
- Stores configuration settings for each archive
- Has an elegant user interface with clearly defined functions which is so easy that even a novice can use all the features with little or no extra help

- Supports all major archive operations including: Add, Extract, Delete and lots more
- Has several tools for archival related purposes such as a File Splitter which debuted in Beezer long before it was made available for *WinZip*
- Have any number of favourite paths for regular/quick extraction
- Since add-on based, more formats can be added in the future
- User adjustable preferences and settings

These are only some of the main features of Beezer. Several other features exists, and they are not enlisted here but they will be discussed in detail in the following chapters.

Chapter 2

Competitive Analysis

This section primarily discusses how Beezer matches up with similar products available on other platforms.

My initial objective for creating something like Beezer was to experiment with a relatively new operating system and its features; to create a file archiver program that would be fast, lightweight and as easy to use as possible. Neither BeOS nor Zeta had a program that incorporated the features I wanted.

Though Windows counterparts were powerful, they were often counter-intuitive and lacked the responsiveness that is typical of a BeOS program. These were the reasons that reinforced my intention to create Beezer.

Beezer takes the strong points and features of a huge number of archivers and combines them with the speed and responsiveness of the operating system - BeOS and Zeta.

2.1 Competitors

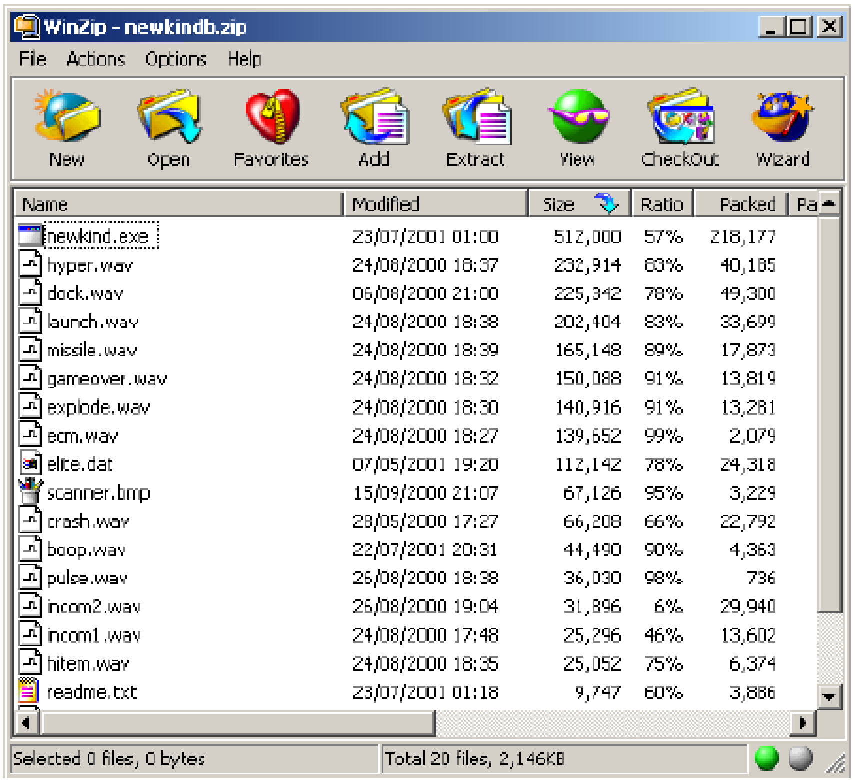
There are thousands of file archivers out there, but some are more similar to Beezer than others. This includes similarity in look-and-feel aspects and not necessarily similarities in design. This is because Beezer's design is radically different from most of its competitors.

WinZip is taken as the comparitive benchmarks for the Windows platform. *Ark* is taken as the comparitive benchmark for Linux. No similar programs exist for Zeta and hence no competitive analysis is possible for Zeta.

2.1.1 WinZip

WinZip is a commercial file archiver designed for Microsoft Windows 9x/NT/XP. Developed by WinZip Computing, formerly known as Nico Mak Computing it uses PKZIP format and is also capable of handling other archive formats as well.

WinZip (figure 2.1) began life around the early 1990s as a shareware GUI frontend for PKZIP. Somewhere around 1996 the creators of *WinZip* incorporated compression code from the Info-ZIP project, thus dispensing with the need for the PKZIP executable to be present.



Name	Modified	Size	Ratio	Packed	Pa
newkind.exe	23/07/2001 01:00	512,000	57%	218,177	
hyper.wav	24/08/2000 18:37	232,914	83%	40,185	
dock.wav	06/08/2000 21:00	225,342	78%	49,300	
launch.wav	24/08/2000 18:38	202,404	83%	33,699	
missile.wav	24/08/2000 18:39	165,148	89%	17,873	
gameover.wav	24/08/2000 18:32	150,088	91%	13,819	
explode.wav	24/08/2000 18:30	140,916	91%	13,281	
ecm.wav	24/08/2000 18:27	139,652	99%	2,079	
elite.dat	07/05/2001 19:20	112,142	78%	24,318	
scanner.bmp	15/09/2000 21:07	67,126	95%	3,229	
crash.wav	28/05/2000 17:27	66,208	66%	22,792	
boop.wav	22/07/2001 20:31	44,490	90%	4,363	
pulse.wav	26/08/2000 18:38	36,030	98%	736	
incom2.wav	26/08/2000 19:04	31,896	6%	29,940	
incom1.wav	24/08/2000 17:48	25,296	46%	13,602	
hitem.wav	24/08/2000 18:35	25,052	75%	6,374	
readme.txt	23/07/2001 01:18	9,747	60%	3,886	

Selected 0 files, 0 bytes | Total 20 files, 2,146KB

Figure 2.1: *WinZip*

WinZip has gained more commercial success than any other archiver of its kind. What makes things interesting is that its perfectly possible to create file archivers that equals or betters *WinZip* by combining and following proper design and coding techniques.

WinZip is also being actively developed by Nico Mak Computing, but comparing Beezer with a popular archiver such as *WinZip* would provide a more clear picture of

Beezer's capabilities.

Strengths Against WinZip

This subsection lists in a concise manner how Beezer scores over *WinZip*, it must be noted that comparisons is made with Beezer 0.09 and *WinZip* 9.0.

- Supports more archive formats than *WinZip* such as Rar, 7zip etc.
- Displays files and folders in a completely functional tree view format.
- Is capable of adding files inside any folder of an existing archive.
- Comes with support tools such as File Splitter and Self-Joining executables.
- Has full text search (with pattern searching) for filenames within the archive.
- Ability to work with several folders simultaneously.
- Ability to work with several archives simultaneously.
- Remembers extract locations, and also makes dynamic extract locations based on the archive name.
- Shows more file statistics than *WinZip* such as number of individual files, folders.
- Supports drag 'n drop extraction with and without extracting with the full path.
- Can remember interface and archive settings for each archive (if in BeFS file system).
- Allows users to fully control how existing files inside an archive are replaced when new files with the same names are added.
- Can cancel operations in progress.
- Keeps a log of all actions performed for the further reference.

Weaknesses Against WinZip

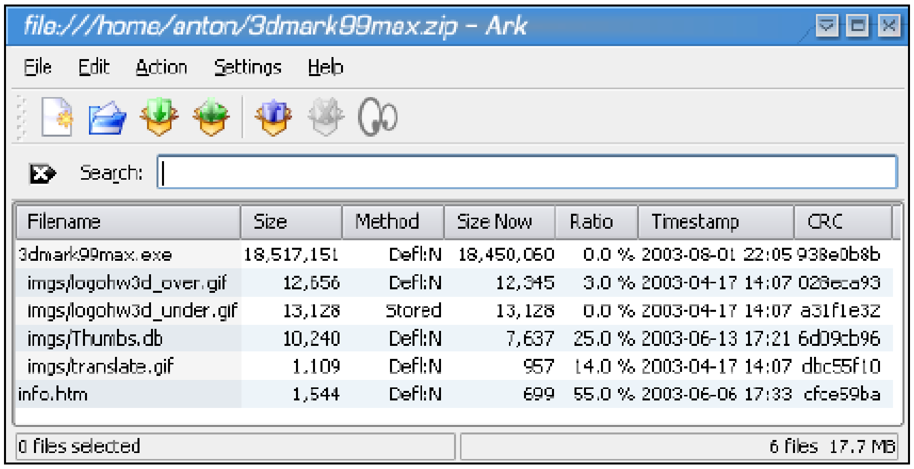
- No Wizard mode in Beezer.
- No self-extractor support.

Most of the weaknesses of Beezer cannot be overcome due to technical constraints of the nature of the program, the operating system and interaction therein.

2.1.2 Ark

Ark is a wonderfully designed archiver program for Linux. In reality it's more similar to Beezer than any other archiver that has inspired the making of Beezer. Ark like Beezer interfaces seamlessly between various back-end resources to provide a native feel of handling the archive format.

From a practical standpoint, the interfacing and direct native support (as the case of *WinZip*) is pretty much the same. The net effect for the end user is that he/she is able to work with archives the way they want to and a casual user will not be able to tell how fundamentally different the two types of archivers are.



Filename	Size	Method	Size Now	Ratio	Timestamp	CRC
3dmark99max.exe	18,517,151	Defl:N	18,450,060	0.0 %	2003-08-01 22:05	938e0b8b
imgs/logohw3d_over.gif	12,656	Defl:N	12,345	3.0 %	2003-04-17 14:07	028eca93
imgs/logohw3d_under.gif	13,128	Stored	13,128	0.0 %	2003-04-17 14:07	a31f1e32
imgs/Thumbs.db	10,240	Defl:N	7,637	25.0 %	2003-06-13 17:21	6d09cb96
imgs/translate.gif	1,109	Defl:N	957	14.0 %	2003-04-17 14:07	dbc55f10
info.htm	1,544	Defl:N	699	55.0 %	2003-06-06 17:33	cfce59ba

0 files selected

6 files 17.7 MB

Figure 2.2: *Ark*

Ark (figure 2.2) is stylistically similar to *WinZip* but fundamentally is very different in the way it handles archives as explained above.

Ark is very popular in the Linux world and is probably the most widely used archiver program in Linux. Its open source nature, free to use license and ease of use in its interface has heavily influenced Beezer in more ways than one.

Beezer also addresses some of the short comings of *Ark* such as handling passwords, comments, adding file splitting abilities among others. This makes Beezer a comprehensive application that fulfills most archiving needs.

Strengths Against Ark

This subsection lists in a concise manner how Beezer scores over *Ark*, it must be noted that comparisons is made with Beezer 0.09 and *Ark* 2.0.1

- Supports some archive formats not supported by *Ark* such as 7zip.
- Displays files and folders in a completely functional tree view format.
- Is capable of adding files and folders to any sub-directory in the archive.
- Can cancel operations in progress.
- Remembers extract locations, and also makes dynamic extract locations based on the archive name.
- Can remember interface and archive settings for each archive (if in BeFS file system).
- Keeps a log of all actions performed for the further reference.

Weaknesses Against Ark

Here are the main shortcomings of Beezer compared to *Ark*.

- Archive loading times are slightly faster in *Ark*.
- Supports creation of pure gzip and bzip2 archives, containing only a single file, while Beezer only supports tar.gz and tar.bzp2.

Chapter 3

System Requirements

This chapter covers all things needed to get Beezer working on a system. This includes software and hardware requirements which are a bare minimum as Beezer is a very lightweight program.

3.1 Hardware

Hardware requirements of Beezer are not demanding, but in any case here's a list of hardware specifications that a system must meet inorder to run Beezer properly...

- CPU: An Intel, AMD or x86 compatible computer
- RAM: 32 MB of RAM
- Hard Disk: 5 MB of free hard-disk space on a BeFS partition*
- Display: 24-bit colour display for undistorted image reproduction

* Some operations, such as addition of files to the archive, may require making copies of files or folders. For this sufficient amount of free space would be required.

3.2 Software

The minimum software requirements of Beezer are as follows:

- BeOS R5, Zeta R1 or compatible operating system
- *OpenTracker* program is highly recommended for faster navigation
- Installed binaries of archivers such as zip, gzip etc.

Note that although Beezer will execute and function properly under both BeOS and Zeta, this has been tested more in Zeta and its stability under R5 may not be on par.

Also note that Beezer has not been tested on all flavours of BeOS such as Dan0, BeOS Max, PhOS, BeOS Wind among others. If these systems don't break the basics of R5 there is a fair chance Beezer will work on them.

Most of these variations of R5 don't differ in their system library compatibility and hence there is a fair chance Beezer will work properly but support is not offered for these systems. And it must be noted that only Beezer for Zeta has multi-language support.

Chapter 4

Required Resources

This chapter discusses the software tools used to create Beezer. Beezer was made using a plethora of tools that most of which are available for free for BeOS and Zeta.

4.1 Build Tools

Build tools available in Zeta are few, but powerful and capable ones. The primary tool used to build Beezer is the *BeIDE* which. The primary tools needed to build Beezer need not be separately installed as all the tools needed come pre-installed with any installation of Zeta.

4.1.1 BeIDE

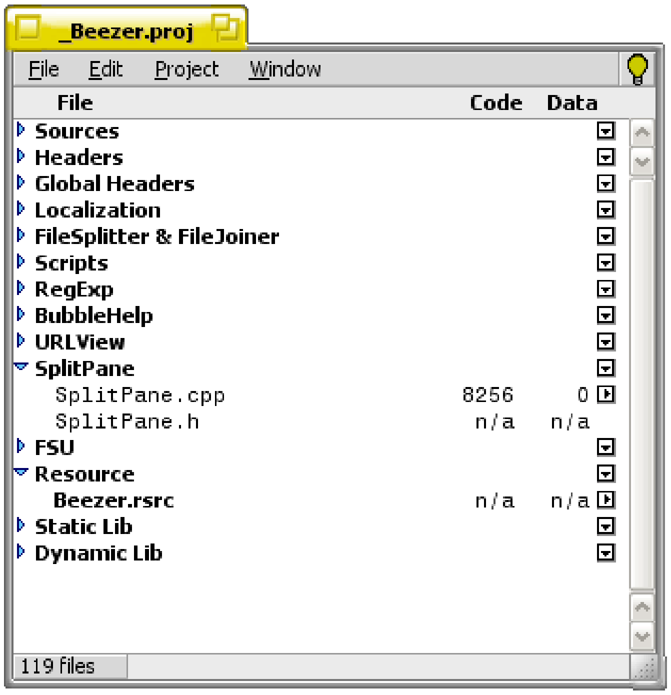
The *BeIDE* is a premier integrated development environment (IDE). *BeIDE* was developed by Metrowerks Corp. for BeOS and works also in Zeta, and hence is sometimes called *Metrowerks IDE*.

It comprises of a comprehensive code editor, supporting syntax highlighting for C/C++ and Java. One of the main features of *BeIDE* is that it has a full-fledged project manager. Projects in *BeIDE* are containers for all the source code for a program (.cpp, .h etc.) inclusive of all the settings and options required during the compilation of the project. *BeIDE* saves project files under the extension of `.proj`.

These "proj" files are files that contain all the filenames (paths) to all the files included in the project. It also contains the settings for the project such as compiler options, linker options etc.

The Project window allows users to add, delete files to the project. These can be source files, resources, dynamic libraries or static libraries and even Bash scripts.

BeIDE supports using custom files (with custom MIME types). They can also be added to the project like any other file. Put inorder to tell *BeIDE* what action it must take (such as compile, link, resource, ignore, etc.) the appropriate filetype and action must be associated. This can be done by choosing `Edit` –> `Project Settings` and "Target" panel. Here, the filetype needs to be added, then the action that must be taken is to be specified. If this is not done, *BeIDE* simply ignores the file.

