Sarawak Open Source Competition, 2006

File Injection and Retrieval Application version 2 (FIRA2)

Project Summary

Written by:

Full name: Lai Chiong Ching

Nickname: LCC Corps

I.C. number: 850211-13-5185

Postal address: 9B, Lorong Merdeka 15, 96000 Sibu, Sarawak

Email: aurora.lai@gmail.com

Contact number: 019-888-1892 (HP)

Institution name: Curtin University of Technology, Miri, Sarawak

Table of content

Author’s details

Full name: Lai Chiong Ching
I.C. number: 850211-13-5185
Postal address: 9B, Lorong Merdeka 15, 96000 Sibu, Sarawak
Secondary address: Lot 2385, Lutong Baru, 98100 Miri, Sarawak
Email: aurora.lai@gmail.com
Contact number: 019-888-1892 (HP)
Institution name: Curtin University of Technology, Miri, Sarawak

1. Software license

These project deliverables, including the program source code, the program itself and all the accompanying documentations are protected under the open source license, The GNU General Public License (GPL). This project is free and there is NO ABSOLUTE WARRANTY for it. Anyone may redistribute this program under The GNU General Public License (GPL)

2. Name of software

The name of the software has been chosen to be “File Injection and Retrieval Application version 2” or in short “FIRA2”. The program can inject or hide a file into a standard bitmap file and safely retrieve it in later time.

3. Project description

This project is a 3-person-month software development in May-July, 2006 for the Sarawak Open Source Competition. The deliverables of this project includes the working system together with its source code and all relevant documentation.

The software developed has the capability of hiding/injecting a file into a standard bitmap file. Extra features like file compression and encryption are planned to be developed as part of the file hiding process. Additionally, the software will also be able to retrieve back the hidden file, including extra features like file decryption and decompression. The main ‘key’ used to inject and retrieve the file is an alpha-numeric password.

The documentations for this project include the Software Requirement Specification, Software Design Document, Software Test Document as well as the source code itself.

4. Brief history

The first version of this program (called “FIRA”) had been developed part-timely in December 2005 – January 2006. This first version of the program is developed mainly for personal interest and educational purpose. It has not been participated in any competition.

The second version of this program (called “FIRA2”) is developed based on the previous version. The main purpose of developing “FIRA2” is to add some extra features to the program, to increase its efficiency and to reduce code redundancy.

5. Main features and functionalities

The first version of FIRA simply injects or hides a file into a standard bitmap file. As a result, the hidden file may not be very safe from intruders and the hidden file size is limited to one tenth of the carrier bitmap file.

However, as of version 2, FIRA2 has been added with some extra features, namely file compression/decompression as well as file encryption/decryption. With file compression/decompression, bigger file can be hidden in standard bitmap file by first compressing the file before it is hidden into the carrier file. This feature is most visible especially while hiding file with very high compression rate (e.g. standard text file).

On the other hand, the file encryption/decryption capability is added to increase the security of the hidden file, which is the main purpose of this program. This file encryption together with the algorithm to inject the file will make it harder for any unauthorized person to crack and recover the hidden file.

6. Future enhancements

The file encryption/decryption algorithm used in FIRA2 is fixed and there is no way for user to choose their preferred encryption algorithm. Hence, for future enhancement, a list of available cipher and key generator algorithm can be developed for user to select their desired algorithm.

Another enhancement which may improve the GUI of this program is to include an image preview component for file chooser dialog when user is choosing the carrier bitmap file.

7. Innovative technology used

In version 1, FIRA had been developed merely for injecting a file into a standard bitmap file. The innovative algorithm used is to securely inject a data file into a bitmap file is by utilizing:

1. Randomly select (based on password entered) the starting location within the carrier bitmap file to inject the data file
2. When injecting stream of bytes from data file into the carrier file, a blank bit is inserted into the stream randomly (with fixed probability) to increase the security of the injected data file

As of version 2, FIRA2 has been developed with additional innovative algorithm such as:

1. Increased file injection capacity
    a. by using file compression/decompression on the data file
    b. by injecting TWO (optimum value) bits of data into one byte of carrier file, rather than only injecting one bit per byte as in FIRA version 1.

2. Increased injected data file security by utilizing file encryption/decryption on the data file. .

File Injection and Retrieval Application version 2 (FIRA2)

Software Requirement Specification (SRS)

<u>Written by:</u>

Full name: Lai Chiong Ching

Nickname: LCC Corps

Last updated: May 14, 2006

Table of content

1. Purpose

The purpose of this document is to provide all the requirements for FIRA2. This document primarily discuss about the C-requirement, which is particularly for end-user, but may also be of interest for programmers.

2. Scope

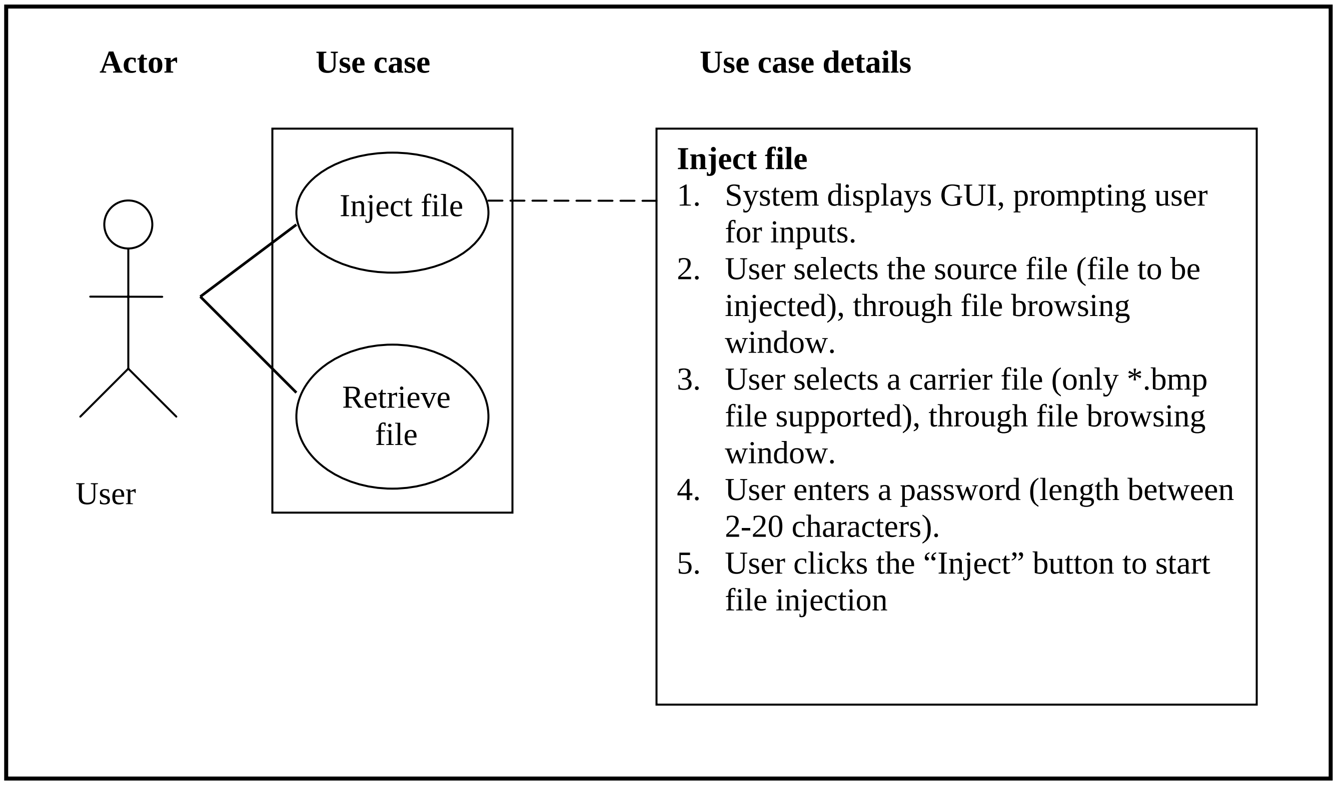
This document will only cover the requirements in program functions, user interface and security level.

3. Overview

FIRA2 is to enable any file (referred as "data file") to be injected or hidden into a standard bitmap file (referred as "carrier file"), as well as to retrieve a hidden file from a carrier file. It must be able to ensure that the given data file can be injected into the given carrier file first before actually performing the injection process. Similarly, FIRA2 must be able to detect if a hidden file exists in the given carrier file first before trying to retrieve the file.

4. Product perspective

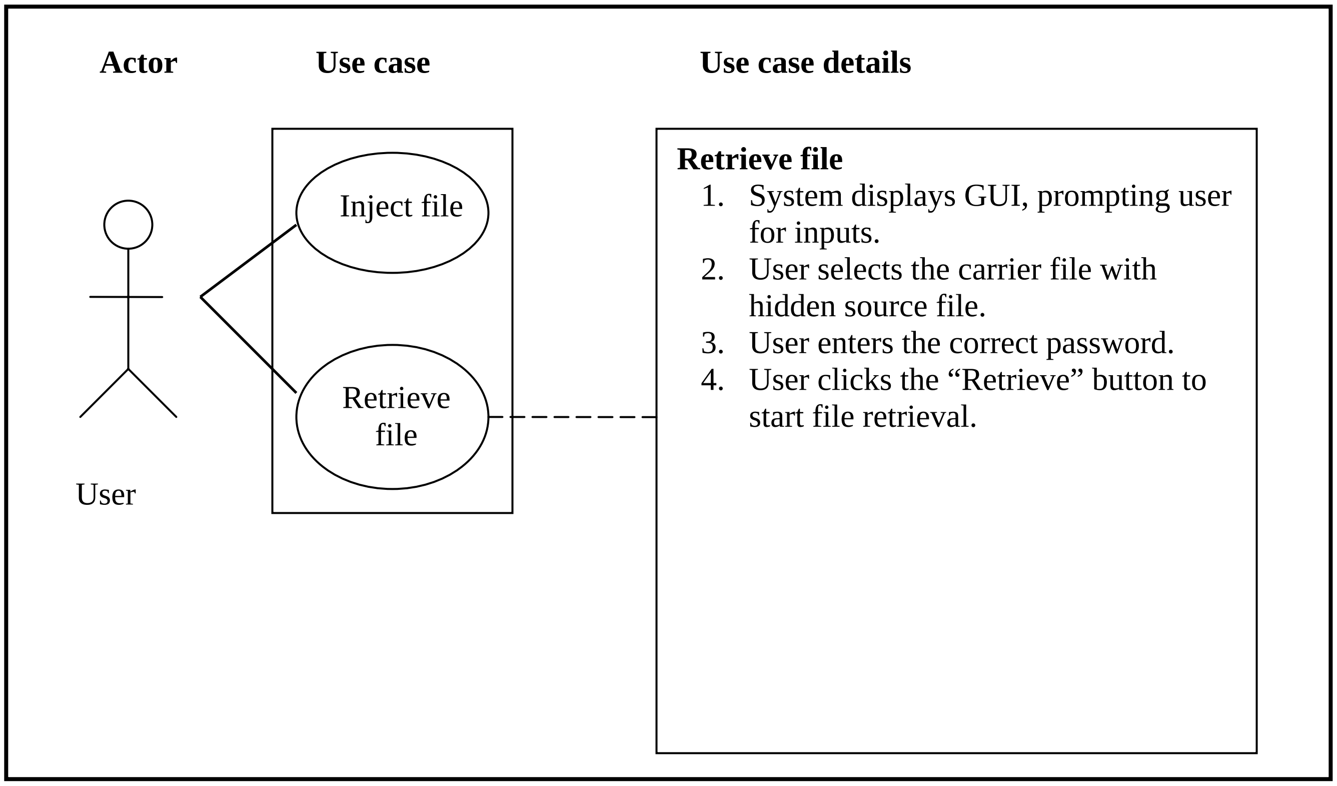
4.1 Software functions



Inject file

1. System displays GUI, prompting user for inputs.
2. User selects the source file (file to be injected), through file browsing window.
3. User selects a carrier file (only *.bmp file supported), through file browsing window.
4. User enters a password (length between 2-20 characters).
5. User clicks the "Inject" button to start file injection

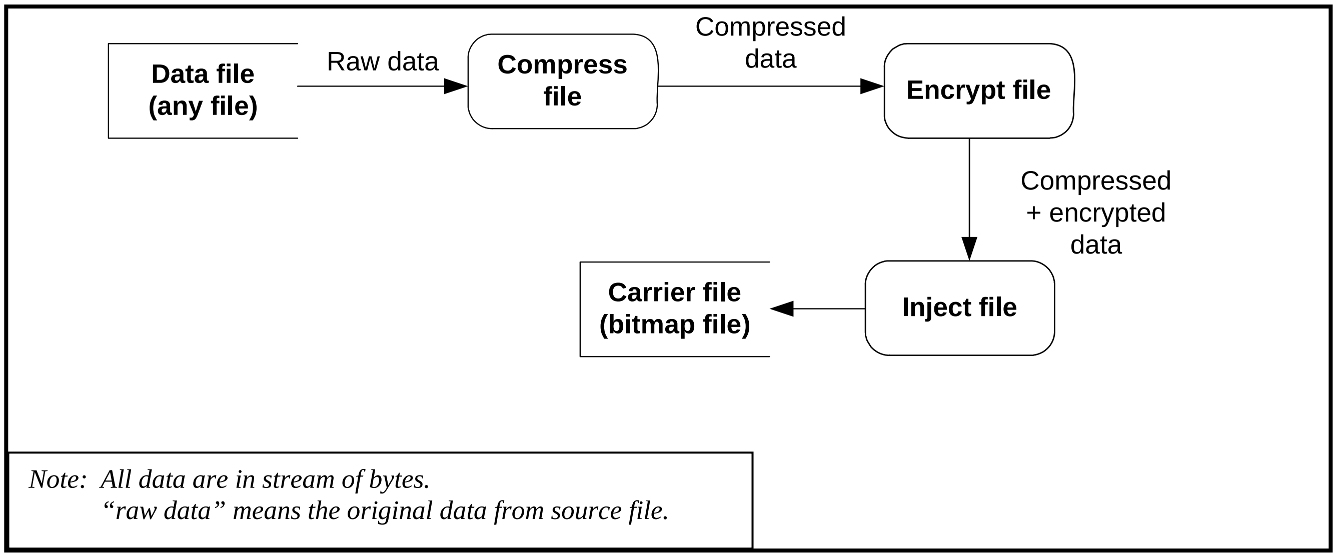
Figure SRS.1 – Use-case description for file injection



Retrieve file

1. System displays GUI, prompting user for inputs.
2. User selects the carrier file with hidden source file.
3. User enters the correct password.
4. User clicks the "Retrieve" button to start file retrieval.

Figure SRS.2 – Use-case description for file retrieval



Note: All data are in stream of bytes.
"raw data" means the original data from source file.

Figure SRS.3 – Data-flow diagram for file injection

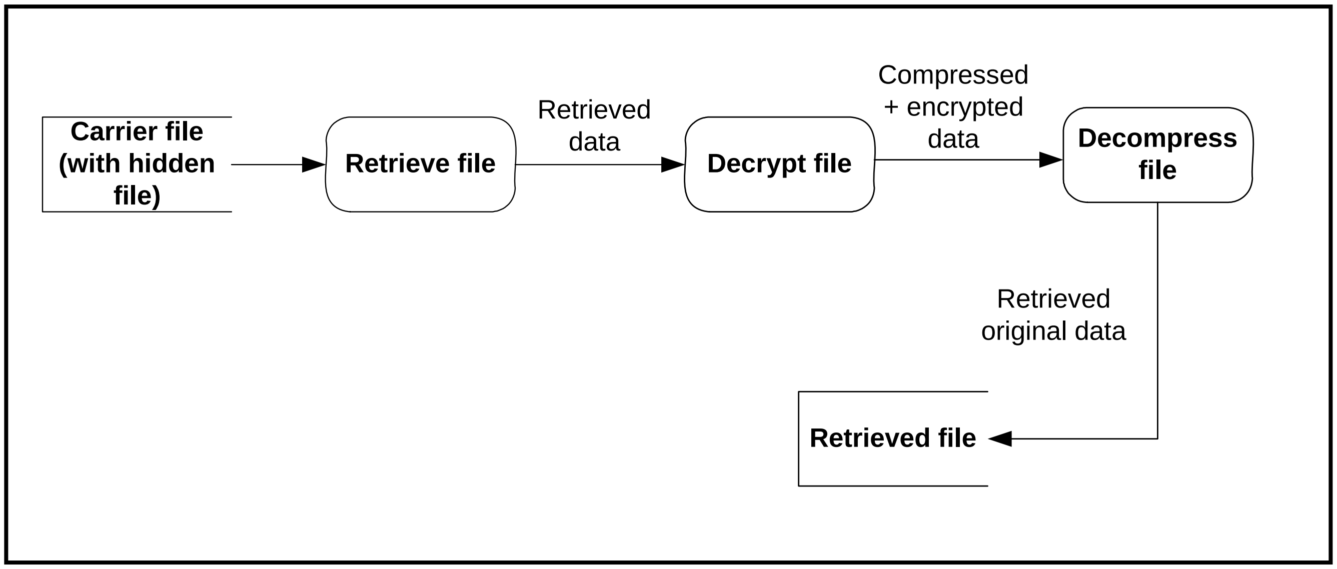


Figure SRS.4 – Data-flow diagram for file retrieval

In the initial design, it has been made a requirement that the user has the option to enable file compression or file decryption or both during the file injection process. Then, the file retrieval process will automatically determined if the injected file is being encrypted or compressed and perform appropriate file decryption and decompression to retrieve the original data file. The rationale for this requirement is to speed up the file injection/retrieval process by not performing file compression/decompression and/or file encryption/decryption.

However, since the file compression/decompression as well as file encryption/decryption is very fast compared to the file injection/retrieval itself, the requirement has been removed. That is, the user no longer has the option to enable/disable file compression and/or file encryption. The data file will always be compressed and encrypted before it is injected into a carrier bitmap file.

4.2 File security

It is required that the hidden data file inside a carrier file is secured from unauthorized person (i.e. the person that is not meant to have access or password to the hidden data file). Hence, there must be a built-in file encryption or custom file injection algorithm to increase the hidden data file security.

Besides, the carrier file with a hidden data file must not differ too much from it original file/picture (without the hidden data file). In other words, both the same carrier file (one with hidden data file, and the other without) must look exactly the same in human naked eyes. This is to prevent outsiders from being able to differentiate between the original carrier file and the one with hidden data file.

4.3 User interface

To promote user-friendliness, GUI has been designed and implemented in FIRA2. The GUI layouts are as shown below. The functions of each components in the GUI is further explained in Software Design Document, Section 6.2.