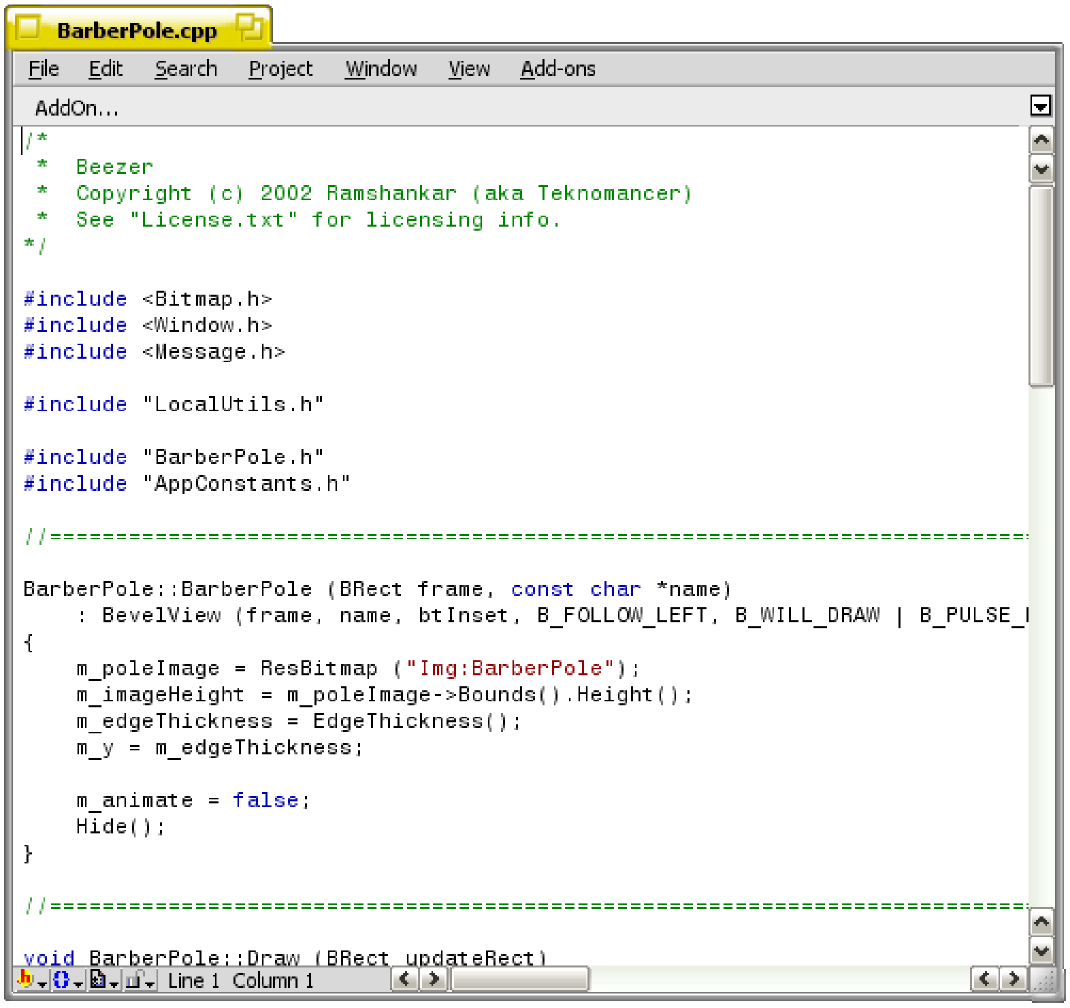


File	Code	Data
Sources		
Headers		
Global Headers		
Localization		
FileSplitter & FileJoiner		
Scripts		
RegExp		
BubbleHelp		
URLView		
SplitPane		
SplitPane.cpp	8256	0
SplitPane.h	n/a	n/a
FSU		
Resource		
Beezer.rsrc	n/a	n/a
Static Lib		
Dynamic Lib		

Figure 4.1: *BeIDE: Project window*

The *BeIDE* code window supports fast navigation using shortcuts, supports syntax highlight, bracket matching (that automatically finds the opening brace when the ending brace is typed), automatic indentation using `TAB` and many other features that makes it perfectly suited for writing code.

The code window also has some very useful and time saving features such as the functon popup. This is a popup menu that when clicked list all the functions in the current source file. Choosing one item from the popup, moves the cursor to that function. This feature is especially handy when writing huge source files with hundreds of functions.



```
/*
 *  Beezer
 *  Copyright (c) 2002 Ramshankar (aka Teknomancer)
 *  See "License.txt" for licensing info.
*/

#include <Bitmap.h>
#include <Window.h>
#include <Message.h>

#include "LocalUtils.h"

#include "BarberPole.h"
#include "AppConstants.h"

//=============================================================================

BarberPole::BarberPole (BRect frame, const char *name)
    : BevelView (frame, name, btInset, B_FOLLOW_LEFT, B_WILL_DRAW | B_PULSE_
{
    m_poleImage = ResBitmap ("Img:BarberPole");
    m_imageHeight = m_poleImage->Bounds().Height();
    m_edgeThickness = EdgeThickness();
    m_y = m_edgeThickness;

    m_animate = false;
    Hide();
}

//=============================================================================

void BarberPole::Draw (BRect updateRect)
```

Figure 4.2: *BeIDE: Code window*

BeIDE also has the ability to save source code in a variety of line feed formats such as Windows, BeOS and MacOS.

The compiler that BeIDE uses is the GNU C Compiler, otherwise called GCC. This is the compiler used for compiling Beezer. The following sections discusses GCC in detail.

4.1.2 GCC

GNU C Compiler, shortly referred to as GCC, is an ISO standard (ISO/IEC 9899:1990) compiler that was created by the Free Software Foundation and has become the most widely used compiler today. It exists for a staggering range of operating systems and languages such as C/C++, Java, Fortran, Ada, Objective-C among others. Zeta and BeOS versions of *GCC* only supports C/C++.

Using GCC

GCC is basically a command-line program and isn't a graphical user interface. Thus, first and foremost it needs to be used from the command-line, such as a Terminal, Bash shell or Command-prompt.

C++ source files conventionally use one of the suffixes '.c', '.cc', '.cpp', '.c++', '.cp', or '.cxx' ; C++ header files often use '.hh' or '.h'; and preprocessed C++ files use the suffix '.ii'. *GCC* recognizes files with these names and compiles them as C++ programs even if the compiler is called the same way as for compiling C programs (usually with the name gcc).

A typical invocation of *GCC* would be as follows:

```
gcc filename.cpp -o exefile
```

That would compile, link and create the binary file named "exefile". The '-o' option is optional, and omitting it would produce the binary file with the suffix '.o', in this case "filename.o". *GCC* has strictly case-senstive options and thus '-o' and '-O' are two entirely different options.

Most of the time the programs would either be built using a makefile or an IDE interface rather than calling *GCC* directly because each time all the libraries need to be specified and doing so manually would be extremely tedious.

4.2 Test Tools

Test tools are programs or methods that were used in the software testing phase of Beezer.

4.2.1 Debugging Macros

One of the simplest, but very effective, convinient and time-saving methods of testing conditions, reachability, correctness in the code is to use Debugging Macros . These macros are conditionally executed using a flag, called the DEBUG flag.

This flag is a preprocessor flag that is defined whenever the Debugging Macros are needed to work. They are un-declared (or rather the definition is removed) when the final program (hopefully stable) is to be distributed to the clients or users, who need not see the debugging information being printed/called in general.

The DEBUG flag for *GCC* is specified as follows:

`-Wp,-DDEBUG`, for *BeIDE*
`CFLAGS := -DDEBUG`, for standard makefile

The above flag is specified in *BeIDE* in `Edit` –> `Project Settings` and under "More Compiler Options" panel. This allows the following macro to be compiled into the Beezer binary:

```
PRINT (("--control has reached till this point\n"));
```

PRINT and SERIAL_PRINT macros

These macros prints any message given to them as an argument. The arguments take the variable form of `printf()` and thus must be wrapped inside a second set of parenthesis as shown above.

The SERIAL_PRINT macro prints data to a serial port at the rate of 19,200 bits per second. Technically, the output is send to `/dev/ports/serial1` on all Intel machines.

The following is a sample piece of code that demonstrates the macro.

```
// Draw if the rectangle is really valid
m_cachedRect = newRect;
if (minX != maxX)
{
   Invalidate (BRect (minX - 1, newRect.top, maxX, maxY));
   PRINT (("control reaches here!\n"));
}
```

Note that it is necessary to explicity use "`\n`" at the end of each PRINT macro.

The advantage of using these message macros is that it need not be deleted throughout the code. Simply by removing the compiler option we make the compiler treat all the macros as comments. They will be ignored. Thus, this saves enormous amount of time and effort. They can again be switched ON by redeclaring the flag under compiler options.

IS_DEBUG_ENABLED and SET_DEBUG_ENABLED macros

These two macros are used to get and set whether debugging should be enabled or not. In otherwords, they are used to access the debug flag. But for these macros to be enabled the DEBUG preprocessor should be used, or similar flag using makefile or *BeIDE*.

DEBUGGER, TRESPASS AND ASSERT macros

These macros cause the program to enter the debugger with a crash and an error message. While the macros, DEBUGGER and TRESPASS always enter the debugger, the ASSERT enters the debugger only if the condition given as argument to it evalutes to `FALSE`.

The debugger that ships with *BeIDE* is called *bdb*, short for Be Debugger. This is a very powerful debugger and the following section covers some of its features.

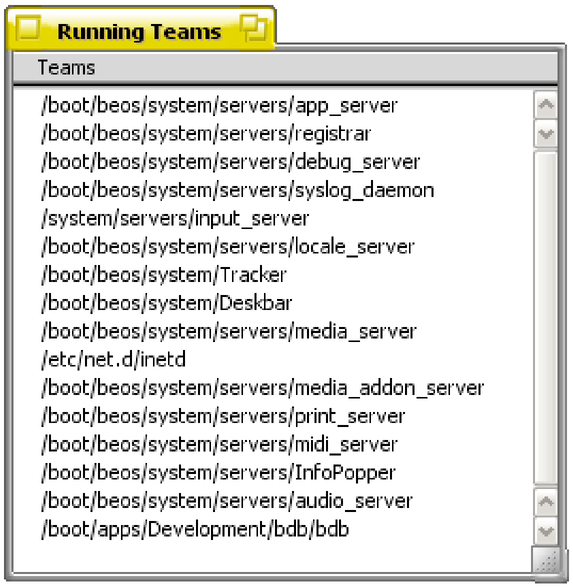
4.2.2 BeDebugger

The advantages of a source level debugger cannot be overstated. Just as it's advantageous to write code in high level languages like C/C++, Java, Pascal and others, it's also important to be able to evaluate the function and performance of code at a higher level. The *BeDebugger* – *bdb* for short – is the source level debugger for BeOS that fulfills this need. *bdb* provides programmers with the necessary tools for verifying program functionality.

The architecture of BeOS encourages multi-threading. In simple terms, a thread is a series of instructions to execute. Multi-threading allows for simultaneous execution of the instructions of more than one thread. Applications written in BeOS often have more than one thread. The most basic GUI (Graphical User Interface) programs have a minimum of two threads – one used by the `BApplication` object and another by the `BWindow` object.

An application's threads are organized into a team, which is similar to a process on other operating systems. The existence of teams and multiple threads makes it necessary to have a debugger that can debug multi-threaded applications. *bdb* provides access to team and thread information, as well as control over the current teams and threads running on the system. *bdb* can halt individual threads in a team, while other threads continue to execute independently or until they block because of the halted thread. *bdb* makes it possible to debug code in a multithreaded environment.

In order for *bdb* to function, the code must be compiled to include debugging information. In *BeIDE*, this can be done by choosing `Edit -> Project Settings`, then selecting "Generate debugging information" under the "Code Generation" options. The code will have to be fully recompiled after the change is made. Compiling with debugging information turned on causes a dramatic increase in the size of the output binaries.



```
/boot/beos/system/servers/app_server
/boot/beos/system/servers/registrar
/boot/beos/system/servers/debug_server
/boot/beos/system/servers/syslog_daemon
/system/servers/input_server
/boot/beos/system/servers/locale_server
/boot/beos/system/Tracker
/boot/beos/system/Deskbar
/boot/beos/system/servers/media_server
/etc/net.d/inetd
/boot/beos/system/servers/media_addon_server
/boot/beos/system/servers/print_server
/boot/beos/system/servers/midi_server
/boot/beos/system/servers/InfoPopper
/boot/beos/system/servers/audio_server
/boot/apps/Development/bdb/bdb
```

Figure 4.3: *BeDebugger: Running Teams window*

There are three window types in *bdb*. First is the Running Teams window (figure 4.3), which shows all teams running on the system. Double-clicking on a listed team activates *bdb* and throws that team into the debugger.

The second window type is a Team window (see figure 4.4 on page 21), which lists all threads that belong to the team, as well as all source code files used in the compilation process.

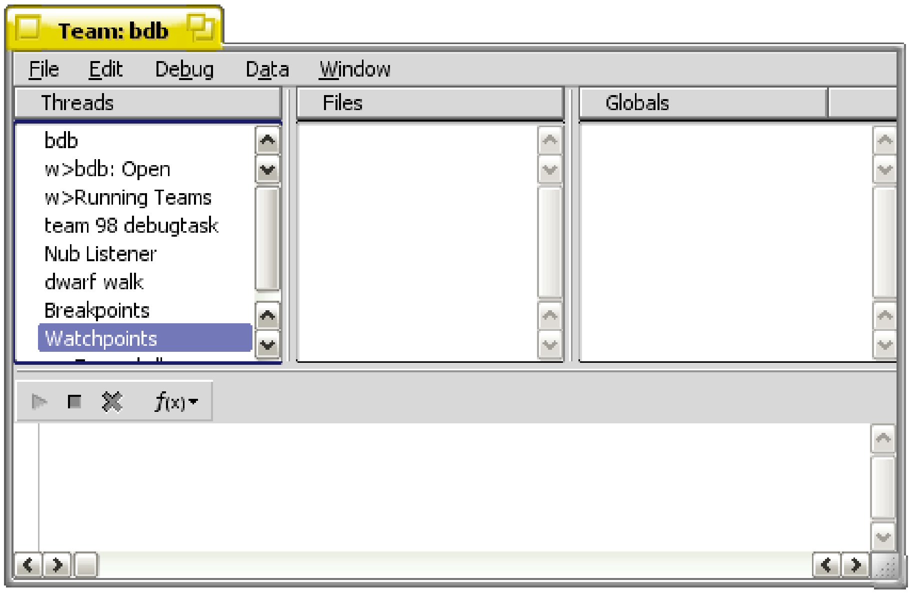


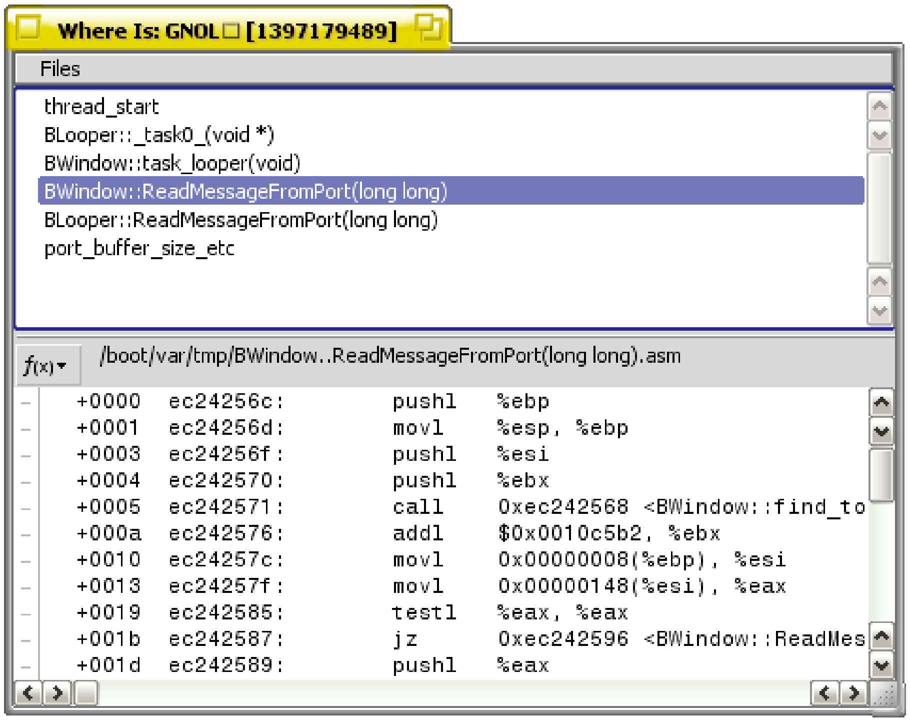
Figure 4.4: *BeDebugger: Team window*

The final window type is the Thread window (see figure 4.5 on page 22), shown when a thread stops executing. The cause could be an error or insertion of a breakpoint that halts program execution. The Thread window shows the current state of the stack and allows programmers to check the values of variables at all levels of the stack. If a line of code causes a segment violation or a breakpoint is reached, the debugger stops on that line of code.

The debugger can easily be invoked from a program by calling `debugger (char*)` function which is declared in `OS.h`.

There are more useful things that are present in *bdb*. One of the handiest features of the debugger is its ability to present the status and value of variables at the point where thread execution has halted. *bdb* can show fixed variables such as integers, floats, and pointer addresses. It also goes a step further by giving the programmer, through the use of add-ons, the ability to view other types of data in intelligible forms (i.e., viewing `Bitmap` and string as pictures and text).

Debugging is an integral part of programming, and having the right tools is essential for programmers. *bdb* is a very valuable debugging tool. Becoming familiar with its



```
thread_start
BLooper::_task0_(void *)
BWindow::task_looper(void)
BWindow::ReadMessageFromPort(long long)
BLooper::ReadMessageFromPort(long long)
port_buffer_size_etc
```

```
/boot/var/tmp/BWindow..ReadMessageFromPort(long long).asm
-   +0000  ec24256c:      pushl   %ebp
-   +0001  ec24256d:      movl    %esp, %ebp
-   +0003  ec24256f:      pushl   %esi
-   +0004  ec242570:      pushl   %ebx
-   +0005  ec242571:      call    0xec242568 <BWindow::find_to
-   +000a  ec242576:      addl    $0x0010c5b2, %ebx
-   +0010  ec24257c:      movl    0x00000008(%ebp), %esi
-   +0013  ec24257f:      movl    0x00000148(%esi), %eax
-   +0019  ec242585:      testl   %eax, %eax
-   +001b  ec242587:      jz      0xec242596 <BWindow::ReadMes
-   +001d  ec242589:      pushl   %eax
```

Figure 4.5: *BeDebugger: Thread window*

abilities can save hours of debugging time.

4.2.3 Other Debugging Methods

Other debugging methods have also been employed during the development and test phase of Beezer. These methods are miscellaneous methods that are not formally documented during the test phase, in the form of their implementation or results. However, these methods are mentioned here as they proved useful, at times critical, in tracing bugs in the logic, finding validation loopholes and performance lags in the program. One such method used in Beezer is the timing method.

Timing method

This method is used to estimate roughly the amount of execution time taken by a procedure, function or any piece of code in general. This is done in BeOS by using its very own `BStopWatch`.

The `BStopWatch` class is a debugging tool designed to time the execution of portions of code. When a `BStopWatch` object is constructed, it starts an internal timer. When it's deleted, it stops the timer and prints the elapsed time to standard out in this format:

```
StopWatch "name":  f usecs.
```

Where `name` is the name that is given to the objcet when its constructed, and `f` is the elapsed time in microseconds.

`Suspend()`, `Resume()`, and `Reset()` of the internal timer is possible.

The current elapsed time without stopping the timer can be retrieved by using the `ElapsedTime()` function. "Lap points" can be recorded and printed out at the end of the run by using the `Lap()` function.

It's possible to declare `BStopWatch` on the stack and in this case, when the stack pop operation occurs (when the data/variables/objects of the function are popped out of the function stack) the object would get destroyed and would print out the elapsed time.

`BStopWatch` objects are useful for getting an idea of where the cycles are going, but it shouldn't be relied on for painfully accurate measures of time, like in the range of a few microseconds.

4.2.4 Stress Tests

Beezer was stress tested in numerous different ways. From overloading many memory locations by manual launching heavy-duty applications thereby restricting Beezer to use minimal memory to comparing speeds of operations right down to the last millisecond with its competitors, many such techniques were employed.

The most often used stress test was to load upto 50 archive files at once. Another one was to load archives whose contents varied from a few hundred to a several thousand files. Beezer faired well in most tests. Beezer also remains one of the most memory efficient archive programs.

4.3 Documentation Tools

Documentation is an essential part of developing good software. No matter how easy or difficult a program is to use it needs to be documented. This documentation is both in the code level as well as the end-user documentation and everything in between. Documentation in the code level was done only using comments. Comments are sparingly used only explaining the code when required or when things are done very unconventionally.

4.3.1 CoolCat

HTML (Hyper-Text Markup Language) was used to create Beezer's electronic documentation explaining the program's features and how to use it, providing tips and tricks etc.This documentation was created using the html editor called *CoolCat*. It's a BeOS native HTML editor that supports HTML syntax highlighting and quick navigation using shortcuts. It also has support for XML. One of the main plus points about using *CoolCat* is that its preview function that automatically launches and opens the current document through the default web browser application.

4.3.2 LaTeX

LaTeX (pronounced "lay-tec") is a high-quality typesetting system, with features designed for the production of technical and scientific documentation. LaTeX is the *de-facto* standard for the communication and publication of scientific documents.

LaTeX was developed during 1985 based on the typesetting language TeX which was developed by *Donald E. Knuth*.

LaTeX is not a word processor! Instead, it encourages authors not to worry too much about the appearance of their documents, but to concentrate on getting the right content. LaTeX takes care of producing table of contents, index, image placement (automatic), referencs such as tables, figures, pages and also takes care of footnotes, page numbering, section and subsection numbering and references to them.

LaTeX contains features for:

- Typesetting journal articles, technical reports, books, and slide presentations.

- Control over large documents containing sectioning, cross-references, tables and figures.
- Typesetting of complex mathematical formulae.
- Advanced typesetting of mathematics with AMS-LaTeX .
- Automatic generation of bibliographies and indexes.
- Multi-lingual typesetting.
- Inclusion of artwork, and process or spot colour.
- Using PostScript or Metafont fonts.

A short example of a template for a document in LaTeX follows:

```
\documentclass[a4paper,10pt]{book}
   %This is a comment
   \author{AuthorName}
   \title{Book Name}
\begin{document}
   \maketitle
   \section{Introduction}
   Hello !! This is what I want to do!
\end{document}
```

This document was created fully by using LaTeX . LaTeX is ideally suited for creating letters, reports, memos, and especially documentation in the form of extremely professional books. LaTeX is capable of outputting documents in the form of HTML, Adobe PDF, DVI as well as many others. Because of this raw production factor, LaTeX documents needs to be compiled. Yes, it needs to be compiled similar to a C/C++ program using tools like "pdflatex", "latex2html" etc.

LaTeX is a huge typesetting language in itself and would require a whole range of books to explain all its features. Thus the scope of explaining LaTeX and how to use it to create documents is beyond the scope of this documentation.

Chapter 5

Design & Implementation

This chapter discusses how Beezer was designed and implemented. Beezer's design consists of several intricately coupled components, most of the time even to the extent of breaking conventional coding methods for the sake of speed or for other reasons.

This chapter is subdivided in the following manner.

1. **Design goals** This section explains the techincal design goals of Beezer and some problems that were faced during the construction of Beezer and how it was overcome.

2. **Concept diagram** This section deals with the conceptual diagram of how Beezer works. It explains the various components and their roles, their independence and dependence on other components and gives an overall picture of the structure behind Beezer's construction.

5.1 Design Goals

Beezer has very simple and clear design goals to achieve. They are enlisted below...

- To design a program that manages archive formats of all kinds.
- To design a program that is customizable enough to suit personal needs.
- To design a program that is fast in its operation, and at all points of time during its execution remains fully responsive.
- The responsiveness includes giving the user the ability to cancel operations that are in progress and do so gracefully.

- To utilize the operating system's multi-threading abilities to the fullest extent possible but only where applicable and necessary.
- To design a program that is quick and easy to use but at the same time is powerful enough, this involves intelligent placement of options and settings that are available yet not obstrusive to the user, reduce the number of pop-up messages that can be annoying and replace them with consolidated messages and such...
- To design a program that is scaleable across a range of computer configurations both modern and old, this includes utilizing minimum possible memory using techniques such as caching, use-and-throw memory blocks etc.
- To design the program as error free and stable as possible.

5.2 Concept Diagram

The concept diagram has been greatly simplified to only highlight the overall framework of Beezer.

The simple block diagram consists of blocks that loosely represent a logical set of components which maybe a class, a collection of classes or even partial classes that overlap physical placement; arrows that represent flow of data or control which maybe in the form of signals, messages and parameters.

5.2.1 Interface

The interface components of Beezer consists of mostly graphical interface objects that allow the user to interact with the underlying components that actually perform the archiver operations. Its a link between the user level input server and the lower components of the system. In other words it handles drawing an interface, getting input from the user (via the operating system's input server) and displaying output.

Though this is the most basic work of the interface components it gets exponentially complex as input and output are done in a totally multi-threaded fashion, as input and output seamless flow between synchronous and asynchronous modes, as inputs require feedback from the backend components for consistent output and so on.

In total there are about 120 interface components in Beezer and most of the time the interaction among them is consistent. Due to the complex nature of the system, at times the interaction among components may not be straight forward for the sake of speed or memory efficiency.

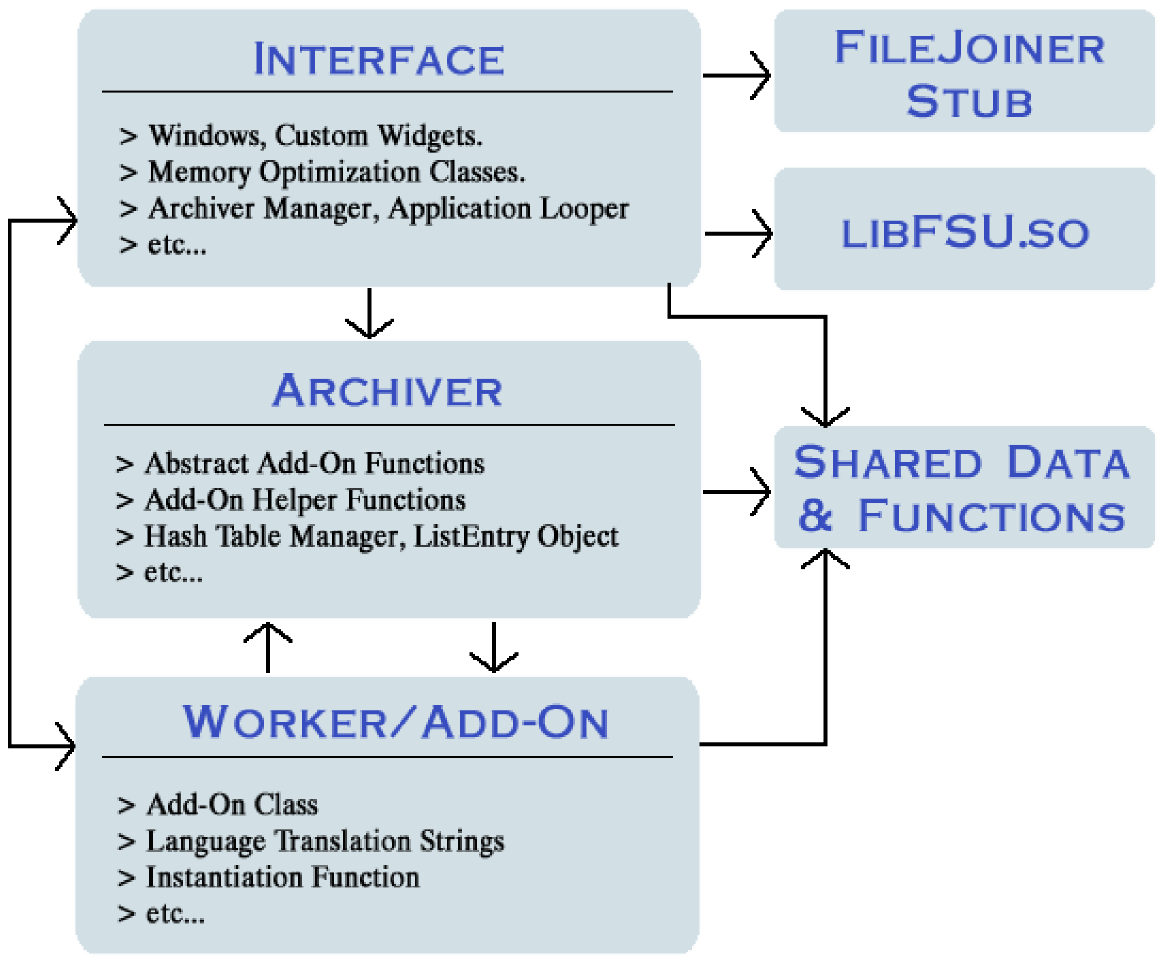

INTERFACE
> Windows, Custom Widgets.
> Memory Optimization Classes.
> Archiver Manager, Application Looper
> etc...
FILEJOINER STUB
LIBFSU.SO
ARCHIVER
> Abstract Add-On Functions
> Add-On Helper Functions
> Hash Table Manager, ListEntry Object
> etc...
SHARED DATA & FUNCTIONS
WORKER/ADD-ON
> Add-On Class
> Language Translation Strings
> Instantiation Function
> etc...

Figure 5.1: *Concept Diagram*

5.2.2 Archiver

The `Archiver` a collection of classes. The main class is a partially abstract class that has abstract and non-abstract functions, some of which are overrided when needed, has helper functions that can be invoked by inherited classes when needed. The archiver apart from helper functions also consists of helper classes such as `PipeMgr`, a class for managing pipes.

The Archiver also consists of an optimized hash table class, the pipe manager, `ListEntry` and some other classes that serve various purposes. Apart from the internal caching that is performed by the `HashTable` class, the Archiver also implements a cache routine to speeden up some hash table related operations. The Archiver acts as a link between the interface components and the add-ons and as a base class from which all add-ons are derived.

One of the most important functions of the archiver components are to manage the set of files inside the archive using a hash table and keeping the actual archive file perfectly synchronised with the in-memory contents. This is also the most complex part of Beezer. Other functions of the Archiver includes making sure wildcards are ignored or applied when needed, constructing accurate time values of files and so on.

The Archiver also manages on-demand temporary directories; constructs bitmaps based on the type of files in the archive and similar functions. Archiver presents various abstract functions, and some implemented placeholder functions that most add-ons use to simplify its task. The idea here is to make the add-ons as simple and elegant as possible making the Archiver bear the brunt of the burden.

5.2.3 Add-Ons

Add-ons are derived from the `Archiver` class. It performs the interaction with the backend binary and translates information into a uniform format as represented by the `Archiver`. This allows Beezer to handle various formats as though they are all uniform in their features. Each add-on supports one archive format and has hook functions for various operations such as addition, deletion etc. and becomes more complex when some operations are not appropriate or available for a format in which case return-codes or new verifier functions are used by the interface components to check for availability.

Each add-on also has an instantiation function which loads is a polymorphic function that loads the appropriate derived class into a base class pointer ready for use for the interface components. The add-ons make use of various archiver functions and objects such as `PipeMgr` to communicate with the worker binaries while performing the required tasks.

Each add-on works independently with its backend programs. It is responsible for throwing up errors when needed, such as failure to find necessary programs and so on. The add-on also dynamically creates and destroys a `BMenu` object consisting of various option specific to that add-on format. Add-ons also override all necessary functions such as to indicate the format supports password protection, support for adding files, support for adding empty directories and so on.

5.2.4 Other components

Beezer has several more components apart from the aforementioned ones. These include the File Splitter, File Joiner, File Joiner Stub, libFSU.so (File System Utilities) etc. Some of these components are tightly coupled with Beezer while some are more generic and with a bit of modification can even be adapted for use with other programs.

5.3 FileType rules

Beezer uses mimetype corrections to rectify incorrect mimetypes for archives that users try to open using Beezer. It also uses these rules to open archives that resides in file systems that don't support mime (such as FAT32), in such cases the file extension would be critical. The mimetype correction rules are necessary for opening archives with incorrect mimetypes/extensions (often the case when archives are downloaded from the Internet).

In general, users shouldn't alter the mimetype corrections file. The file named `_bzr_rules.txt` in Beezer's `settings` folder is the mimetype corrections file. It is a plain text file whose format is explained within the file itself.

Should anything happen to this file (deleted/renamed/moved), Beezer will, for every archive that is opened, re-assign the mimetype which is not an efficient thing to do. Hence it is advised users don't do anything to this file. Users can however, use this file to add new mimetypes that the users' system has (and provided Beezer can open them), remove unwanted mimetypes etc.

5.4 Compile Timestamp

The file `settings/_bzr_ctstamp.txt` contains the date & time of when the copy of Beezer was compiled. It is a generate file and should not be modified!

This is often useful in knowing how Beezer is, and also useful for debugging purposes (in situations when even the app refuses to start). Users can also see this date & time in the About box. This file is internally used by Beezer and in general you should not be tampered with.

5.5 Add-ons & Workers

Add-ons, also referred to as archivers are independent modules that handle various archive types. Add-ons are named "ark_zip", "ark_tar" etc. Additional add-ons maybe written to support more archive formats.

Workers in Beezer are those command-line programs that physically manipulate the archives, such as zip, unzip, tar etc. Workers are placed in the `workers` folder and in general must not be removed or renamed. By default they are all symbolic links that point to the location where the actual command-line program is suppose to reside. If users find broken links in this folder that means, either they do not have the command-line program installed or you have it installed in someother location.