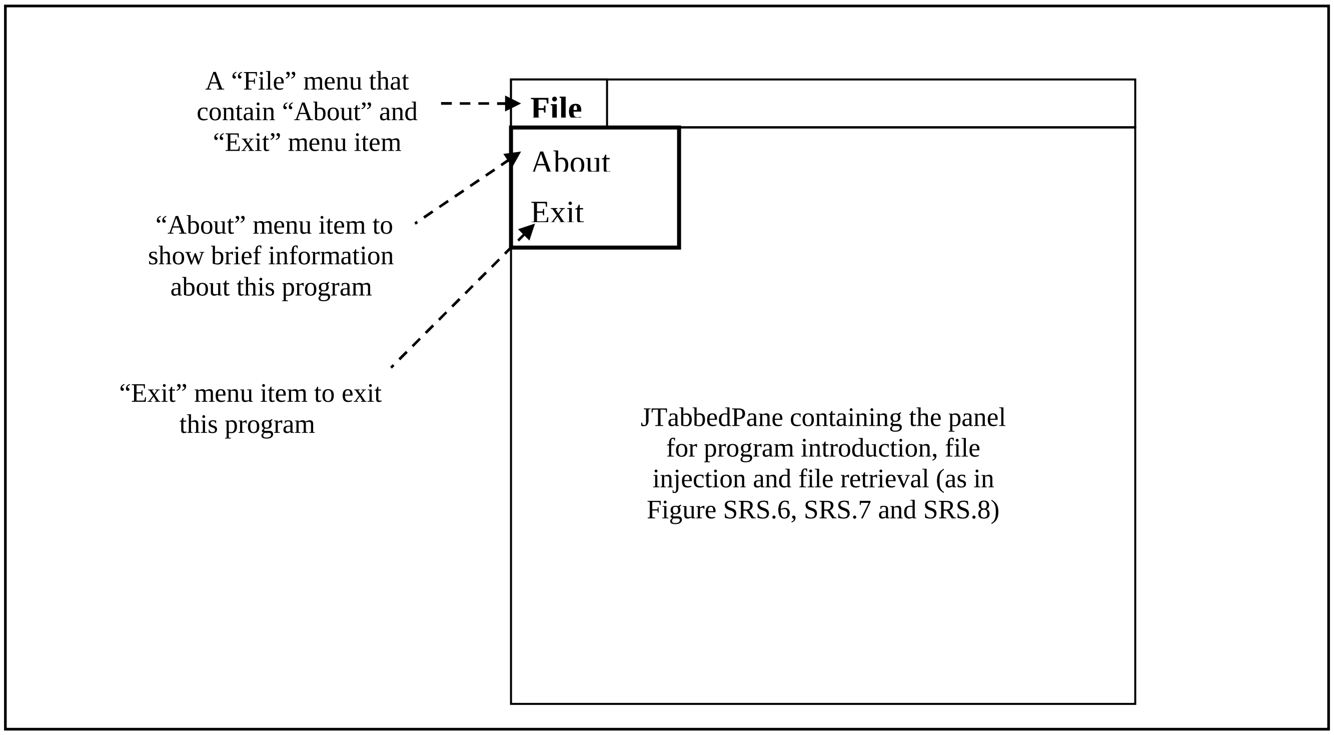

A "File" menu that contain "About" and "Exit" menu item
File
About
Exit
"About" menu item to show brief information about this program
"Exit" menu item to exit this program
JTabbedPane containing the panel for program introduction, file injection and file retrieval (as in Figure SRS.6, SRS.7 and SRS.8)

Figure SRS.5 – GUI layout for the program main frame

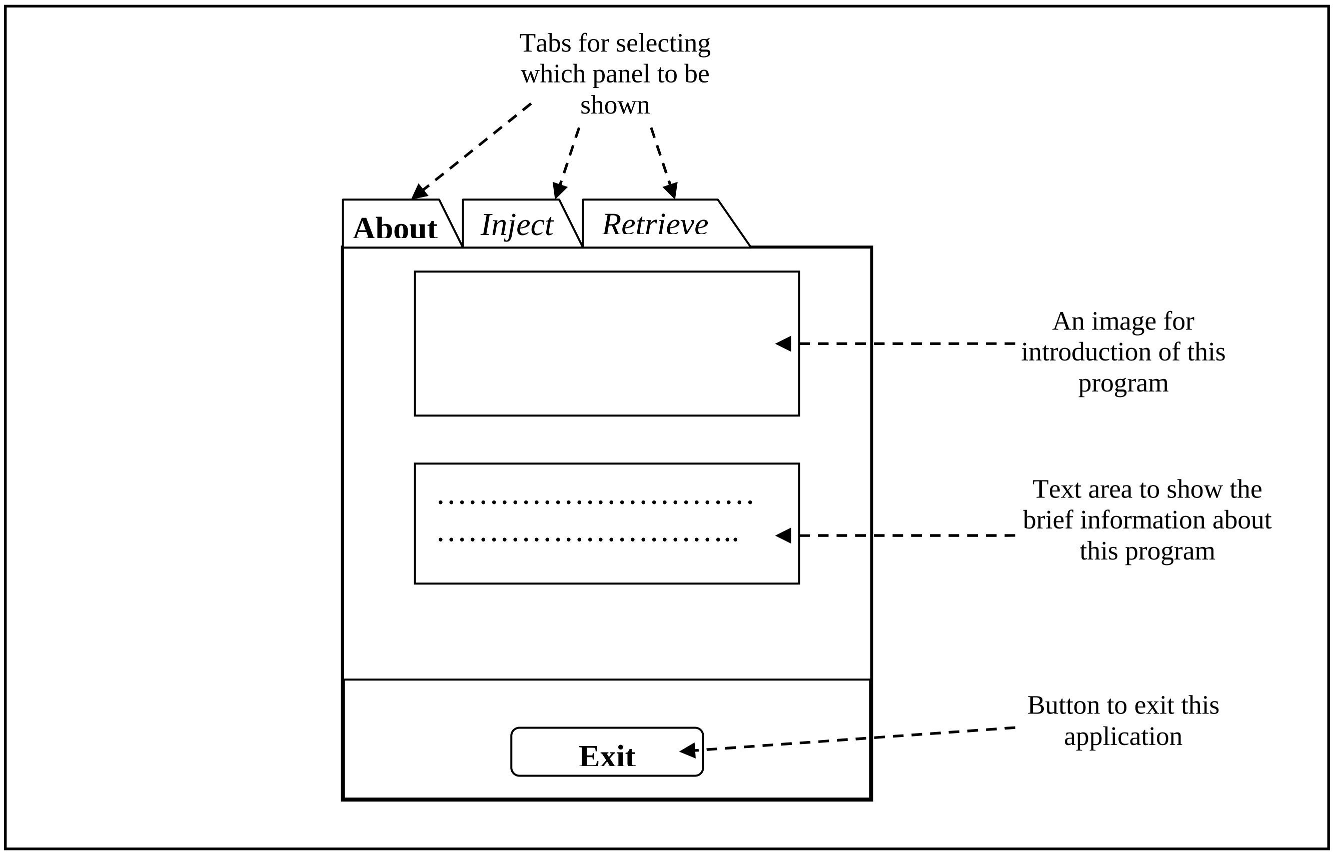

Tabs for selecting which panel to be shown
About
Inject
Retrieve
An image for introduction of this program
Text area to show the brief information about this program
Exit
Button to exit this application

Figure SRS.6 – GUI layout showing the "About" panel

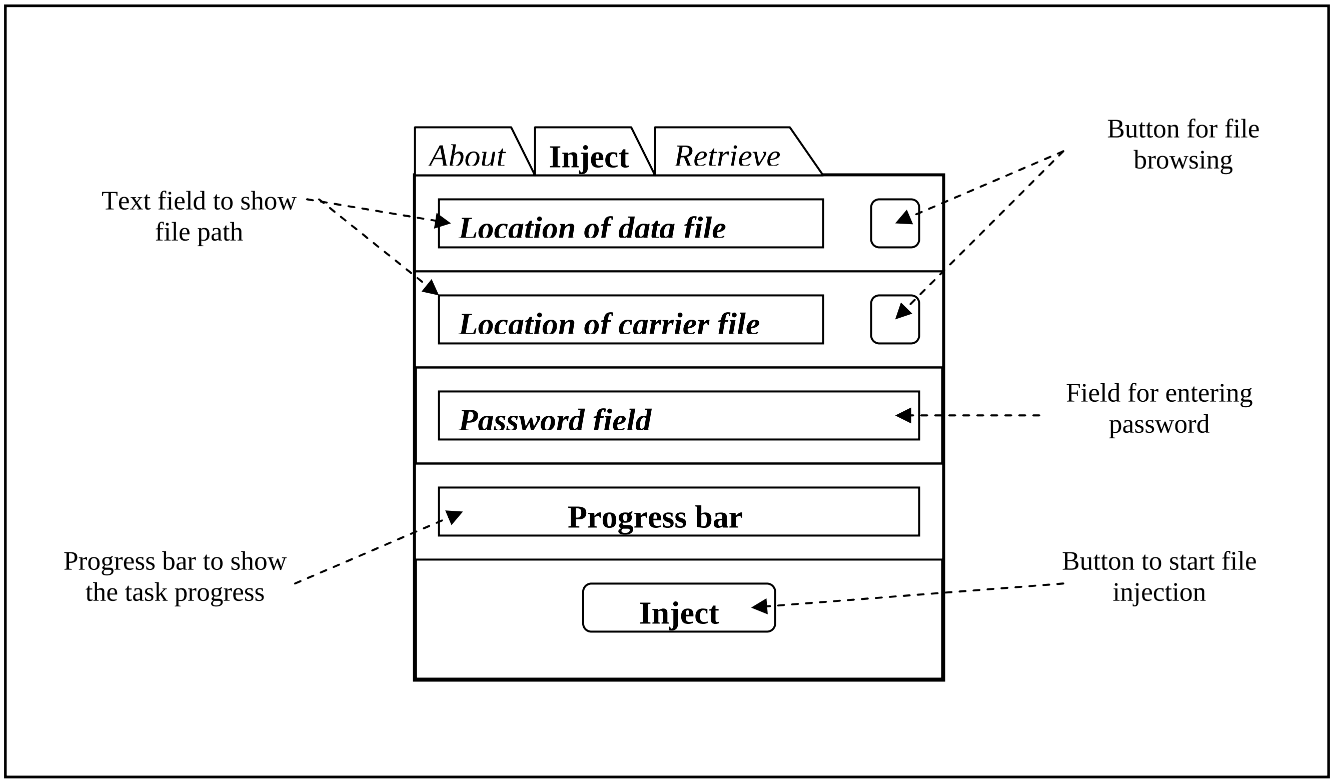

About
Inject
Retrieve
Button for file browsing
Text field to show file path
Location of data file
Location of carrier file
Password field
Field for entering password
Progress bar
Progress bar to show the task progress
Button to start file injection
Inject