To fix this, the user must either install the command-line program in the place where the link points to, or replace the link itself by a copy of the program. The reason why workers are made as symbolic links is because, in case a future version of the worker binary breaks compatibility with Beezer, users can still use the older version by replacing the symbolic link with the older version file.

Chapter 6

Usage Guidelines

This chapter is a walkthrough (instructional) manual that explains how to use Beezer from user interface, to the more advanced concepts like editing settings files etc.

6.1 Installation

Beezer is distributed as a *SoftwareValet* package. Just locate the file `Beezer.pkg` and double-click it. Now Beezer's installer can be seen. Follow the instructions on screen and complete the installation. Users can move Beezer to another location even after installation, as all of its application and data files are self-contained in the `Beezer` folder. Users can move the folder to any BeOS File System as long as the data files are kept within it and no alterations are done to the directory structure.

Important note: No matter to which folder Beezer is installed, that folder must be inside a BeOS File System (BFS volume). Some of Beezer's operations require BFS support, hence users cannot install Beezer to a FAT16 or FAT32 partition.

6.2 Interface

This section deals with how to Beezer's interface. Beezer's interface does not need any detailed explaination as its very straight forward, hence this section focusses on the convenience provided by the interface and explains how the interface facilitates minimal input, non-obstrusiveness etc.

6.2.1 Welcome window

When Beezer is launched, the Welcome window (figure 6.1) will be displayed on screen.

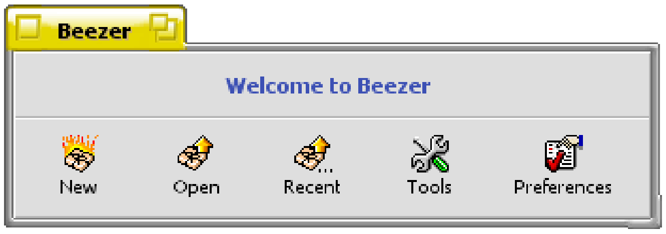


Figure 6.1: *Welcome Window*

- **New**: For creating a new archive
- **Open**: Open an existing archive
- **Recent**: Gives a list of recently used archives
- **Tools**: Gives a list of available tools for Beezer
- **Preferences**: Opens up the Preferences window for configuring various settings

Drag and Drop: Apart from the above functions, the Welcome window also provides an easy way to open archives. Dropping any archive (or set of archives) from *Tracker* and Beezer will try to open them.

6.2.2 Archive window

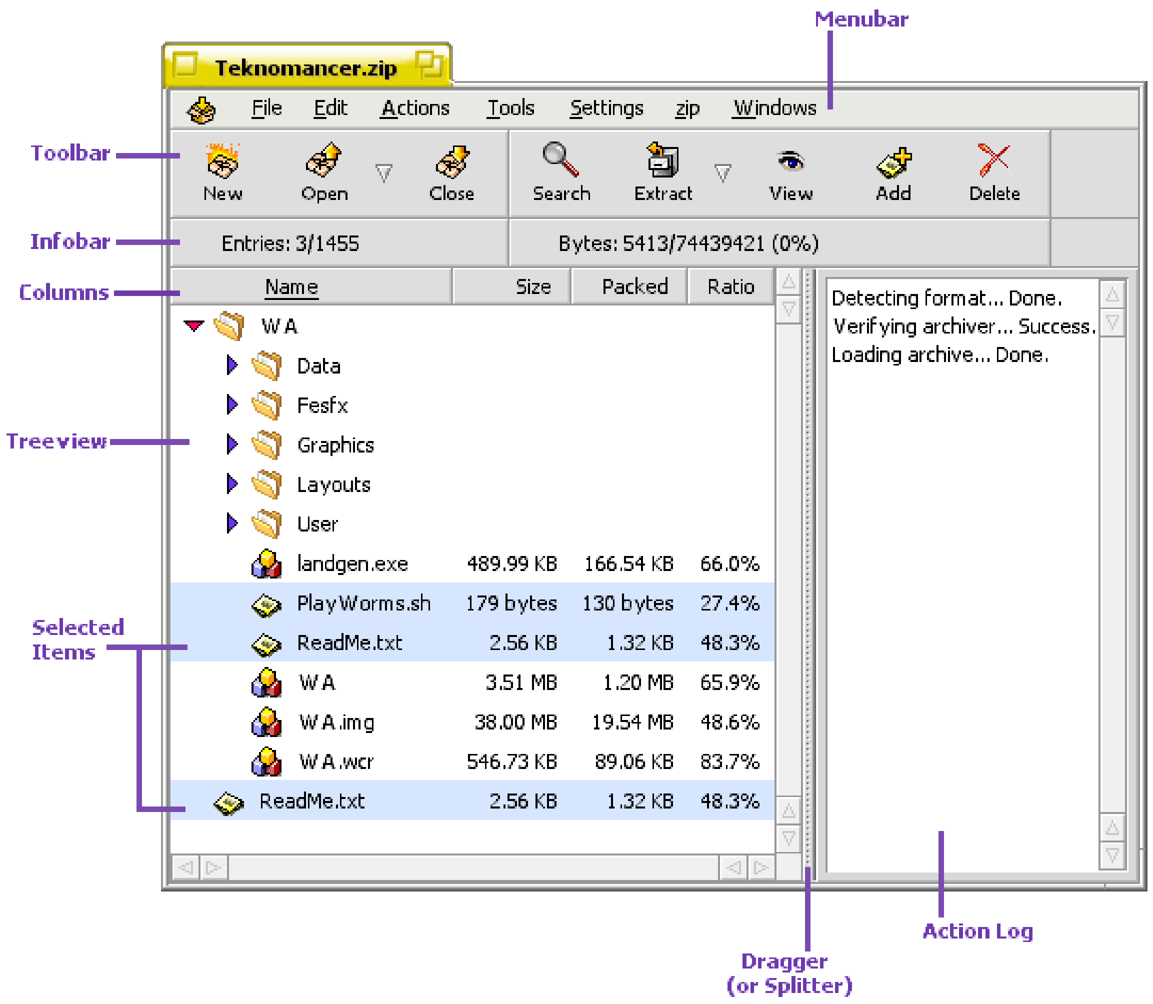
Each archive window (figure 8.1) represents an archive as closely to the physical state of the archive as possible. The archive windows shows an in-memory representation of the archive on disk.

The major components of the archive window are the *TreeView*, *Context menu*, *LogView*, *Toolbar*, *Infobar* and the *Menubar*.

Beezer reserves an archive window for every archive that is opened. The Archive window is written in such a way that it always remains as responsive as possible, rather than locking up during long operations.

Tree View

Archive contents are displayed in a directory-tree structure - the *Tree View*. Such a tree structures improves readability as compared to a plain list of files. It also makes



Entries: 3/1455 Bytes: 5413/74439421 (0%)

Name	Size	Packed	Ratio
WA			
Data			
Fesfx			
Graphics			
Layouts			
User			
landgen.exe	489.99 KB	166.54 KB	66.0%
PlayWorms.sh	179 bytes	130 bytes	27.4%
ReadMe.txt	2.56 KB	1.32 KB	48.3%
WA	3.51 MB	1.20 MB	65.9%
WA.img	38.00 MB	19.54 MB	48.6%
WA.wcr	546.73 KB	89.06 KB	83.7%
ReadMe.txt	2.56 KB	1.32 KB	48.3%

```
Detecting format... Done.
Verifying archiver... Success.
Loading archive... Done.
```

Figure 6.2: *Archive Window*

working with entire folders effortless. For instance, an entire folder can be selected and then the user can perform operations on the folder rather than selecting each file inside the folder. The Tree View also allows users to hide unessential details by hiding contents of sub-folders, thus the advantages of having a Tree View are plenty.

Columns

Columns in the *Tree View* are flexible. They can be resized by clicking and dragging the left or right borders of the column. The order of the columns can be re-arranged by dragging the column's title area. During the course of moving a column a blue-outline of the column is visible so as to indicate it is being dragged/moved. Columns can also be hidden from the menu: `Settings` –> `Columns`. Columns can be sorted by clicking on them. Columns that are sorted are underlined. Multiple columns can be sorted by holding SHIFT and clicking more columns. Double-clicking the edge of a column will automatically resize the column to fit all its content, which is similar to double-clicking a column edge in a *Tracker* window.

Sorting can be ascending, descending or unsorted (no underline). The various sorting methods can be cycled through by repeated clicks on the column until the desired sorting method is applied. For example, if a column is sorted in ascending order, clicking it will sort it in descending order. Another click will leave it unsorted, the next click will sort it in ascending order again, and so on.

Folders and Files

Folders in an archive can be collapsed (folded) and expanded (unfolded). This can be done through several ways. Hitting the LEFT arrow to collapse a selected folder and the RIGHT arrow key to expand (unfold) a folder. This is a very handy method of doing it as its close to the other arrow-keys which is used to navigate (scroll up and down) the archive. Folding/unfolding folders can also be done by clicking the arrow-icon besides the folder name or pressing ENTER key on the selected folder will also fold/unfold it.

An entry (file or folder) is selected by clicking on it. Multiple entries can be selected by using ALT or CTRL keys (depending on the user's keyboard settings). A range of entries can also be selected by using the SHIFT key. A selection can be further extended by using SHIFT with the DOWN or UP arrow-keys. Double-clicking an expanded (unfolded) folder selects all the files and sub-folders in it.

Please bear in mind that items that are hidden (in collapsed folders) are not selected. But some operations select the file if the parent folder is selected. For example, when deleting a folder that is collapsed (folded), the files inside it are automatically selected and deleted as well. So in general it must be assumed that any operations involving entire folders will select a folder's content automatically while operations involving files will not.

Selecting several folders and pressing ENTER will only collapse the first selected folder. For selecting a group of folders, see the Edit menu in the following section.

Context menu

Right-clicking on the treeview will pop-up the *Context menu* (figure 6.3). This comes in handy as the user need not to "reach" for the main menus each time he/she wants to do an frequent action. The context menu can be invoked either on a selection, or when no selection is present invoking the context menu will make a selection before invoking the menu. The `Select Folder` command select all the contents of all selected folders. It is sometimes handy to right-click a folder and choose this. It must be noted that the `Copy` command will not copy the selected file, but will only copy the text related to the file entry, i.e. the textual row information about the file seperated by commas.

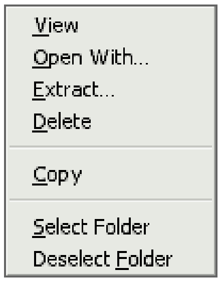


Figure 6.3: *Context Menu*

Log View

The *Log View* (also called *Action Log*) is a textual information area that updates the user with relevant information regarding a process and its result. It's like a status bar with a history. The log may also come in handy when users wants to verify their previous actions.It's a basic log that is not saved to the harddisk. Hence, once the archive widnow is closed the Log View's information is lost.

Right-clicking the Log View invokes a menu that allows users to retain its contents or clear its data. The Log View can be hidden using `Settings` –> `Action Log`. The Log View is completely resizable, by dragging the dotted space between it and the Tree View. This space is referred to as "dragger" or *Splitter*. The orientation of the view can also be changed by right-clicking the Splitter region, i.e. from vertical to a horizontal position and vice versa.

Toolbar

The *Toolbar* (figure 6.4) is a set of buttons that contains the most frequently used commands for easy access. The toolbar can be hidden by right-clicking space not occupied by a toolbar button (an empty space) or the space at the far end of the toolbar. It can also be hidden from `Settings` –> `Toolbar`.

1. **New**: Used to create new archives and add files to them as a simple two step process.

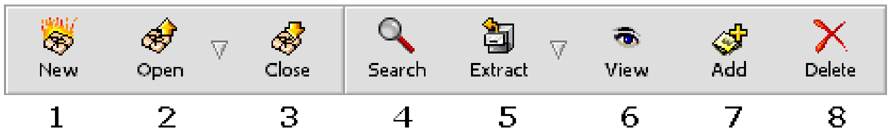


Figure 6.4: *Toolbar*

2. **Open**: Used to open existing archives without actually extracting its contents. The context menu, gives a quick list of recently opened archives.

3. **Close**: Close the archive window.

4. **Search**: Allows for searching the contents of an archive.

5. **Extract**: Allows for quick extraction of the archive to a default path, the context menu gives numerous locations for extraction with a single click!

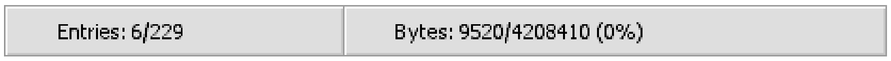
6. **View**: Opens the selected files for viewing. This automatically opens the appropriate application required to handle the file being viewed. For example, if an audio file is opened, it will be opened by a music player and so on.

7. **Add**: Allows further addition of files and folders to the current archive.

8. **Delete**: Allows removal of individual files and folders from the archive. Note: This deletion process cannot be reverted and its effects on the archive are permanent.

Infobar

The *Infobar* (figure 6.5) displays information regarding the selected entries (files and folders). The Infobar can be hidden/revealed by right-clicking on it or from `Settings` –> `Infobar`.



Entries: 6/229 | Bytes: 9520/4208410 (0%)

Figure 6.5: *Infobar*

The first 'slot' in the Infobar, displays the number of selected entries as against the total entries in the archive. For example, "6/229" indicates that 6 entries are selected from a total of 229 entries in the archive.

The next portion of the Infobar, displays the total size of the selected entries as against the total size of the archive (in bytes). It also shows how much percent of the archive size is selected.

It must be remembered that the display does NOT go deep into folders. Selecting a collapsed (folded) folder in the archive does NOT select the entries inside the selected folder. The Infobar thus shows only what is selected.

Menubar

The *Menubar* (figure 6.6) is like in any other BeOS application.

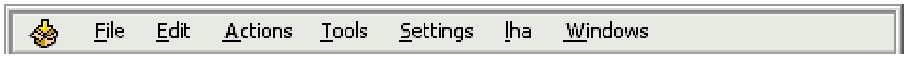


Figure 6.6: *Menubar*

(System)

- **Help**: Opens up the HTML documentation for Beezer.
- **About Beezer**: Shows copyright and credits.
- **Developer Info**: Shows information about the developer.
- **Quit**: Closes all open windows and quits the application.

File

- **New**: Allows creation of new archives.
- **Open**: Open an existing archive (sub-menu contains recent archives)
- **Close**: Close the current window
- **Delete**: Allows deletion of the entire the archive.
- **Archive Info**: Shows statistics about the archive.
- **Password**: If the archive type supports a password, this is where it can be edited.

Edit

- **Copy**: Either copy the selected archive entries or selected log-view text to the clipboard, this will only copy the visibile on-screen text, NOT the contents of the files in the archive.
- **Select All**: Selects all visible entries in the archive, sub-menu does as it says :)
- **Deselect All**: Deselects any selected entry from the treeview
- **Invert Selection**: Selects what is not selected and vice versa in one operation
- **Expand All**: Unfolds (expands) all folder entries
- **Expand Selected**: Unfolds (expands) all the folder entries in the selection
- **Collapse All**: Folds (collapses) all folder entries
- **Collapse Selected**: Folds (collapses) all the folder entries in the selection
- **Preferences**: Opens up Beezer's preferences

Settings

- **Save as Defaults**: Saves the current interface settings as the default settings, so that next time an archive (without interface settings) is opened, these settings would be used
- **Save to Archive**: Saves the current interface settings to the archive. The next time the archive is opened (and if permitted by Preferences) the settings will be restored. Please note that these settings tend to increase the size of the archive by a few bytes. Also they work only on BFS volumes as attributes are used, if the archive if moved to, say, a FAT32 volume and moved back to BFS, the attributes would be lost.
- **Toolbar**: Hide/Show the *Toolbar*
- **Infobar**: Hide/Show the *Infobar*
- **Action Log**: Hide/Show the *Log View*
- **Columns**: Hide/Show columns in the *Tree View*

- **While Opening**: Allows users to set the level of collapsed folders while loading the archive. The more collapsed the folders are, the faster the archive loads. But for working with archives the user may have to expand them anyway. Generally if extraction is the only reason for opening the archive, a folding level of 1 should give the fastest possible loading time.

(archiver) The archiver menu is a dynamically created menu. It contains the settings used by the archive format. For example, zip add-on creates a ”zip” settings menu having options specific to the zip format. Thus, the archiver menu varies in its options for each archive format. Some archive formats may not even have such a menu.

- **Save as Defaults**: Saves the current archiver settings as defaults. The next time an archive of this type is opened, these settings would be used
- **Save to Archive**: Saves the current archiver settings to the archive. The next time the archive is opened (and if permitted by Preferences) the settings will be restored. Please note that these settings tend to increase the size of the archive by a few KB. Also they work only on BFS volumes as attributes are used.

Windows Gives a list of archives that are currently open. The first 10 archives have shortcut keys as `COMMAND` + n, where ”n” is from 0 to 9. The currently active window is indicated by a tick mark.

6.3 Performing Actions

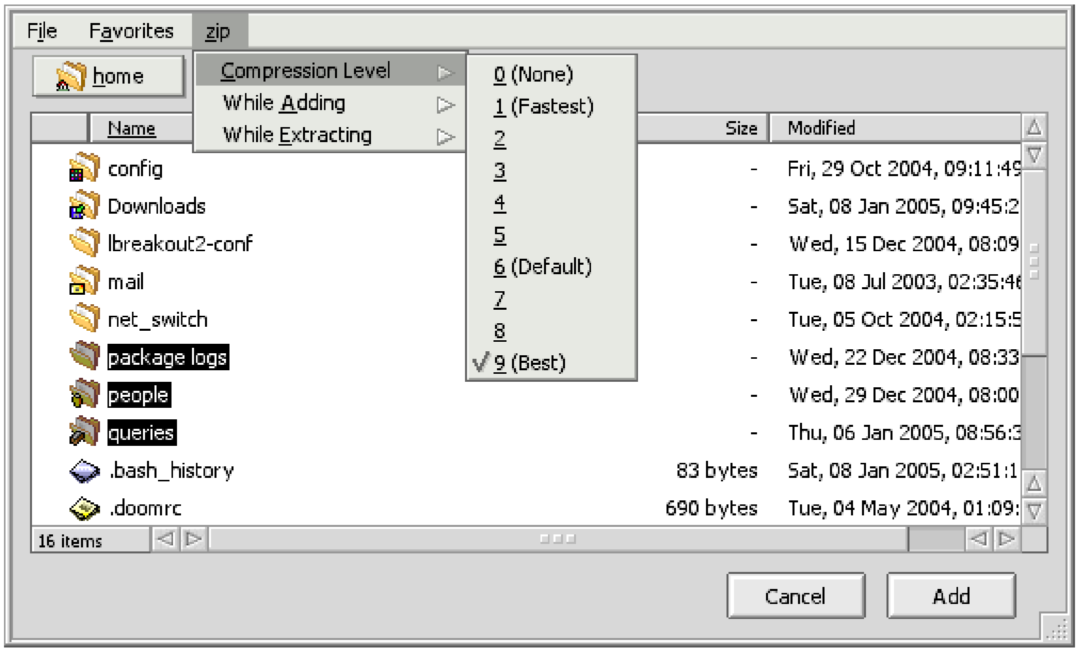
This section covers in detail how to create new archives, extract archive contents and other manipulations that can be performed on archives.

6.3.1 Creating archives

1. When a new archive is created, a file requester prompts the user to enter the location (path) and name of the archive. The type of archive can also be selected. Once the name of the archive, its type (format) has been selected, clicking the ”Create” button will proceed to the next step of adding files to archive.

   It must be carefully noted that this file requester is only for choosing the name, type and location of the archive that is to be created and NOT what files should be in the archive. The following file requester is for that.

2. Next, a new file requester (figure 6.7) that prompts users to add files & folders to the archive is shown. Here, depending on the type of archived selected, the appropriate archiver menu is displayed.



Name	Size	Modified
config	-	Fri, 29 Oct 2004, 09:11:49
Downloads	-	Sat, 08 Jan 2005, 09:45:2
lbreakout2-conf	-	Wed, 15 Dec 2004, 08:09
mail	-	Tue, 08 Jul 2003, 02:35:4
net_switch	-	Tue, 05 Oct 2004, 02:15:5
package logs	-	Wed, 22 Dec 2004, 08:33
people	-	Wed, 29 Dec 2004, 08:00
queries	-	Thu, 06 Jan 2005, 08:56:3
.bash_history	83 bytes	Sat, 08 Jan 2005, 02:51:1
.doomrc	690 bytes	Tue, 04 May 2004, 01:09:

Figure 6.7: *Add File Panel*

The advantage of accessing the menu here is that users get to adjust the options before creating the archive. Say, for example, zipping mp3s (to transfer them with their attributes to another BeOS system via FTP), the user might want to set the compression level to zero (no compression just store) as it makes things much faster and the size-tradeoff is less. Thus it can be done here by choosing the compression level.

Also, if the archive supports password protection for files, Beezer gives users the chance to enter a password for the files being added. This can be left blank to specify no password protection is to be used for the files being added. Once the required files have been selected, clicking the "Add" button will start adding files to the new archive.

6.3.2 Deleting archives

Beezer also users to delete archives from the disk. This can be done by choosing `File` –> `Delete`. Beezer always asks a confirmation before deleting the archive permanently from the disk, and also gives an option of moving the archive into the BeOS Trash (similar to recycle bin).

Note If the archive is deleted permanently it cannot be recovered. This option must be used with care.

6.3.3 Searching archives

Beezer allows users to search for files and folders in an archive. Please note, this search does not search the actual contents of each file in the archive. Rather it searches the names of files/folders; more specifically allows searching of any *Column* in the *List View*. Once the user chooses the search option, the Search window (figure 8.2) pops up.

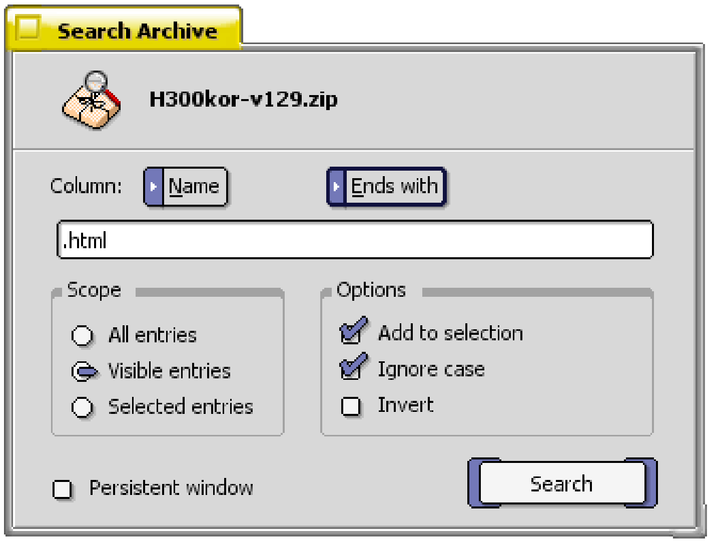


Figure 6.8: *Search Window*

Column The column pop-up lets users choose which column to search in. The columns that are listed here depend on the type of archive that is opened. Some archive types do not have information about all columns. For example, the tar archiver does not list information about the CRC of files or the "Packed" size of files (as tar doesn't support compression only storage). Thus only the columns that have searchable content are listed. The pop-up menu beside the Column pop-up menu is a list of the methods of searching. They include "Starts with", "Ends with", "Contains", "Matches wildcard expression" and "Matches regular expression".

Search text field Users need to enter what text they are searching for in this text box. In the above example, the user is searching for ".html" files.

Scope Defines the scope of the search, the following are the available options:

- **All Entries**: This means all the entries in the archive will be searched. Before the search begins, all the folded (collapsed) folders will be expanded, then the search will commence

- **Visible Entries**: This option specifies only physically visible entries be searched. Thus if a folder is collapsed, files and folders inside the collapsed folder will not be searched

- **Selected Entries**: This option specifies that only the entries that have been selected must be searched

Options Other search options that add to the criteria of the search is to be specified here. The following are available:

- **Add to selection**: This means any newly found entries are added to the existing selection. If this option is not checked, a search always deselects all entries before searching

- **Ignore case**: Uppercase and lowercase alphabets are treated the same, e.g. a search for "ReadMe" and "readme" yield similar results

- **Invert**: Reverses the result of the search, i.e. all entries that do not match the search are selected and vice versa

Persistent window Checking this option prevents the search window from closing after a search. This can be very useful when several searches are needed for an accumulate result. Like say, searching for all textual content such as ".htm", ".txt", ".doc" etc.

6.3.4 Testing archives

Testing archives is a process of determining if the content of the archive is flawless. Some archive formats do not support testing. After selecting the testing action, the archive will immediately be tested for errors. After the test process is completed the user is shown the result of the test, and if needed he/she can view the full details of the test.

If an error occurs, a Warning pop-up warns you that the archiver has found one or more errors and warnings. Here again the user can choose to inspect the detailed test information.

6.3.5 Extracting archives

Extracting the contents of the archive can be done through any of the following ways:

- Choosing `Actions` –> `Extract` extracts the entire archive to the default extract folder. If no default extract folder is specified, Beezer asks a destination folder. A default extraction folder can be specified in Preferences. The sub-menu displays various probable extract destinations, clicking them starts extraction immediately without any further prompts.
- Choosing `Actions` –> `Extract To` always prompts for a destination folder whether a default extract folder is specified or not. Sub-menu displays various probable extract destinations, clicking them starts extraction immediately without any further prompts
- Choosing `Actions` –> `Extract Selected` does the same as Extract To,but for selected entries.
- Choosing `toolbar`: `Extract`. This always extracts the entire archive and not selected entries. Clicking on it extracts the entire archive to the default extract location or asks for a destination folder. The context portion provides some probable extract locations.

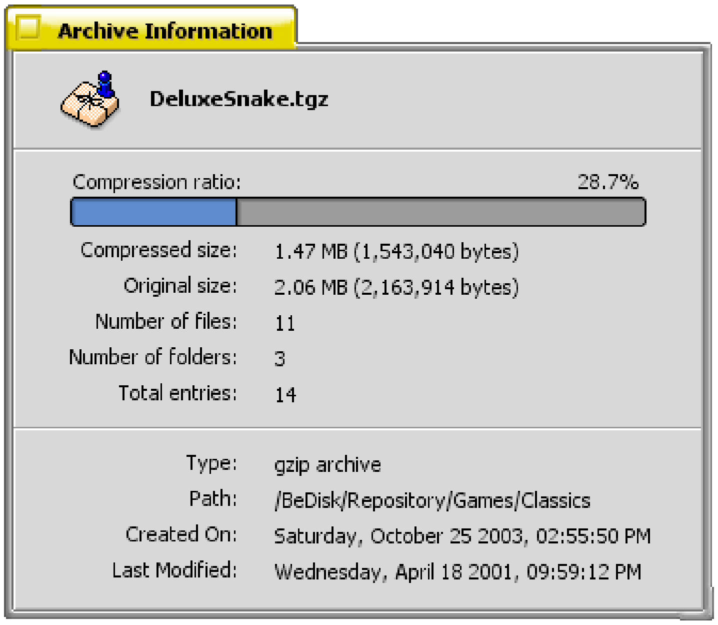
Drag 'n Drop

Files and folders can be extracted by dragging and dropping them on to a *Tracker* window.

Drag 'n drop extraction always extracts selected entries. If the entire archive is to be extracted through drag 'n drop, `Edit` –> `Select All` and then drag 'n drop the selection. While dragging entries to be extracted, the number of files and folders that will be extracted will be shown. But since extraction is a hierarchial action, parent folders that are not selected while dragging are not counted and displayed although they will be extracted.

6.3.6 Archive information

The Archive Info window (figure 6.9) displays information relevant to the archive. The information includes the compression ratio of the archive, the type of archive and other information which maybe useful.



DeluxeSnake.tgz

Compression ratio: 28.7%

Compressed size:	1.47 MB (1,543,040 bytes)
Original size:	2.06 MB (2,163,914 bytes)
Number of files:	11
Number of folders:	3
Total entries:	14
Type:	gzip archive
Path:	/BeDisk/Repository/Games/Classics
Created On:	Saturday, October 25 2003, 02:55:50 PM
Last Modified:	Wednesday, April 18 2001, 09:59:12 PM

Figure 6.9: *Archive Info Window*

- **Compression Ratio**: Denotes how compressed the archive is compared to its uncompressed size. The higher the percentage, the more compressed the archive
- **Compressed Size**: The current storage size of the archive
- **Original Size**: The estimated size of files when the archive is uncompressed. (note: size of file attributes are not included)
- **Number of files**: The total number of files present in the archive

- **Number of folders**: The total number of directories in the archive
- **Total entries**: The total number of entries (directories + files)
- **Type**: The type of archive as detected by Beezer
- **Path**: The location of the archive
- **Created On**: The date and time when the archive was created (as reported by filesystem)
- **Last Modified**: The date and time when the archive was last modified (as reported by filesystem)

6.3.7 Adding files

Beezer allows users to add files and folders (entries) to any folder inside the archive and not just at the root of the archive. Before adding entries the user needs to choose where to add them (unless they are adding files through drag 'n drop). The following list explains where Beezer will add new files:

- If a file is selected, newly added entries will be added in the same folder as the selected file.
- If a folder is selected, newly added entries will be added inside the selected folder.
- If nothing is selected, newly added entries will be added to the root of the archive.
- If there are multiple selections, the first visible selected item will be used.

Once the user knows where the files are going to be added, the next step is choosing one of the following:

- From `Actions` –> `Add`, or
- From `toolbar: Add`

A file requester will pop-up asking the user to choose which files and folders are to be added. After choosing them, pressing the "Add" button will start adding the selected entries to the archive.

Drag 'n drop

Users can also add files to an archive by dropping them from *Tracker* on to the *List View*. In this case, the selected entry will not be used to determine where the new entries will be added. Instead a horizontal marker (that appears on the List View as the user drags the entries to be added) will indicate where the new files will be added.

If the user wants to add the files to the root of the archive, he/she must use the `SHIFT` key. When `SHIFT` is held down the horizontal marker becomes red in colour indicating that the files will ba added in the root of the folder no matter where it appears.

Thus a blue marker indicates the exact position where the files will be added, while a red marker indicates the files will be added to the roof of the archive regardless of its position in the List View.

6.3.8 Viewing files

Beezer allows users to view files in an archive (without extracting the entire archive) using the program (as configured in the *FileTypes* preferences) that can open that file. For example, view image files using an image viewing application, view text files using a text editor and so on...

To view files one of the following methods may be used:

- Choose `Actions` –> `View`, or
- Press `toolbar` –> `View`, or
- Use the context menu's `View` command, or
- press the `ENTER` on the selection, or
- double-click the selected files

Since Beezer allows users to view several files simultaneously (using multiple selections), Beezer warns and confirms when more than 10 files simultaneously are going to be viewed. If the selection contains folders, they will not be checked out for viewing.

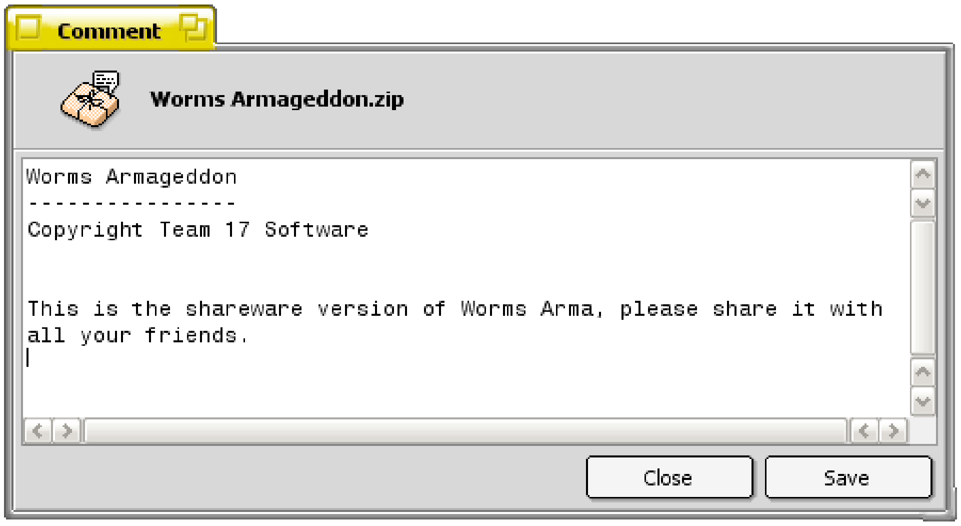
6.3.9 Creating folders

aUsers can create directories (folders) anywhere in an archive. They must choose where to create the new folder similar to how they choose where to add files.

Once the location has been determined, choosing `Actions` –> `Create Folder`. The user will be prompted to enter the name of the new folder. The Create Folder window also shows the user where the new folder will be added. ":Root:" indicates that the new folder will be added to the root of the archive. Once the folder name is entered, pressing the "Create" button will make a new folder in the archive in the specified location. A status window will be shown until the operation is complete.