Figure SRS.7 – GUI layout showing the "Inject" (for file injection) panel

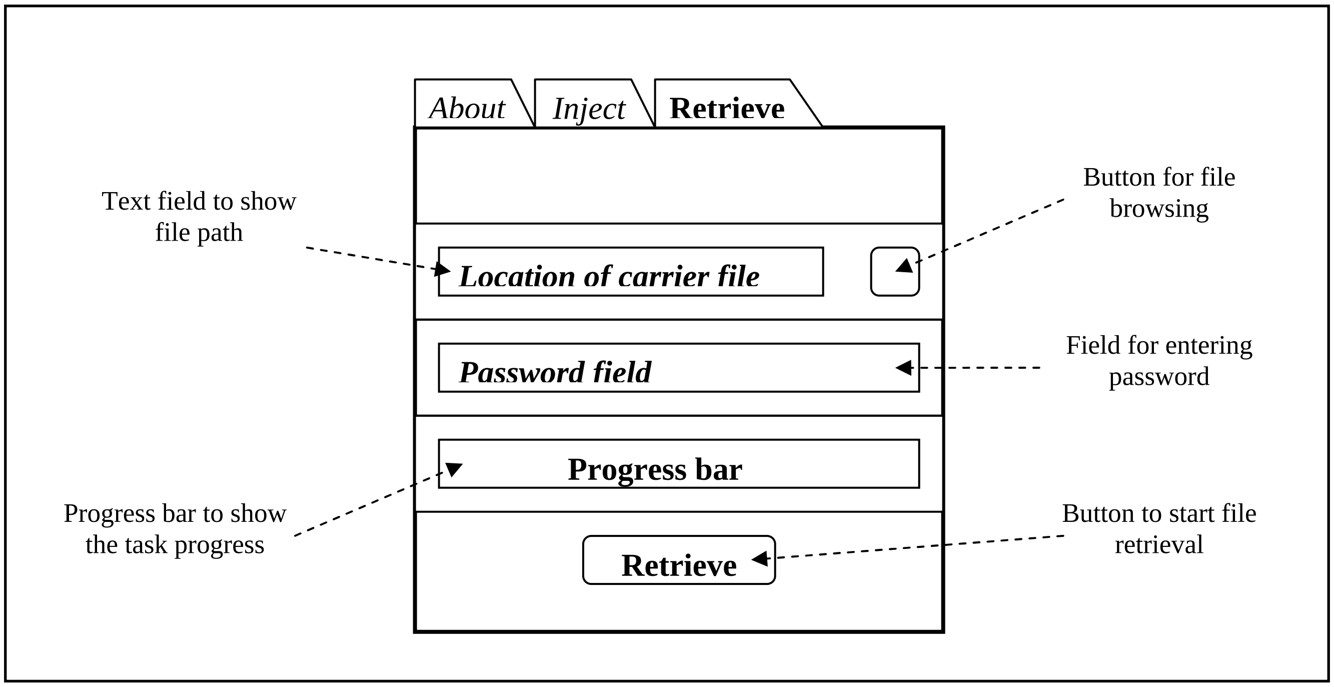

About
Inject
Retrieve
Button for file browsing
Text field to show file path
Location of carrier file
Password field
Field for entering password
Progress bar
Progress bar to show the task progress
Button to start file retrieval
Retrieve

Figure SRS.8 – GUI layout showing the "Retrieve" (for file retrieval) panel

5. User characteristics

The target user for FIRA2 is people with basic computer knowledge who at least know how to interact with computer using keyboard and mouse. They need not be very intellect in computer as a user-friendly GUI has been designed for FIRA2.

6. Constraint and dependencies

FIRA2 has been developed in Window XP Professional Edition. As a result, there is no strong verification that it may work as expected in some legacy windows. Besides, there is no guarantee that FIRA2 will work in other platform, like Linux or Mac.

File Injection and Retrieval Application version 2 (FIRA2)

Software Design Document (SDD)

Written by:

Full name: Lai Chiong Ching

Nickname: LCC Corps

Last updated: June 11, 2006

Table of content

1. Narrative description

In its most simplest form, during file injection, FIRA2 will, in order, compress the original data file, then encrypt the compressed file and finally inject the compressed and encrypted file into the given carrier (bitmap) file. The file retrieval process is exactly the reverse of the file injection process: retrieve the hidden file, then, decrypt the retrieved file and finally decompress the retrieved and decrypted file.

Both the file compression/decompression as well as file cryptography algorithm are being handled by Java built-in packages. Only the file injection and retrieval algorithm is being developed as there is no relevant built-in package for it. Hence, the algorithm will be discussed in this section.

1.1 Data injection

An example will be provided as aid in data injection explanation. Consider it is required to inject a String "abc" (without the double quotes) into a carrier file having its contents as shown in Table 1 in hexadecimal representation.

Table SDD.1 – The original contents of carrier file before any modification

Address	**Data**
`00000000`	`42 4d 36 4b 00 00 00 00 00 00 36 00 00 00 28 00`
`00000010`	`00 00 50 00 00 00 50 00 00 00 01 00 18 00 00 00`
`00000020`	`00 00 00 4b 00 00 00 00 00 00 00 00 00 00 00 00`
`00000030`	`00 00 00 00 00 00 ff 6f de 98 9d 1f 10 99 29 96`
`00000040`	`32 4e dd ee ba ff ff ff ff ff ff ff ff ff ff ff`
`...`	`...`

The ASCII code for each character in the want-to-be-injected String is given in Table 2.

Table SDD.2 – The ASCII code for the characters 'a', 'b', and 'c'.

Unicode character	ASCII code		
	Decimal	Hexadecimal	Binary
a	97	0x61	%0110 0001
b	98	0x62	%0110 0010
c	99	0x63	%0110 0011

The data injection process will proceed byte by byte. In this example, the character 'a' will be injected first and is shown in the following figures. Assume that the character 'a' is to be injected starting at location 0x056 (its value is red highlighted in Table 1).

Data/byte need to be injected: %0110 0001

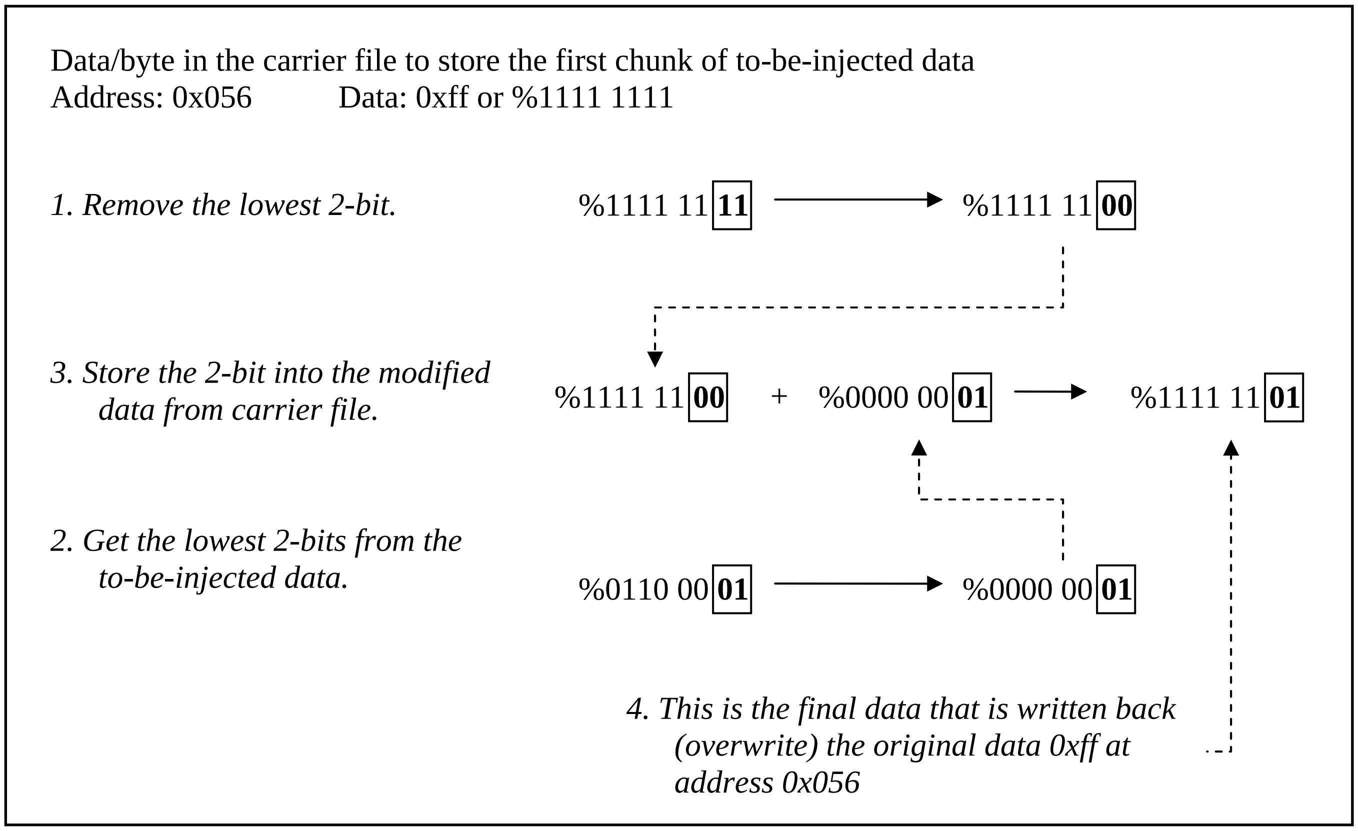

Data/byte in the carrier file to store the first chunk of to-be-injected data
Address: 0x056 Data: 0xff or %1111 1111
1. Remove the lowest 2-bit.
%1111 11 **11**
%1111 11 **00**
3. Store the 2-bit into the modified data from carrier file.
%1111 11 **00** + %0000 00 **01**
%1111 11 **01**
2. Get the lowest 2-bits from the to-be-injected data.
%0110 00 **01**
%0000 00 **01**
4. This is the final data that is written back (overwrite) the original data 0xff at address 0x056