6.3.10 Editing comments

Some archivers supports comments, for example, the zip archiver supports zipfile comments. `Actions` –> `Comments` will allow users to edit the comments of the archive.



```
Worms Armageddon
----------------
Copyright Team 17 Software


This is the shareware version of Worms Arma, please share it with
all your friends.
```

Figure 6.10: *Comments Window*

The Comment window (figure 6.10) will allow users to edit, save as well as discard comments. Pressing "Cancel" discards any change since the window was opened. Deleting the existing comment and pressing "Save" will remove comments from the archive. Please note that zip file comments are supported only if the zipinfo program has been installed. If it is not installed you will not be able to work with zipfile comments. By default, BeOS R5 and Zeta comes with zipinfo.

6.4 Tools

6.4.1 QuickCreate archive

The QuickCreate Archive (figure 6.12) is a *Tracker* add-on that lets users create archives with minimal clicks! The basic idea was to do a replacement for *Zip-O-Matic* but instead of just supporting zip format alone, QuickCreate would allow control over the format, and all the options that are supported in Beezer while creating new archives.

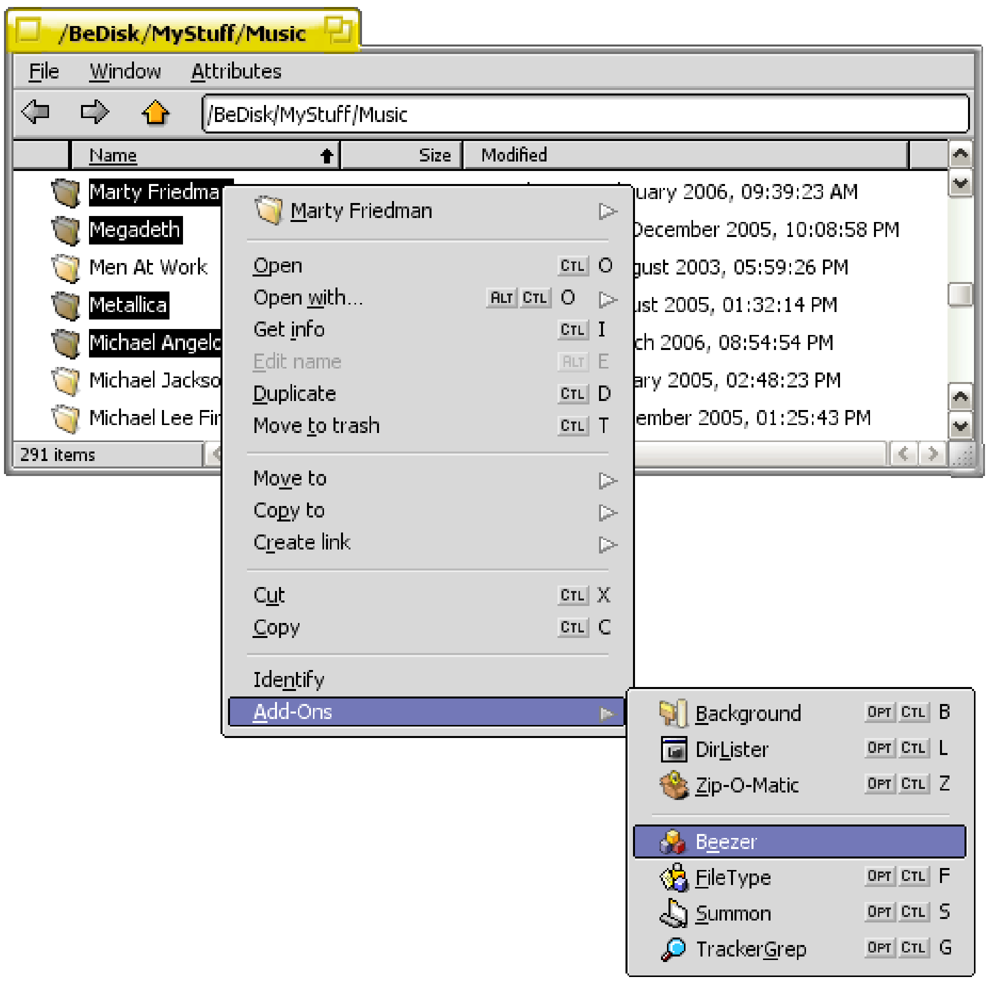


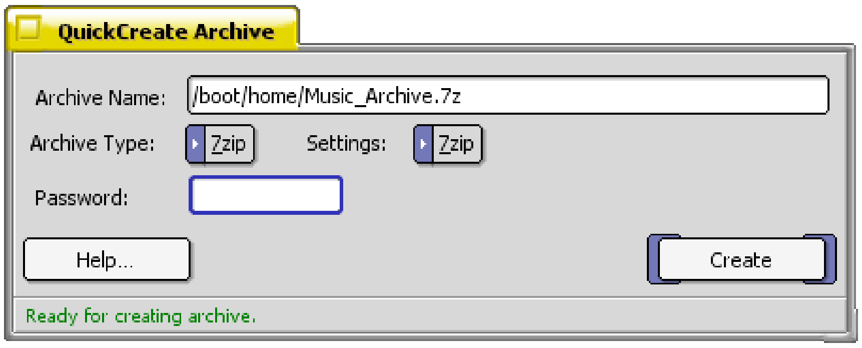
Figure 6.11: *Invoking Tracker Add*

The QuickCreate Archive is in fact a kind of hack. It resides within Beezer but is invoked using a *Tracker* add-on. Users cannot launch QuickCreate explicitly from Beezer as it is not listed under ”Tools”. It is included in the documentation under ”Tools” from a functional point of view as it’s an extra tool.

How to use

This sub section gives a quick run through on how to use QuickCreate add-on to make a new archive.

1. The files that should be archived must first be selected in *Tracker*.
2. The selection must be right clicked, and `Add-Ons` –> `Beezer` (figure 6.11)
3. Now the QuickCreate Archive window (figure 6.12) will be shown.



Archive Name: /boot/home/Music_Archive.7z
Archive Type: 7zip
Settings: 7zip
Ready for creating archive.

Figure 6.12: *QuickCreate window*

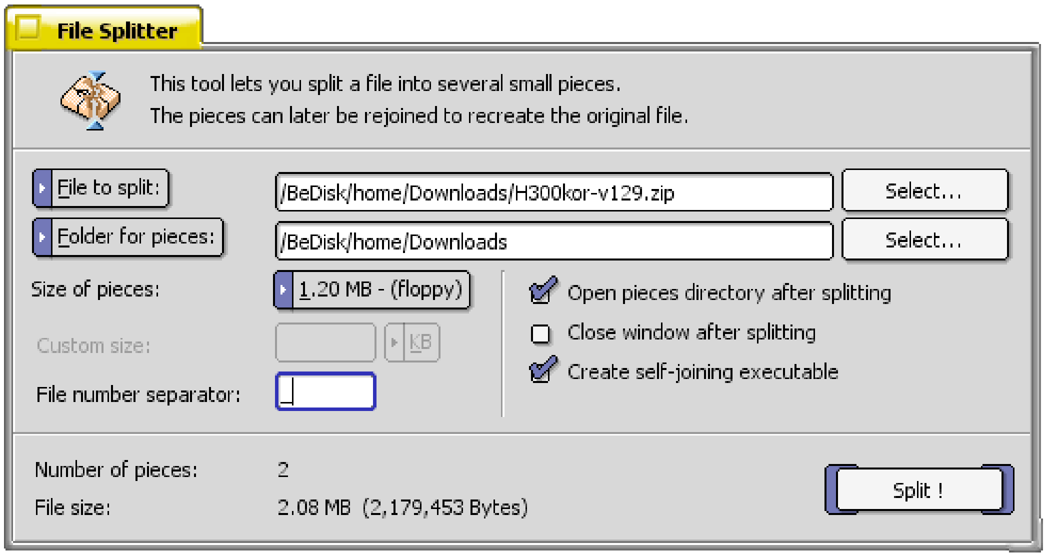
The various elements in the QuickCreate window are documented below....

- **Archive Name**: This shows the path and name of the archive that is to be created. Users can chop and change elements of the path (such as modifying destination directories, and file name).
- **Archive Type**: Allows users to select the type of archive. As always, the necessary binaries needs to be installed in the system and the "workers" folder must have valid (non-broken) links to the binaries.
- **Settings**: For the chosen archive type, this is the list of available settings.
- **Password**: For those archive types that supports passwords (and supported by Beezer), this is the place where they need to be entered before creating the archive.
- **The status line**: The status line at the very bottom of the QuickCreate window shows some information to assist users in creating archives. For example, if the archive name chosen already exists, the status line shows a warning text reminding the user that the existing file will be overwritten and so on.

6.4.2 File Splitter

The File Splitter is used for creating small pieces of a larger file. The File Splitter will not actually break a file into smaller pieces but rather make small pieces from a file, meaning the original file is not altered, only the pieces are created.

The File Splitter window (figure 8.3) can be opened from two places, eitherfrom the Welcome window or from the Archive window.



File Splitter

This tool lets you split a file into several small pieces.
The pieces can later be rejoined to recreate the original file.

File to split: /BeDisk/home/Downloads/H300kor-v129.zip
Folder for pieces: /BeDisk/home/Downloads
Size of pieces: 1.20 MB - (floppy)
Custom size:
File number separator:

[✓] Open pieces directory after splitting
[ ] Close window after splitting
[✓] Create self-joining executable

Number of pieces: 2
File size: 2.08 MB (2,179,453 Bytes)

Figure 6.13: *File Splitter*

- **File to split**: When clicked gives a list of recently split files. Clicking the "Select" button gives a file request panel. Users can alternatively, directly type in the path of the file.
- **Folder for pieces**: When clicked gives a list of recently chosen directories. Clicking the "Select" button gives a folder request panel. Users can alternatively, directly type in the path of the folder. This directory is where the File Splitter will put the split file pieces.
- **Size of pieces**: Allows the user to specify what is the size of each piece/chunk. They can either choose from a list of pre-defined sizes such as "1.44 MB - (floppy)", "650 MB - (CD)" and so on, or choose "Custom Size" and specify their own size. The minimum size of a piece is 1 byte, and the maximum size being 1 bytes less than the size of the file being split.

- **File number separator**: By default, File Splitter when splitting a file called `MySong.mp3` into 2 pieces, would create the pieces named: `MySong.mp3_01` and `MySong.mp3_02`. The underscore ("_") here is called the file number separator. The file number separator thus separates the file name from the split order number.

  Users can specify any alterate text in place of the underscore as the file number separator, for example using ".z" for compatibility with the *WinZip* spanning method and so on. Currently, digits (0-9), wildcards (* and ?) and path characters (: and /) are not allowed.

- **Open pieces folder after splitting**: If this option is checked, after splitting of the file is complete, the folder in which the pieces were created is automatically opened

- **Close window after splitting**: Closes the File Splitter tool after splitting is complete. If the split operation is cancelled, the window is nott closed even if this option is specified.

- **Create self-joining executable**: If this option is checked, the File Splitter will create a BeOS executable file in the pieces folder, that when run will automatically re-join the pieces to create the original file.

  This executable is a BeOS ELF executable and cannot be run from Windows. Also this executable can be run from a system that does not have Beezer installed. It is completely independent of Beezer and would be useful while sending the split files to someone who has BeOS or flavours of it and whom does not have Beezer. It is also very useful for quickly re-joining the files and is faster than opening Beezer, and using the File Joiner tool.

  Note: For the self-joining executable to work it must be placed in the same directory as the pieces and the pieces should not be renamed, as the executable contains information about the pieces and it's file number separators

- **Number of pieces**: Shows how many pieces will be created for the specified size of pieces and the file size. The File Splitter cannot create any more than 32767 pieces.

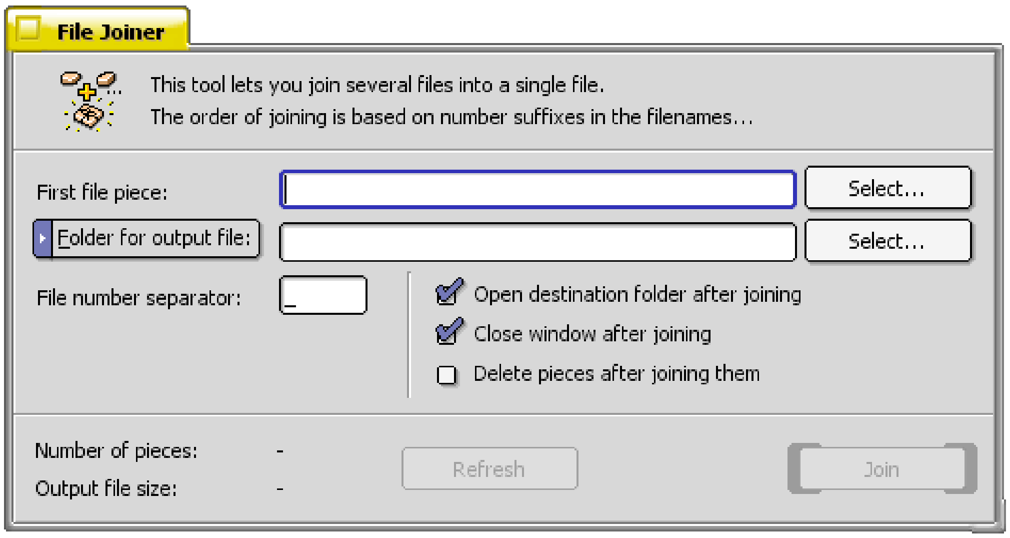
- **File size**: Shows the size of the file that is to be split.

- **Split!**: Clicking this would start the split process. This process can safely be cancelled before it completes. File Splitter will only delete partially created pieces, leaving fully created pieces and the original file as they are.

Tip! Files to be split can be dropped from *Tracker* on to the File Splitter window. Also, dropping a folder will be used as the destination folder for the pieces.

6.4.3 File Joiner

The File Joiner is used to re-join the pieces of files created by the File Splitter tool. Though there is an option of using the self-joining executable the Joiner tool offers more options. The File Joiner window (figure 6.14) can be opened from two places within Beezer, one from the Welcome window and the other from the Archive window.



This tool lets you join several files into a single file.
The order of joining is based on number suffixes in the filenames...

First file piece:
Folder for output file:
File number separator: _

✓ Open destination folder after joining
✓ Close window after joining
☐ Delete pieces after joining them

Number of pieces: -
Output file size: -

Figure 6.14: *File Joiner*

The various components of the File Joiner window follow:

- **First file piece**: Allows users to specify the first piece in the list of pieces that are to be joined. Clicking the "Select" button gives a file request panel. Users can alternatively, directly type in the path of the first piece file.
- **Folder for output file**: When clicked gives a list of recently chosen directories. Clicking the "Select" button gives a folder request panel. Users can alternatively, directly type in the path of the folder. This directory is where the joined file will be created.

- **File number separator**: The file number separator separates the file name from the split order number. similar to the one specified in File Splitter.
- **Open destination folder after joining**: If this option is checked, after joining the files is complete, the folder in which the joined file was created is automatically opened through *Tracker*.
- **Close window after joining**: Closes the File Joiner tool after joining is complete. If the join operation is cancelled, the window is not closed even if this option is specified.
- **Delete pieces after joining**: If this option is checked, the piece/chunk files are automatically deleted after they are joined. The pieces will not be deleted if the operation is cancelled before it's complete.
- **Number of pieces**: Shows you how many pieces are there to be joined. When using custom number separators this can be very useful.
- **Output file size**: Shows you the estimated size of the joined file before creating it.
- **Refresh**: Clicking this will update the above two controls.
- **Join!**: This will start the process of joining the files.