Figure SDD.1 – Injecting a byte

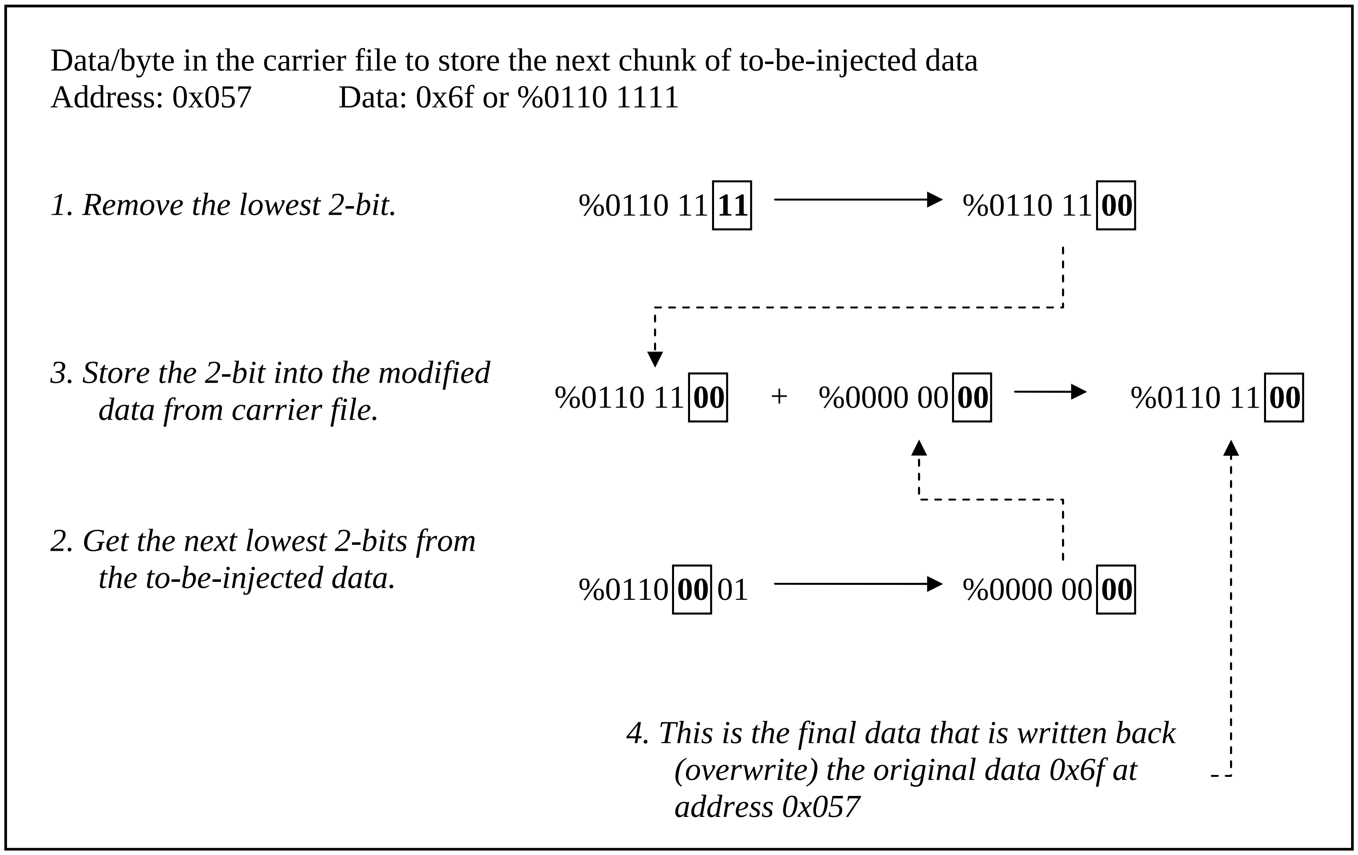
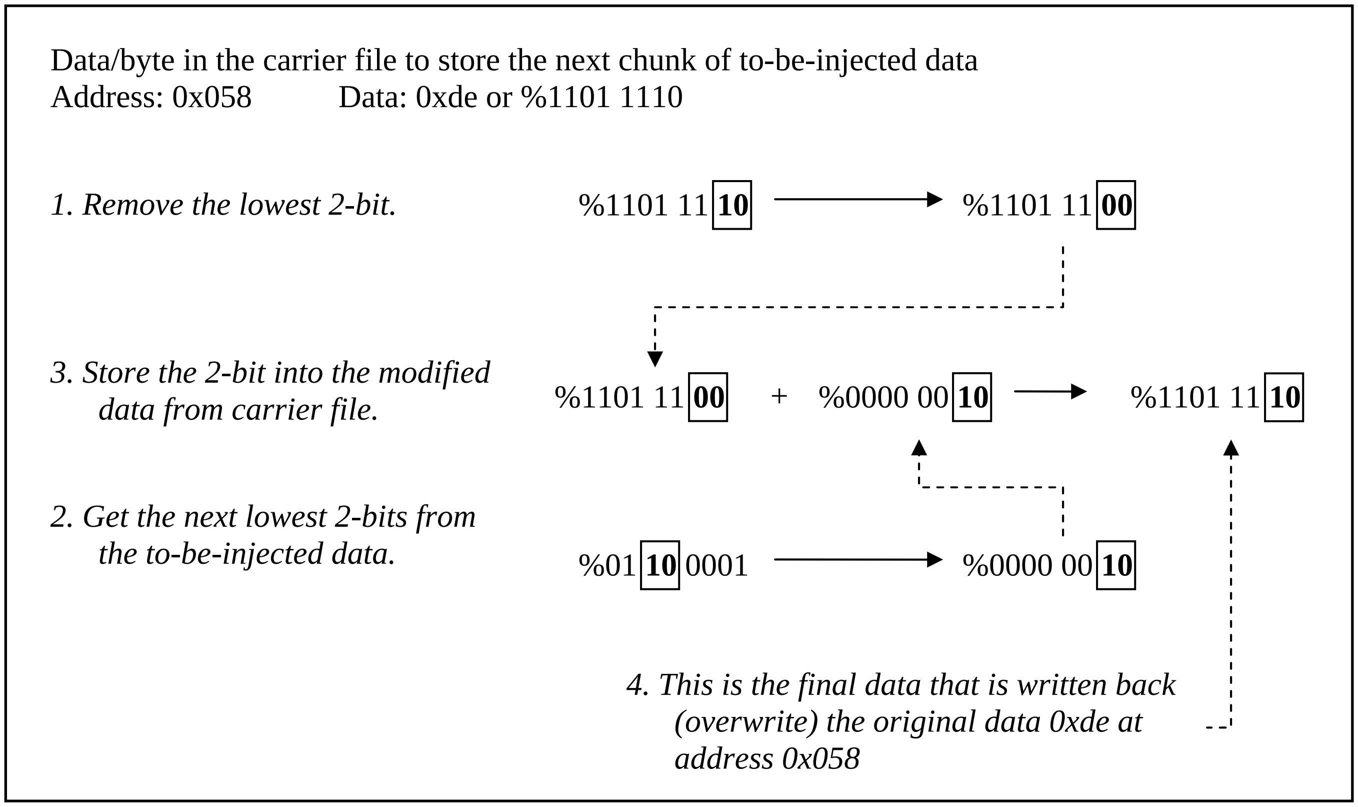

Data/byte in the carrier file to store the next chunk of to-be-injected data
Address: 0x057 Data: 0x6f or %0110 1111
1. Remove the lowest 2-bit.
%0110 11 **11**
%0110 11 **00**
3. Store the 2-bit into the modified data from carrier file.
%0110 11 **00** + %0000 00 **00**
%0110 11 **00**
2. Get the next lowest 2-bits from the to-be-injected data.
%0110 **00** 01
%0000 00 **00**
4. This is the final data that is written back (overwrite) the original data 0x6f at address 0x057

Figure SDD.2 – Injecting a byte (cont')



Data/byte in the carrier file to store the next chunk of to-be-injected data
Address: 0x058 Data: 0xde or %1101 1110

1. Remove the lowest 2-bit.

$\%1101\ 11\boxed{\mathbf{10}} \longrightarrow \%1101\ 11\boxed{\mathbf{00}}$

3. Store the 2-bit into the modified data from carrier file.

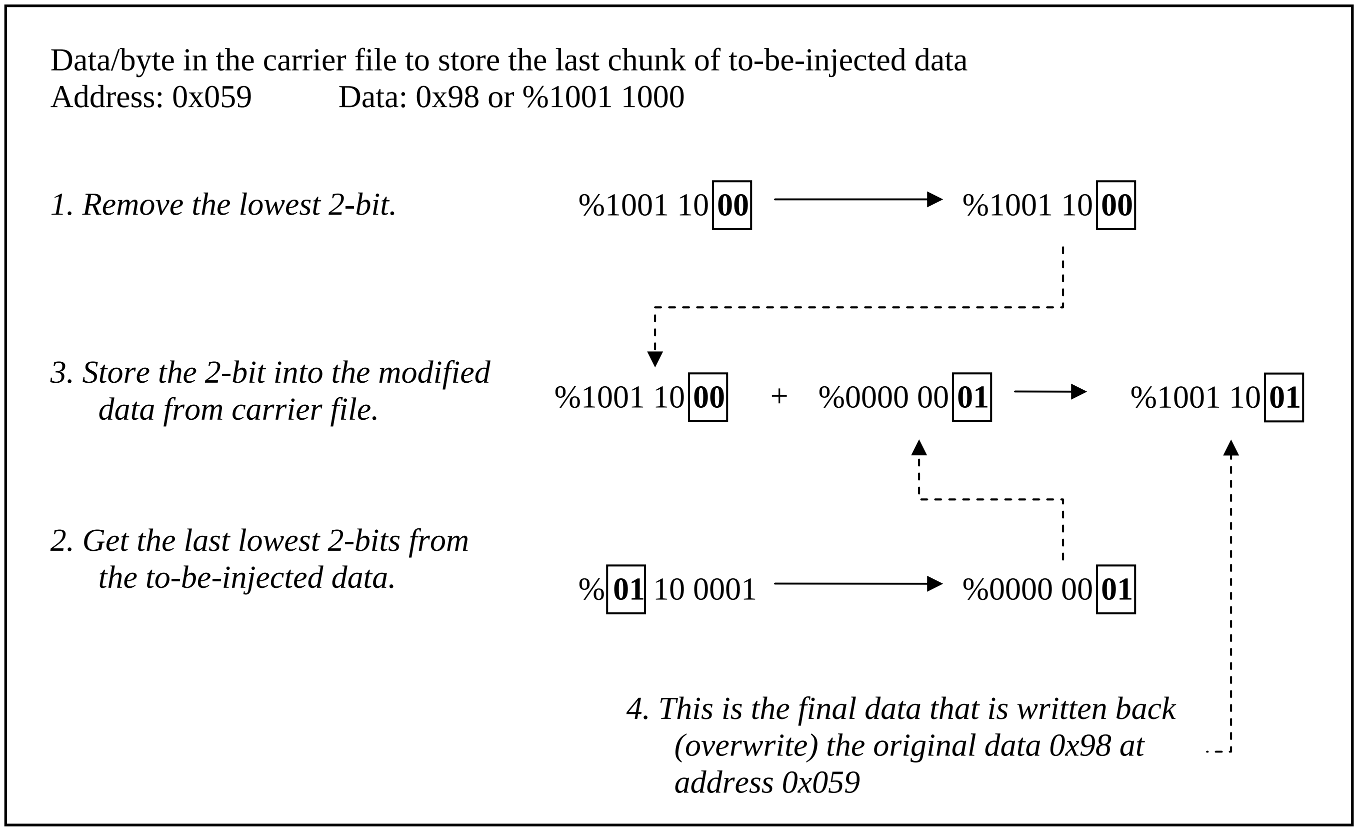
$\%1101\ 11\boxed{\mathbf{00}} + \%0000\ 00\boxed{\mathbf{10}} \longrightarrow \%1101\ 11\boxed{\mathbf{10}}$

2. Get the next lowest 2-bits from the to-be-injected data.

$\%01\boxed{\mathbf{10}}0001 \longrightarrow \%0000\ 00\boxed{\mathbf{10}}$

4. This is the final data that is written back (overwrite) the original data 0xde at address 0x058

Figure SDD.3 – Injecting a byte (cont')



Data/byte in the carrier file to store the last chunk of to-be-injected data
Address: 0x059 Data: 0x98 or %1001 1000

1. Remove the lowest 2-bit.

$\%1001\ 10\boxed{\mathbf{00}} \longrightarrow \%1001\ 10\boxed{\mathbf{00}}$

3. Store the 2-bit into the modified data from carrier file.

$\%1001\ 10\boxed{\mathbf{00}} + \%0000\ 00\boxed{\mathbf{01}} \longrightarrow \%1001\ 10\boxed{\mathbf{01}}$

2. Get the last lowest 2-bits from the to-be-injected data.

$\%\boxed{\mathbf{01}}10\ 0001 \longrightarrow \%0000\ 00\boxed{\mathbf{01}}$

4. This is the final data that is written back (overwrite) the original data 0x98 at address 0x059

Figure SDD.4 – Injecting a byte (last)

As can be seen, injecting a byte with 2-bits per byte will require 4-bytes of carrier file to accommodate 1-byte of data file. The same procedure applies for injecting the remaining characters ‘b’ and ‘c’.

After the injection of all the three characters ‘a’, ‘b’, and ‘c’, the contents of carrier file is as shown in Table 3.

Table SDD.3 – The contents of carrier file with injected text “abc”. The modified contents are highlighted in yellow.

Address	Data
00000000	42 4d 36 4b 00 00 00 00 00 00 36 00 00 00 28 00
00000010	00 00 50 00 00 00 50 00 00 00 01 00 18 00 00 00
00000020	00 00 00 4b 00 00 00 00 00 00 00 00 00 00 00 00
00000030	00 00 00 00 00 00 fd 6c de 99 9e 1c 12 99 2b 94
00000040	32 4d dd ee ba ff ff ff ff ff ff ff ff ff ff ff
...	...

1.2 Data retrieval

The data retrieval process is exactly the reverse of the data injection process. For example, if it is required to retrieve back the injected character ‘a’, the steps are:

1. Get four consecutive bytes from carrier file: fd 6c de 99
2. Retrieve the lowest 2 bits from each byte: 01 00 10 01
3. Rebuild/concatenate back the retrieved data in appropriate order to get back %0110 0001

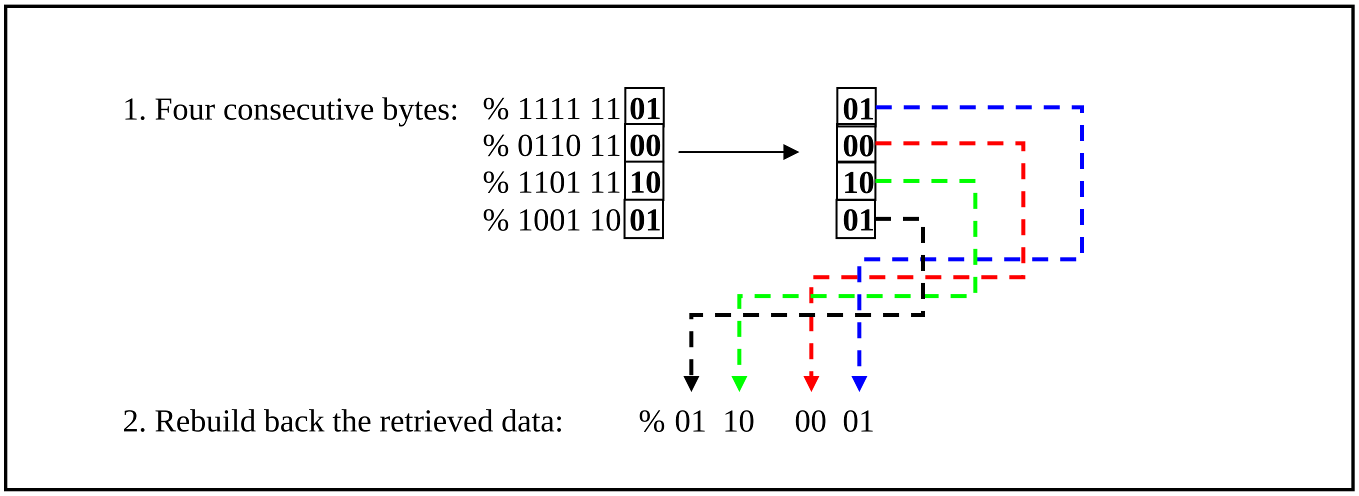


Figure SDD.5 – Retrieving a byte

Similar process apply to retrieving characters ‘b’, and ‘c’.

1.3 Overall injection process[1]

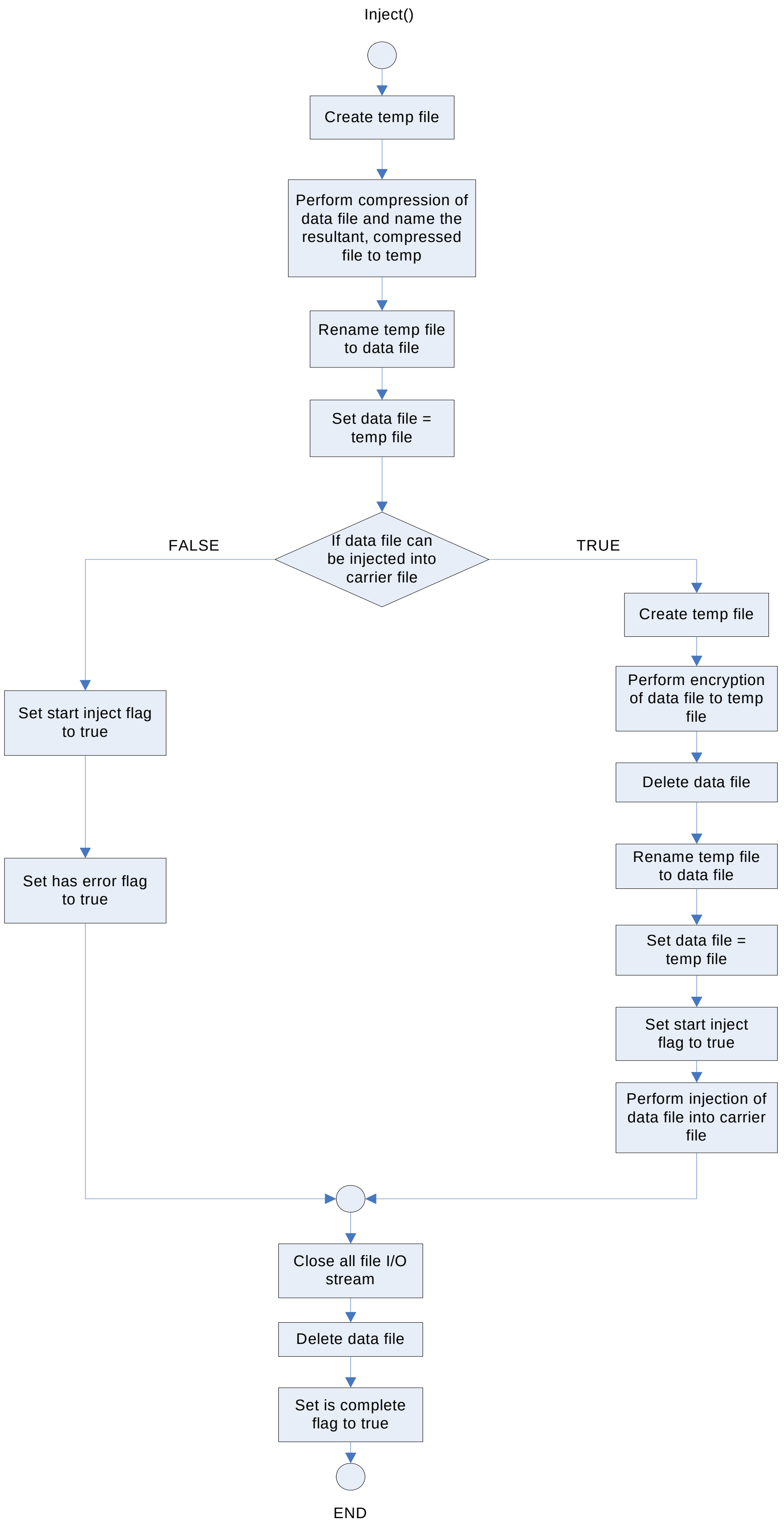
← first 60-bytes, header file + allocated margin (untouched) → - - - ← (data continue here if not finish before ‘END’) -
E E E x x x x x x x x x x x x x x x x x x x x x S < - - -filename - - - > F < -data - - - - - END