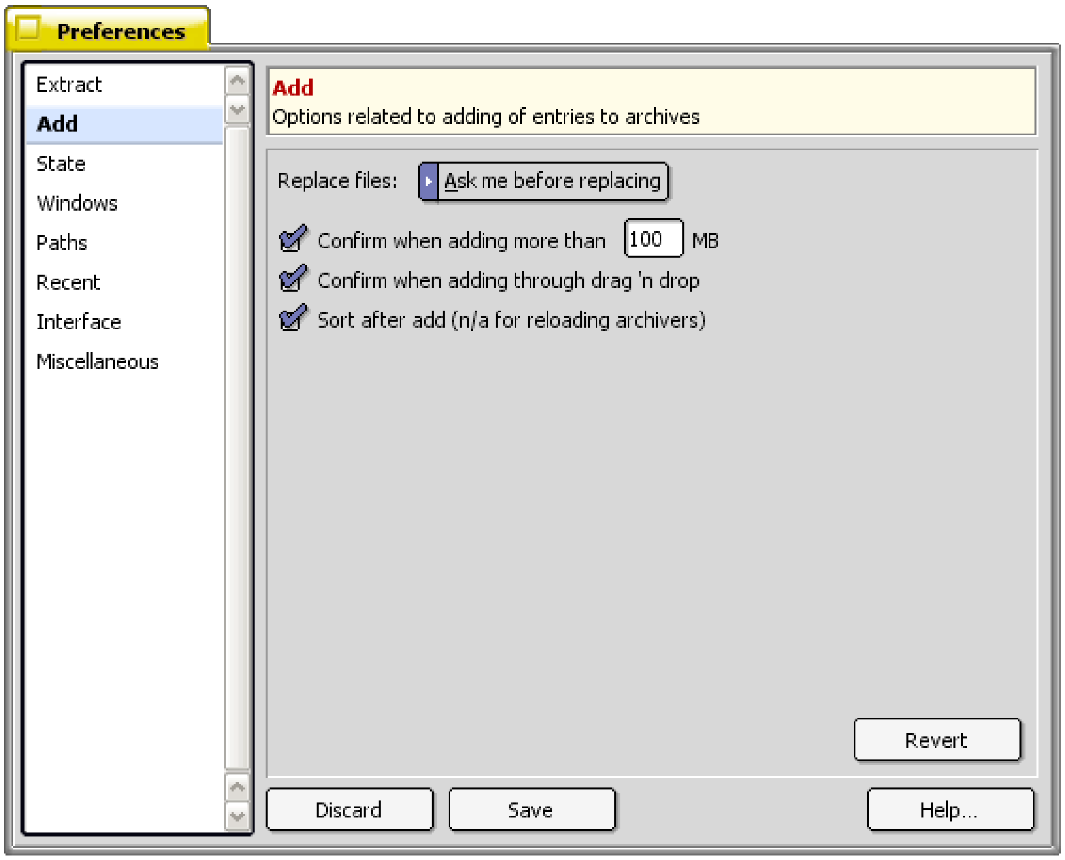
Tip! Users can drop the first piece file to the File Joiner window. Also, you they can drop a folder on the File Joiner window to specify the folder for creating the joined file.

6.5 Preferences

Preferences allow users to customize and tweak Beezer in every possible way. Beezer's strong point has always been its customizability. Preferences can be accessed either from `Edit` –> `Preferences` or from `toolbar`: `Preferences` from the Welcome window.

The Preference window (figure 6.15) classifies various options into "Panels", such as "Extract", "Add" etc. Extract panel would have options related to extracting of archives, Add with adding of files to the archive and so on.

Each panel has "Revert" button which when clicked undoes any change made since the particular preference panel was opened. The "Save" button located at the bottom of the Preference window, saves and applies the currently shown preferences, while "Discard" cancels any changes made since the opening of the Preference window.



Add
Options related to adding of entries to archives

Replace files: Ask me before replacing

✓ Confirm when adding more than 100 MB
✓ Confirm when adding through drag 'n drop
✓ Sort after add (n/a for reloading archivers)

Figure 6.15: *Preferences window*

6.5.1 Restoring default options

Beezer does not provide any internal method to restore the default options for a panel, but users can easily restore the default of any preference panel simply by deleting that panel's settings file. Every panel saves its settings in a seperate file. Deleting that file will force Beezer to use the default options for that panel. For example, the add panel will have be saved in `settings/add_settings`, extract panel in `settings/extract_settings`.

6.5.2 Extract panel

The extract panel gives options related to extracting of archives. This extraction includes not just extracting the entire archive but also extracting selected files and drag 'n drop extraction as well.

- **Open destination folder after extraction**: After extraction of files is complete the folder in which it was extracted is automatically opened
- **Close window after extraction**: Closes the archive window after extraction is complete (note: this may lead to quitting of Beezer depending on "When last archive is closed" option in Miscellaneous panel)
- **Quit Beezer after extraction**: Quits Beezer after extraction of an archive is complete

6.5.3 Add panel

Here users can configure options related to adding entries to the archive

- **Replace files**: Specifies what action has to be taken when a file(s) that a user is adding already exists in the archive.
- **Confirm when adding more than "n" MB**: Whether confirmation is asked when more than "n" MB of data is added. The default is 80 MB
- **Confirm when adding through drag 'n drop**: Whether confirmation is asked when users use drag 'n drop method for adding files to an archive
- **Sort after add (n/a for reloading types)**: Whether the *Tree View* must be sorted after each add operation. For very large archives users may not want to sort after every add operation as it can take time on a slow machine. However, for archive types that reload the entire archive after an add operation this option wouldn't be applicable

6.5.4 State panel

The state panel allows users to configure state information. State info is information that is 'tagged' along with archives using BFS attributes. Because of this, state info would be lost if the archive is moved to a filesystem that doesn't support BFS attributes (such as vfat, NTFS, ext3 etc.).

Storing Archiver settings to the archive can increase the archive's size by several KB. Interface state only occupies about 413 bytes. Also users can manually remove state info from an archive by opening the archive through *QuickRes* utility and deleting the fields named "bzr:ui" and "bzr:ark".

Users can manually save state info into archives of their choice rather than using these automatic options for every archive. For this they must use `Settings` –> `Save to Archive` and `(archiver menu)` –> `Save to Archive` from the Archive window.

- **Store Automatically**: The options under this section determine what state information to automatically store in archives. Interface state saves interface settings such as the size, position, columns, sorting, folding level etc. of the archive window. Archiver settings saves the archiver's settings menu in the archive. By default automatic storing of state info is off.
- **Restore Automatically**: This automatically restores the respective states from the archive when its loaded, by default restoring of state info is ON

6.5.5 Paths

The paths panel allows users to specify various default locations (paths) etc.

- **Open Path**: Specifies the default directory that will be shown in the "Open archive" file requester. For example, if most of the archives are in one folder, say `/boot/home/Downloads/`, users can specify that path here so that they can quickly open archives without navigating to that folder each time
- **Add Path**: Specifies the default directory that will be shown in the "Add files to archive" file requester
- **Extract Path**: Specifies the default extract path
- **Favourite extract paths**: Here users can add (using "+" button) any number of favourite extract locations (paths). These locations show up in the extract sub-menus so users can quickly extract to any of your favourite locations. Removal of a path from here can be done using the "-" (minus sign) button
- **List more paths (using archive name)**: When this option is turned on (which it is by default), for each favourite path, the name of the archive is added and a new path is generated and shown in the extract sub-menus

For example, for `/boot/Programs` as a favourite path, and the user opens the archive `MyFiles.zip`, and has this option on, will produce another favourite path `/boot/Programs/MyFiles` apart from `/boot/Programs`. This can be useful when users open archives that do not have a root folder

6.5.6 Recent panel

The recent panel allows users to configure settings related to recent archives and recent paths.

- **Number of recent archives**: Allows users to specify how many recent archives to list in the recent archives menu (10 recent archives are listed by default)
- **Show full path in recent archives**: If this option is turned on, the full path of the archive would be displayed, otherwise only the name of the archive would be displayed. (by default full paths are turned OFF)
- **Number of recent extract paths**: Allows users to specify how many recent extract paths to list (by default 5 recent extract paths are listed)

6.5.7 Interface panel

The interface panel allows users to configure settings related to Beezer's GUI.

- **Full length Toolbar & Infobar**: Allows users to choose if you want the *Toolbar* and the *Infobar* to have a full window width look.
- **Configure colours**: Allows users to configure colours used in the interface - users can restore default interface settings by deleting the corresponding settings file
- **Default interface settings**: Allows users to modify the default Archive window settings. These default settings will be used for newly created archives and for archives that do not have interface settings stored with them, and if you have turned OFF the "Restore State" option for all archives!
  - **Show Toolbar**: Toggle default Toolbar visibility
  - **Show Infobar**: Toggle default Infobar visibility
  - **Show Action Log**: Toggle the Action Log's visibility

  - **Folding**: Allows users to choose the depth of collapsed items. This can sometimes prove very useful as it can greatly enhance the speed of loading archives!

6.5.8 Miscellaneous

Offers all options that don't fit into any of the other panels.

- **When last archive is closed**: Allows users to choose what action to take when Beezer starts up:
  - **Show welcome window**: Shows the welcome window when starting up, this is the default action
  - **Show create archive panel**: Shows the panel for creating archives, choosing "Cancel" will quit Beezer
  - **Show open archive panel**: Shows the panel for opening an existing archive, choosing "Cancel" will quit Beezer
- **When last archive is closed**: Users can specify what action should be taken when the last archive window is closed:
  - **Show welcome window**: re-opens the welcome window
  - **Quit Beezer**: quits Beezer completely
- **Show comments**: Shows comment (if any) after opening the archive, by default it's on
- **Check file types at startup**: Each time Beezer starts, checks if Beezer is the preferred application for its supported archive types, by default it is off
- **Register file types now**: This registers Beezer as the preferred application for its supported archive types. Once this is done, the "Check file types at startup" option can be turned off
- **Default archiver**: Users may choose the default archiver (archive format) while creating archives

The arj archiver gives you the following options:

Compression Level

Allows users to specify the amount of compression while creating archives on a scale of 0 to 4. 1 is maximum compression, 4 is minimum and 0 (zero) is no compression (just storage).

While Adding

- **Recurse into folders**: Adds all the contents of folders being added, including sub-directories and its contents

While Extracting

- **Update files (new and newer)**: Overwrites existing file if archive file is newer, creates files that are missing
- **Freshen existing files**: Only existing files are overwritten if they are older than the ones in the archive, no new files are extracted
- **Enable multiple volumes**: Enables extraction of multi-volume archives (recommended)

Chapter 7

Source Code

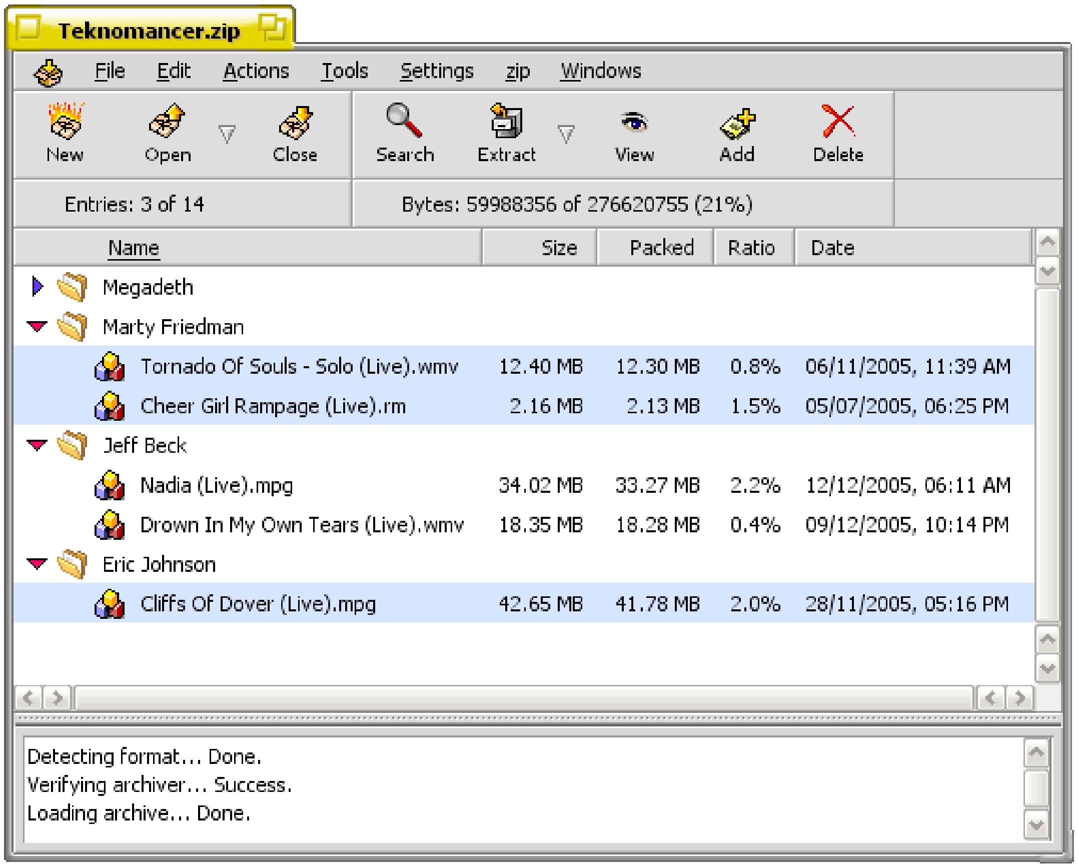
```
/*
 *    Beezer
 *    Copyright (c) 2002 Ramshankar (aka Teknomancer)
 *    See "License.txt" for licensing info.
*/

#include "BevelView.h"
#include "UIConstants.h"

//=============================================================================//
BevelView::BevelView (BRect frame, const char *name,
    BevelType bevelMode, uint32 resizeMask, uint32 flags)
    : BView (frame, name, resizeMask, flags | B_FRAME_EVENTS)
{
    // Set up colors, edges and cache the Bounds() rectangle
    m_bevelType = bevelMode;
    rgb_color backColor;

    if (Parent())
        backColor = ViewColor();
    else
        backColor = K_BACKGROUND_COLOR;

    switch (m_bevelType)
    {
        case btDeep: case btInset:
            m_darkEdge1 = tint_color (backColor, B_DARKEN_2_TINT);
            m_darkEdge2 = tint_color (backColor, B_DARKEN_3_TINT);
            m_lightEdge = K_WHITE_COLOR;
            m_edgeThickness = m_bevelType == btInset ?
                        btInsetThickness : btDeepThickness;
            break;

        case btOutset:
            m_darkEdge1 = K_WHITE_COLOR;
            m_darkEdge2 = tint_color (backColor, B_DARKEN_3_TINT);
            m_lightEdge = tint_color (backColor, B_DARKEN_2_TINT);
            m_edgeThickness = btOutsetThickness;
            break;

        case btBulge:
            m_lightEdge = tint_color (backColor, B_DARKEN_3_TINT);
            m_darkEdge2 = tint_color (backColor, B_DARKEN_2_TINT);
            m_darkEdge1 = tint_color (backColor, B_LIGHTEN_1_TINT);
            m_edgeThickness = btBulgeThickness;
            break;

        case btNoBevel:
            break;
    }

    m_cachedRect = Bounds();
}
//====================================================================//
void BevelView::Draw (BRect updateRect)
{
    // Draw the edges based on the type of edge specified
    switch (m_bevelType)
    {
        case btNoBevel:
            break;
        case btDeep: case btBulge:
```

```
        {
            SetHighColor (m_darkEdge2);
            StrokeRect (BRect (m_cachedRect.left + 1,
                            m_cachedRect.top + 1,
                            m_cachedRect.right - 1,
                            m_cachedRect.bottom - 1));
            BeginLineArray (4L);
            AddLine (m_cachedRect.LeftTop(),
                     m_cachedRect.RightTop(), m_darkEdge1);
            AddLine (m_cachedRect.LeftTop(),
                     m_cachedRect.LeftBottom(), m_darkEdge1);
            AddLine (m_cachedRect.RightTop(),
                     m_cachedRect.RightBottom(), m_lightEdge);
            AddLine (m_cachedRect.RightBottom(),
                     m_cachedRect.LeftBottom(), m_lightEdge);
            EndLineArray ();
            break;
        }
        case btInset: case btOutset:
        {
            rgb_color c = m_lightEdge;
            c.red += 30; c.green += 30; c.blue += 30;
            SetHighColor (m_bevelType == btInset ? m_lightEdge : c);
            StrokeRect (Bounds());
            SetHighColor (m_darkEdge1);
            StrokeLine (m_cachedRect.LeftTop(),
                        m_cachedRect.RightTop());
            StrokeLine (m_cachedRect.LeftTop(),
                        m_cachedRect.LeftBottom());
            break;
        }
    }
    BView::Draw (updateRect);
}
//======================================================================//
```

```
void BevelView::FrameResized (float newWidth, float newHeight)
{
    // Cached drawing. Draw only when the "extra" area
    BRect newRect (Bounds());
    float minX, maxX, minY, maxY;
    // Capture the new co-ords of the "extra" rect
    minX = newRect.right > m_cachedRect.right ? m_cachedRect.right
                                   : newRect.right;
    maxX = newRect.right < m_cachedRect.right ? m_cachedRect.right
                                   : newRect.right;
    minY = newRect.bottom > m_cachedRect.bottom ? m_cachedRect.bottom
                                   : newRect.bottom;
    maxY = newRect.bottom < m_cachedRect.bottom ? m_cachedRect.bottom
                                   : newRect.bottom;
    // Draw if the rectangle is really valid
    m_cachedRect = newRect;
    if (minX != maxX)
        Invalidate (BRect (minX - 1, newRect.top, maxX, maxY));
    if (minY != maxY)
        Invalidate (BRect (newRect.left, minY - 1, maxX, maxY));
    BView::FrameResized (newWidth, newHeight);
}
//=====================================================================//
float BevelView::EdgeThickness() const
{
    return m_edgeThickness;
}
//=====================================================================//
void BevelView::AttachedToWindow ()
{
    if (Parent())
        SetViewColor (Parent()->ViewColor());
    BView::AttachedToWindow ();
}
//=====================================================================//
```