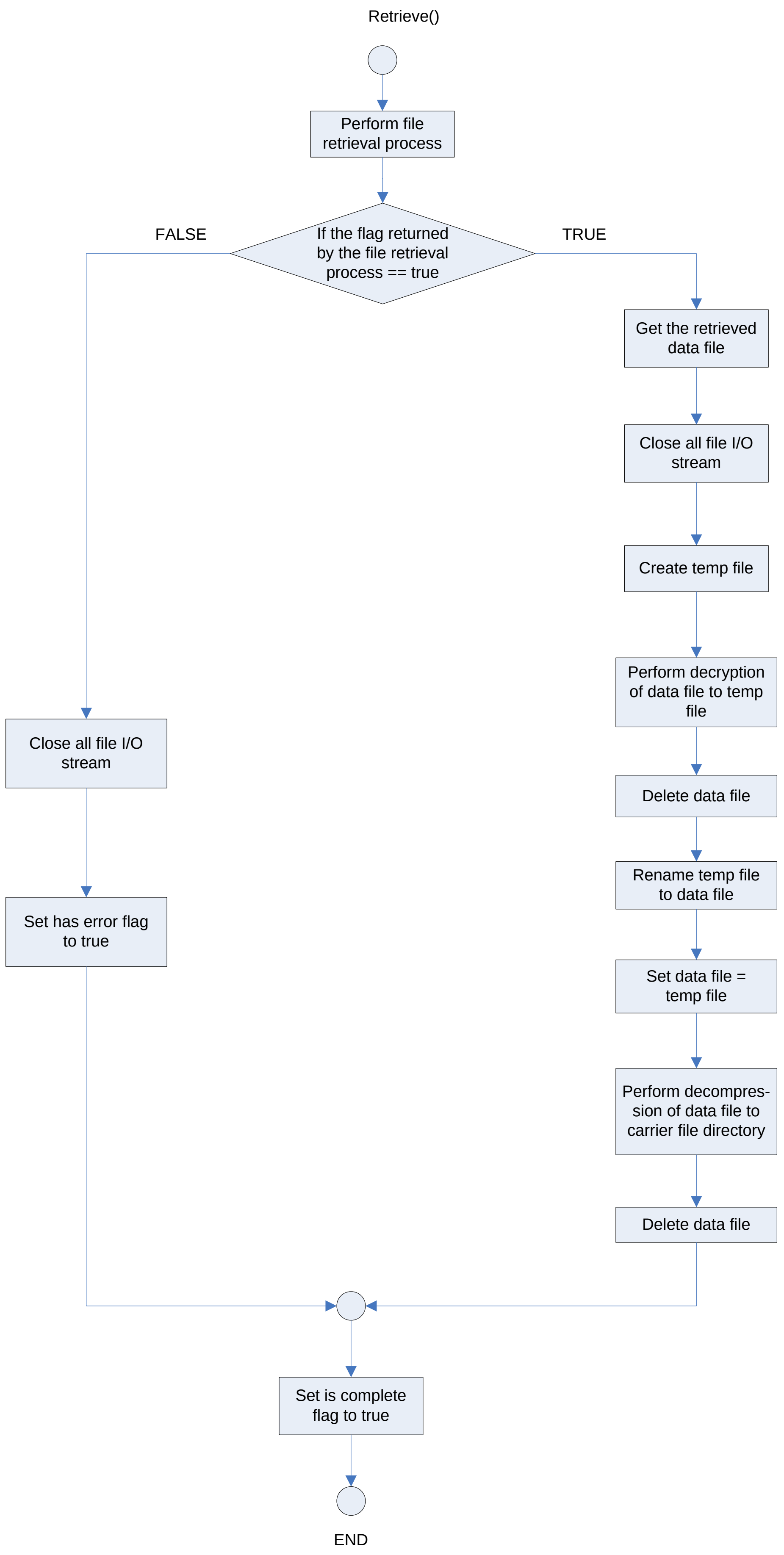
NOTE: One character ‘x’ or ‘-‘ represent a byte in the bitmap carrier file.

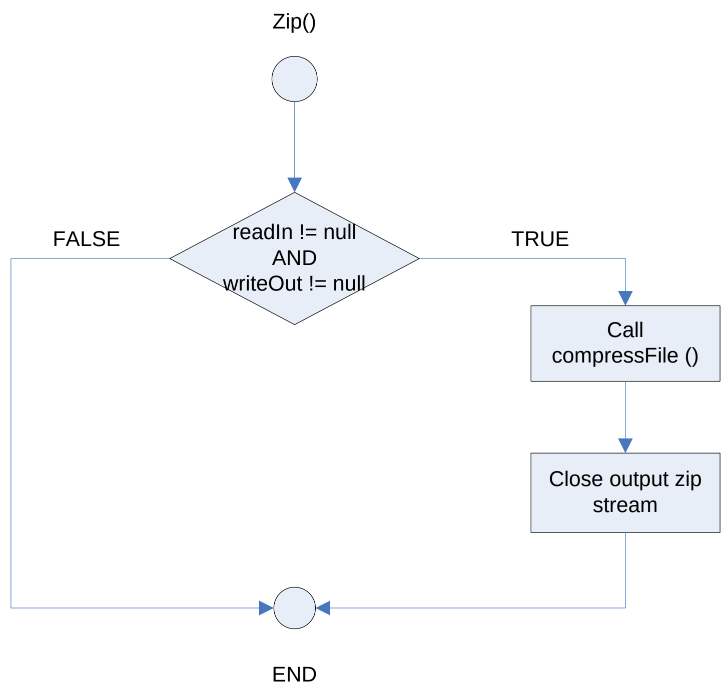
S	→	is the start flag of the injection. It signifies the start location where the data file is injected. The location of this flag is determined solely on the entered password. Hence, similar password produce similar start location of ‘S’ flag
< filename >	→	The filename of the hidden data file. It has at most 20 characters of name + 3 characters of file extension, i.e. “name.ext”
F	→	the end flag to signify the end of filename.
< data >	→	The content of the hidden data file
E E E	→	Three consecutive of EOF flag to signify the end-of-file of the hidden data file. The use of 3 character is to avoid/minimize false flag trigger, especially when injecting bin or executable file.
END	→	End-of-file for the bitmap carrier file

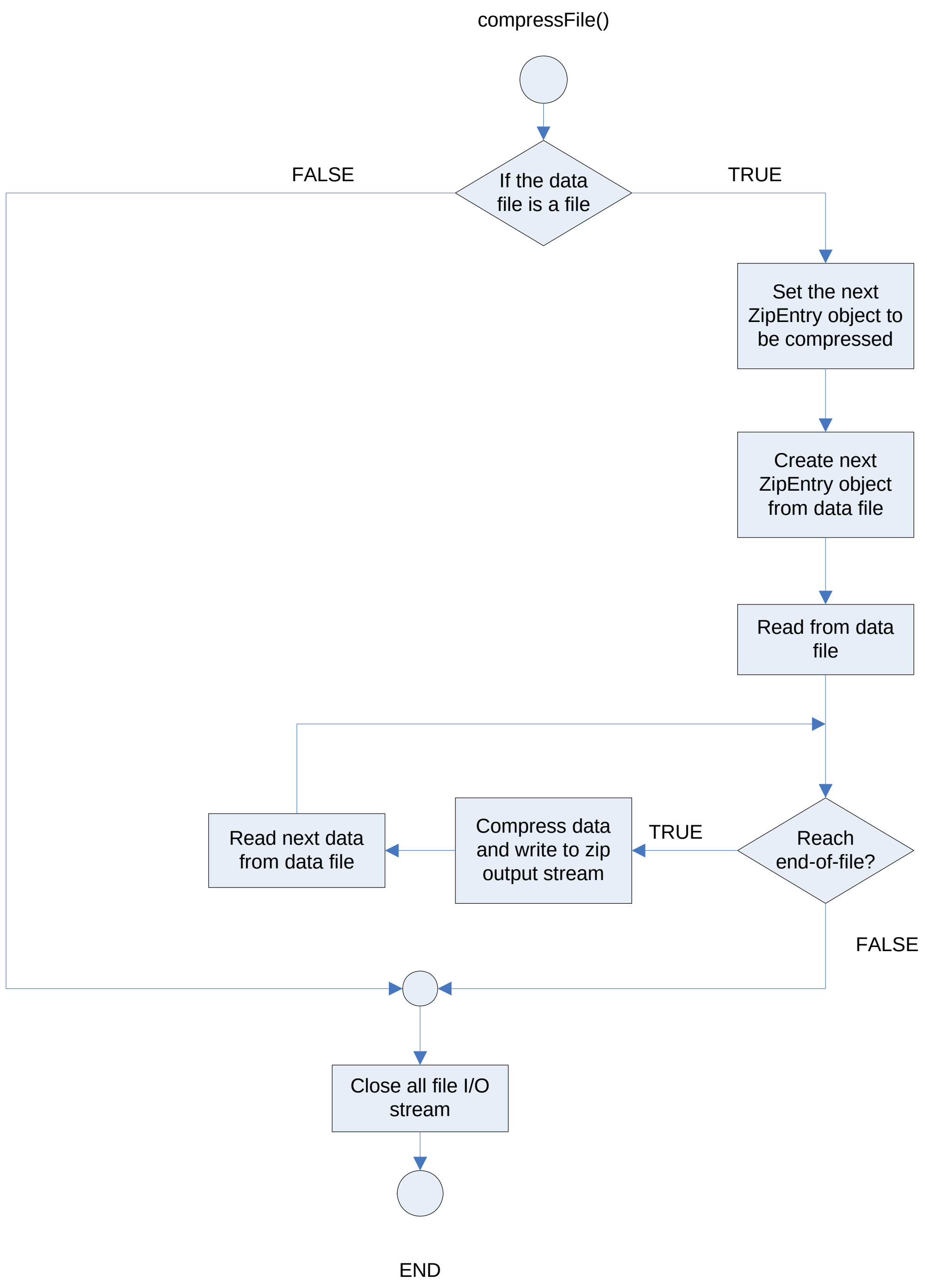
[1] Refer Appendix A to know more about the bitmap file format