Chapter 8

Results


Teknomancer.zip
File Edit Actions Tools Settings zip Windows
New Open Close Search Extract View Add Delete

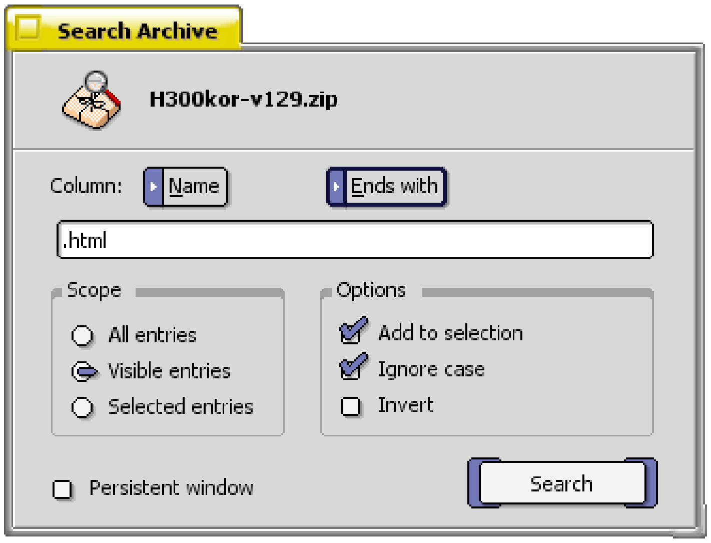
Entries: 3 of 14

Bytes: 59988356 of 276620755 (21%)

Name	Size	Packed	Ratio	Date
Megadeth				
Marty Friedman				
Tornado Of Souls - Solo (Live).wmv	12.40 MB	12.30 MB	0.8%	06/11/2005, 11:39 AM
Cheer Girl Rampage (Live).rm	2.16 MB	2.13 MB	1.5%	05/07/2005, 06:25 PM
Jeff Beck				
Nadia (Live).mpg	34.02 MB	33.27 MB	2.2%	12/12/2005, 06:11 AM
Drown In My Own Tears (Live).wmv	18.35 MB	18.28 MB	0.4%	09/12/2005, 10:14 PM
Eric Johnson				
Cliffs Of Dover (Live).mpg	42.65 MB	41.78 MB	2.0%	28/11/2005, 05:16 PM

```
Detecting format... Done.
Verifying archiver... Success.
Loading archive... Done.
```

Figure 8.1: *Archive Window*


Search Archive

H300kor-v129.zip

Column: Name | Ends with

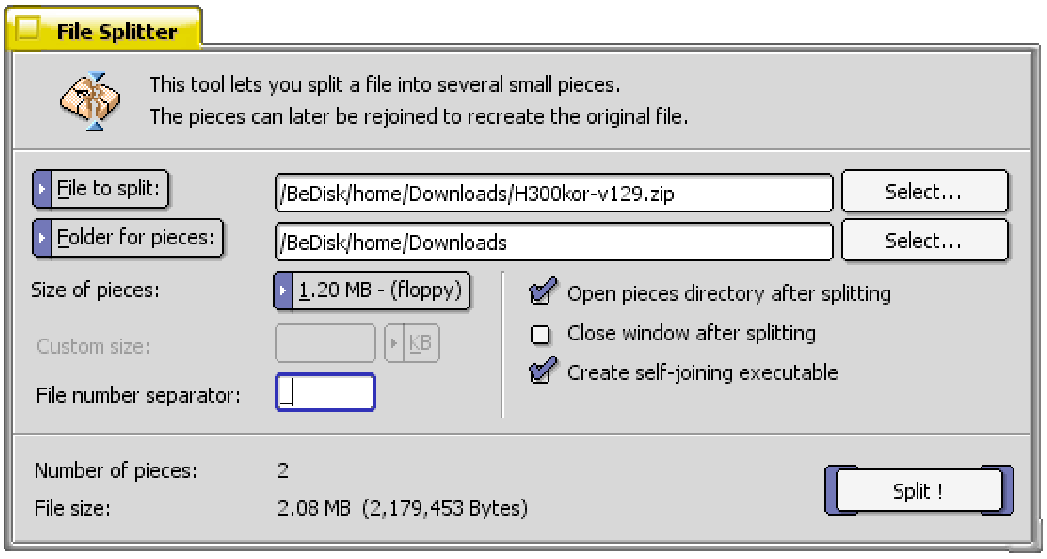
.html

Scope
- ○ All entries
- ● Visible entries
- ○ Selected entries

Options
- ✓ Add to selection
- ✓ Ignore case
- ☐ Invert

☐ Persistent window

Figure 8.2: *Search Window*


File Splitter

This tool lets you split a file into several small pieces.
The pieces can later be rejoined to recreate the original file.

File to split:	/BeDisk/home/Downloads/H300kor-v129.zip
Folder for pieces:	/BeDisk/home/Downloads
Size of pieces:	1.20 MB - (floppy)
Custom size:	KB
File number separator:	

- ✓ Open pieces directory after splitting
- ☐ Close window after splitting
- ✓ Create self-joining executable

Number of pieces:	2
File size:	2.08 MB (2,179,453 Bytes)

Figure 8.3: *File Splitter*

Chapter 9

Conclusion & Future Scope

While Beezer is full fledged archive management solution, like any other program there always exists room for improvement. This chapter briefly discusses some of the possible, probably and even some outrageous improvments that 'can' be incoroprated in Beezer.

One feature that can be added is to support for more and more formats. While Beezer supports all the popular formats, and even most of the not-so-popular ones, there are hundreds of archive formats in the world, and Beezer can strive to support as many as it can. One such format is the lzo.

Another feature is to support deep searching within the archive. Since Beezer handles all formats in a generic manner, implementing deep search is technically a complicated affair.

Despite some of these 'shortcomings' Beezer still remains the most powerful and popular archiver program available for Zeta.

Chapter 10

References

Books

1. Leslie Lamport: LaTeX *User's Guide And Reference Manual*, Second Edition, Pearson Education Asia.

2. Michel Goosens, Frank Mittelbach, Alexander Samarin: *The LaTeX Companion*, Addison-Welsey

3. Dan Parks Sydow: *Programming The Be Operating System*, O'Reilly.

4. Be, Inc.: *The Be Book (API)*, N/A.

Web Sites

1. BeBits, *http://www.bebits.com*, BeOS software and resources.

2. Wikipedia, *http://www.wikipedia.org*, Online editable encyclopedia.

3. yellowTAB, *http://www.yellowtab.com*, yellowTAB site.

Credits

The following are due credits given to all those responsible directly and indirectly for the successful completion of project Beezer. Many thanks to my Teacher and my parents for reasons that cannot be described by words. I also express my gratitude to the following people:

Development Assistance

- **7zip add-on**: Marcin Konicki

Testing Assistance

- **Rar add-on**: Miguel Guerreiro
- **Overall Testing**: Zartan, xyphn, BiPolar, Technix, tschertel, Esperantolo, Icewarp, AlienSoldier, RepairmanJack, Jack Burton, BeGeistert, SoulBender, hUMUNGUs, mmu_man, mahlzeit, disreali, Bryan, Pahtz, fyysik, Sir Mik, Ingenu, Tenzin, BGA.
- **General Feedback**: slaad, DaaT, Shackan, AnEvilYak, Dr_Evil

Artwork Assistance

- **Beezer's Icon**: Jess Tipton for the BeWicked Iconset
- **Generic Icons**: RL's Icon Collection

Beezer was designed and written by me, V. Ramshankar.

Index

Symbols

7zip 5

A

Action Log 36
Ada 17
add panel 56
add-ons 21, 29, 31
  instantiation function 29
AMD 12
applications
  Ark 7, 10, 11
  bdb 19–21
  BeDebugger 19
  BeIDE 14–16, 18–20
  CoolCat 24
  Deskbar 4
  FileTypes 47
  GCC 16–18
  GNU C Compiler 16
  Metrowerks IDE 14
  OpenTracker 12
  QuickRes 57
  SoftwareValet 32
  Tracker . 4, 5, 33, 35, 44, 47, 49, 50, 53, 54
  WinZip 1, 5–10, 52
  Zip-O-Matic 49
ar 3
archive formats
  7zip 5
  ar 3
  arj 5
  bzip2 . . . . . 5, 11
  gzip 5, 11
  lha 5
  lzh 5
  lzo 67
  rar 5
  shar 3
  tar 3, 5
  tar.bzip2 5
  tar.bzp2 11
  tar.gz 11
  tar.gzip 5
  zip 4, 5, 40
Archiver 28
`Archiver` 28, 29
archives
  adding files 46
  comments 48
  creating 40, 49
  creating folders 48
  deleting 42
  extracting 44
  information 45
  makedir 48
  searching 42
  splitting 51

testing 44
viewing files 47
arj 5
Ark 7, 10, 11

B

`BApplication` 19
Bash
scripts 14
bdb 19–21
Running Teams window 20
Team window 20
Thread window 21
Be Debugger 19
BeDebugger 19
BeIDE 14–16, 18–20
code window 15
custom files 15
debug flag 18, 19
function popup 15
project files 14
project window 14
projects 14
BeOS . 1, 2, 7, 12–14, 16, 17, 19, 22, 24, 32, 42, 48, 52
BeOS File System 32
BFS 32, 39, 40, 56
`Bitmap` 21
`BMenu` 30
breakpoint 21
`BStopWatch` 22, 23
`BWindow` 19
bzip2 . . . . . 5, 11

C

C/C++ 14, 17, 19, 25
Column 42
Columns 34
comments 48
companies
AMD 12
Free Software Foundation 16
Intel 12, 18
Metrowerks Corp. 14
Microsoft 8
Nico Mak Computing 8
WinZip Computing 8
yellowTAB 1, 68
Compile Timestamp 31
compiler 16
options 14
Concept Diagram 27
Context menu 33, 36
CoolCat 24
copyright 2
CRC 42

D

DEBUG flag 17
debugger 20, 21
breakpoint 21
function 21
debugging information 20
generating 20
size 20
Debugging Macros 17, 18
ASSERT 19
DEBUGGER 19

IS_DEBUG_ENABLED 19
PRINT 18
SERIAL_PRINT 18
SET_DEBUG_ENABLED 19
TRESPASS 19
debugging methods 22
`BStopWatch` 22
`ElapsedTime()` 23
`Lap()` 23
`Reset()` 23
`Resume()` 23
`Suspend()` 23
timing method 22
Design Goals 26
Design goals 26
Deskbar 4
directories
adding 48
Documentation Tools 24
LaTeX 24, 25, 68
TeX 24
Donald E. Knuth 24
drag 'n drop 44, 47
DVI 25

E

`ElapsedTime()` 23
ELF 52
ext3 56
extract panel 56
extracting 3

F

FAT16 32
FAT32 . . . . 30, 32, 39
File Joiner 30
File Joiner Stub 30
file permissions 3
File Splitter 30, 51
files
adding 46
viewing 47
FileType rules 30
FileTypes 47
folders
creating 48
folding level 40
Fortran 17
Free Software Foundation 16
FTP 41
Functions
`ElapsedTime()` 23
`Lap()` 23
`Reset()` 23
`Resume()` 23
`Suspend()` 23

G

GCC 16–18
debug flag 18
languages
Ada 17
C/C++ 17
Fortran 17
Java 17
Objective-C 17

GNU C Compiler 16
graphical user interface
  threads 19
gzip 5, 11

H

hash table 29
`HashTable` 29
help
  install 32
  interface 32
HTML 24, 25

I

IDE 17
  BeIDE 14
Info-ZIP 8
Infobar 33, 37, 39, 58
input server 27
instantiation function 29
Intel 12, 18
Interface components
  Action Log 36
  Column 42
  Columns 34
  Context menu 33, 36
  Infobar 33, 37, 39, 58
  List View 42, 47
  Log View 36, 39
  LogView 33
  Menubar 33, 38
  Splitter 36
  Toolbar 33, 36, 39, 58
  Tree View 33, 34, 39, 56
  TreeView 33
interface panel 58
Internet 30
ISO
  GCC (ISO/IEC 9899:1990) 16

J

Java 14, 17, 19

L

`Lap()` 23
LaTeX 24, 25, 68
lha 5
libFSU.so 30
libraries
  dynamic 14
  static 14
linker
  options 14
links 50
Linux 7, 10
List View 42, 47
`ListEntry` 29
locale 2
Log View 36, 39
LogView 33
lossless 3
lzh 5
lzo 67

M

MacOS 16

makefile . . . 17, 19
  debug flag . . . 18
Menubar . . . 33, 38
Metafont . . . 25
Metrowerks
  IDE . . . 14
Metrowerks Corp. . . . 14
Metrowerks IDE . . . 14
Microsoft . . . 8
MIME . . . 15
mimetype . . . 30
miscellaneous panel . . . 59
mp3 . . . 41
multi-threading . . . 19, 27

N

Nico Mak Computing . . . 8
NTFS . . . 56

O

Objective-C . . . 17
Objects
  `BApplication` . . . 19
  `BMenu` . . . 30
  `BStopWatch` . . . 22, 23
  `BWindow` . . . 19
  `Bitmap` . . . 21
Objects, Beezer
  `Archiver` . . . 28, 29
  `HashTable` . . . 29
  `ListEntry` . . . 29
  `PipeMgr` . . . 28, 29
OpenTracker . . . 12
OS.h . . . 21
ownership . . . 3

P

Pascal . . . 19
password . . . 38
paths panel . . . 57
PDF . . . 25
`PipeMgr` . . . 28, 29
PKZIP . . . 8
pointer . . . 21
PostScript . . . 25
Preferences . . . 44, 54
  add . . . 56
  extract . . . 56
  interface . . . 58
  miscellaneous . . . 59
  panels . . . 54
  paths . . . 57
  recent . . . 58
  settings file . . . 55
  state . . . 56
process . . . 20

Q

QuickCreate . . . 49
QuickRes . . . 57

R

RAM . . . 12
rar . . . 5
recent panel . . . 58

recycle bin . . . 42
requirements
  hardware . . . 12
  software . . . 12
`Reset()` . . . 23
resources . . . 14
`Resume()` . . . 23

S

segment violation . . . 21
self joining executable . . . 52
shar . . . 3
software tools
  build tools . . . 14
  test tools . . . 17
SoftwareValet . . . 32
Splitter . . . 36
stack . . . 21
  pop . . . 23
state panel . . . 56
`Suspend()` . . . 23
symbolic links . . . 50

T

tar . . . 3, 5
tar.bzip2 . . . 5
tar.bzp2 . . . 11
tar.gz . . . 11
tar.gzip . . . 5
Taskbar . . . 4
teams . . . 20
TeX . . . 24
threads . . . 19–21
timestamp . . . 3
timing method . . . 22
Toolbar . . . 33, 36, 39, 58
Tracker . . 4, 5, 33, 35, 44, 47, 49, 50, 53, 54
Trash . . . 42
Tree View . . . 33, 34, 39, 56
TreeView . . . 33

U

unarchiving . . . 3
Unix . . . 3
unpacking . . . 3

V

vfat . . . 56

W

While Opening . . . 40
wildcards . . . 29
Windows . . . 4, 7, 8, 16, 40, 52
  Archive . . . 33
  Archive Info . . . 45
  Comments . . . 48
  Create Folder . . . 48
  File Splitter . . . 51
  FileJoiner . . . 53
  Preferences . . . 33, 54
  QuickCreate Archive . . . 49
  Search . . . 42
  Welcome . . . 32
WinZip . . . 1, 5–10, 52

WinZip Computing 8
Workers 31
workers 50

X

XML 24

Y

yellowTAB 1, 68

Z

Zeta 1, 2, 7, 12–14, 17, 48, 67
zip 4, 5, 40
Zip-O-Matic 49
zipinfo 48