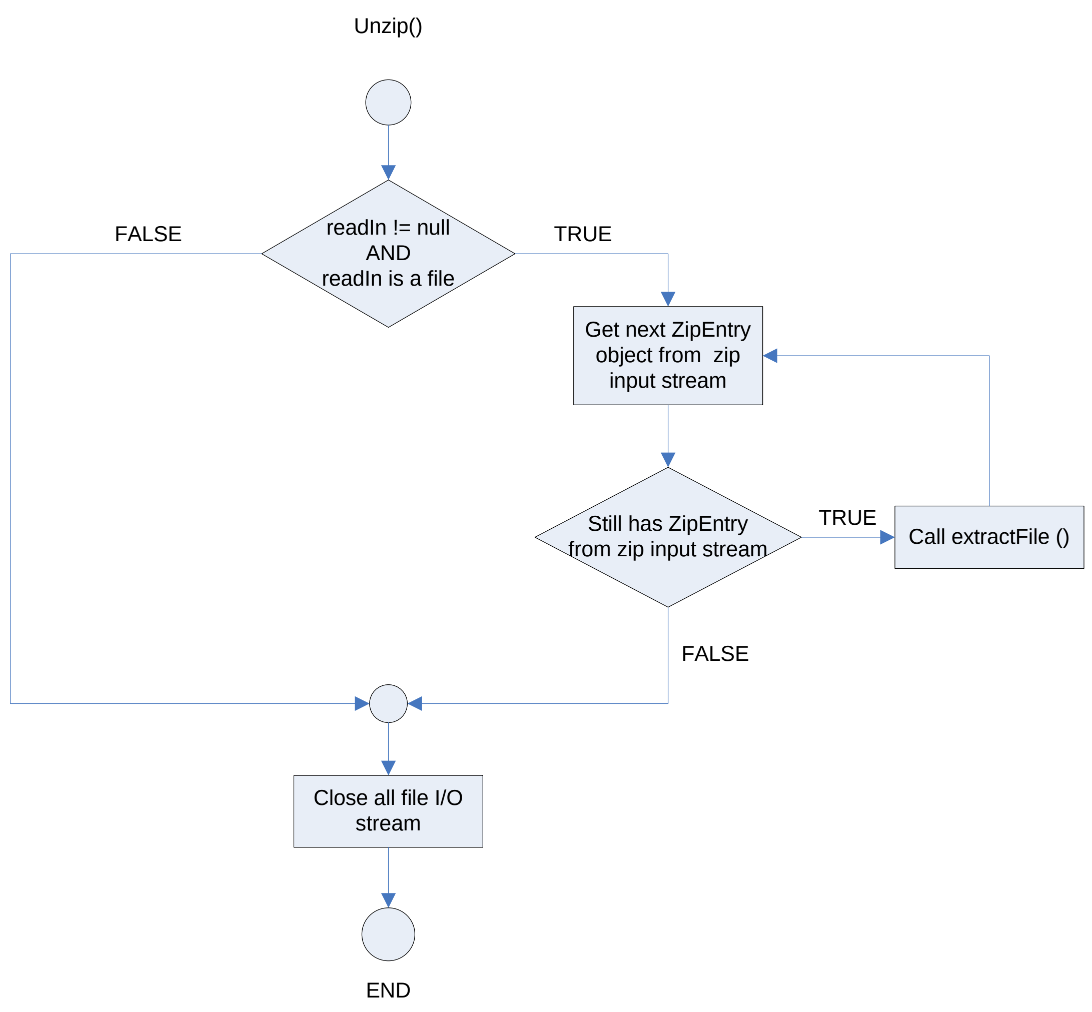
2. Flow charts

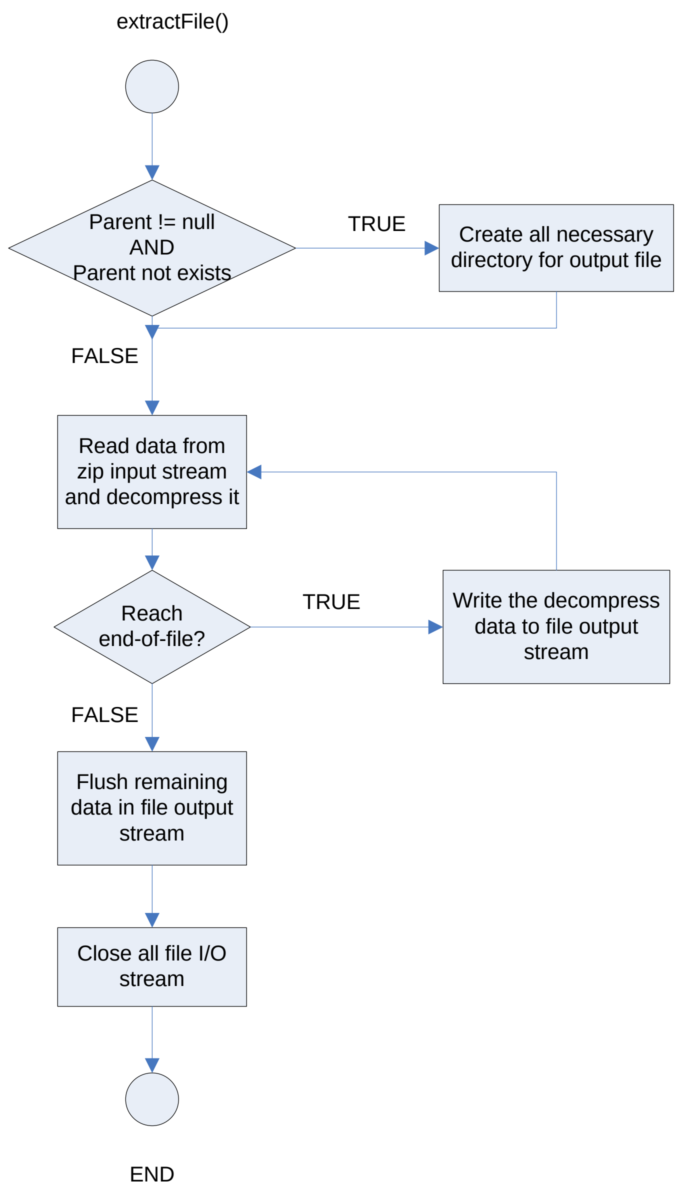

Inject()
Create temp file
Perform compression of data file and name the resultant, compressed file to temp
Rename temp file to data file
Set data file = temp file
If data file can be injected into carrier file
FALSE
TRUE
Create temp file
Perform encryption of data file to temp file
Delete data file
Rename temp file to data file
Set data file = temp file
Set start inject flag to true
Perform injection of data file into carrier file
Set start inject flag to true
Set has error flag to true
Close all file I/O stream
Delete data file
Set is complete flag to true
END

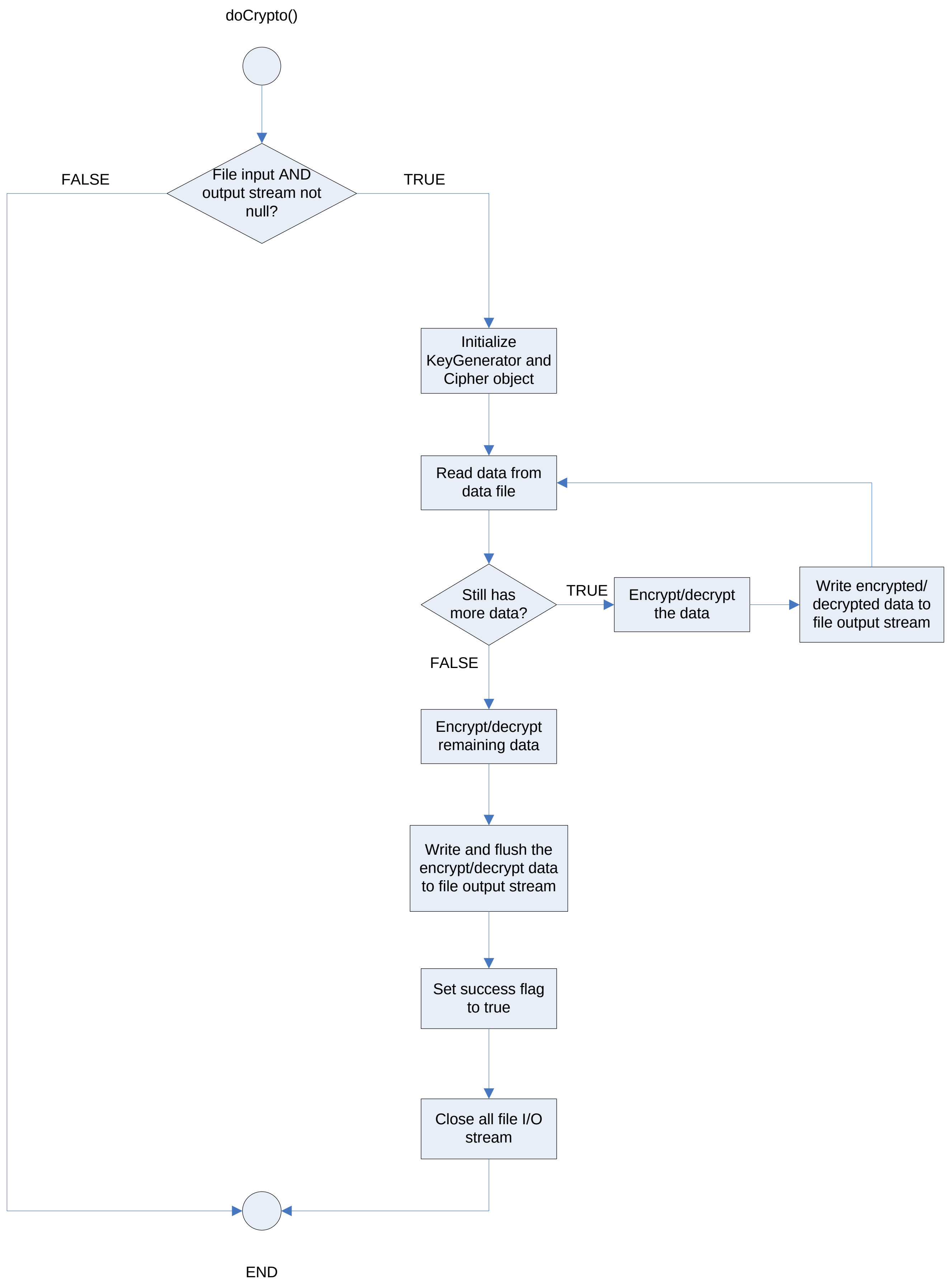

Retrieve()
Perform file retrieval process
If the flag returned by the file retrieval process == true
FALSE
TRUE
Get the retrieved data file
Close all file I/O stream
Create temp file
Perform decryption of data file to temp file
Delete data file
Rename temp file to data file
Set data file = temp file
Perform decompres-sion of data file to carrier file directory
Delete data file
Close all file I/O stream
Set has error flag to true
Set is complete flag to true
END

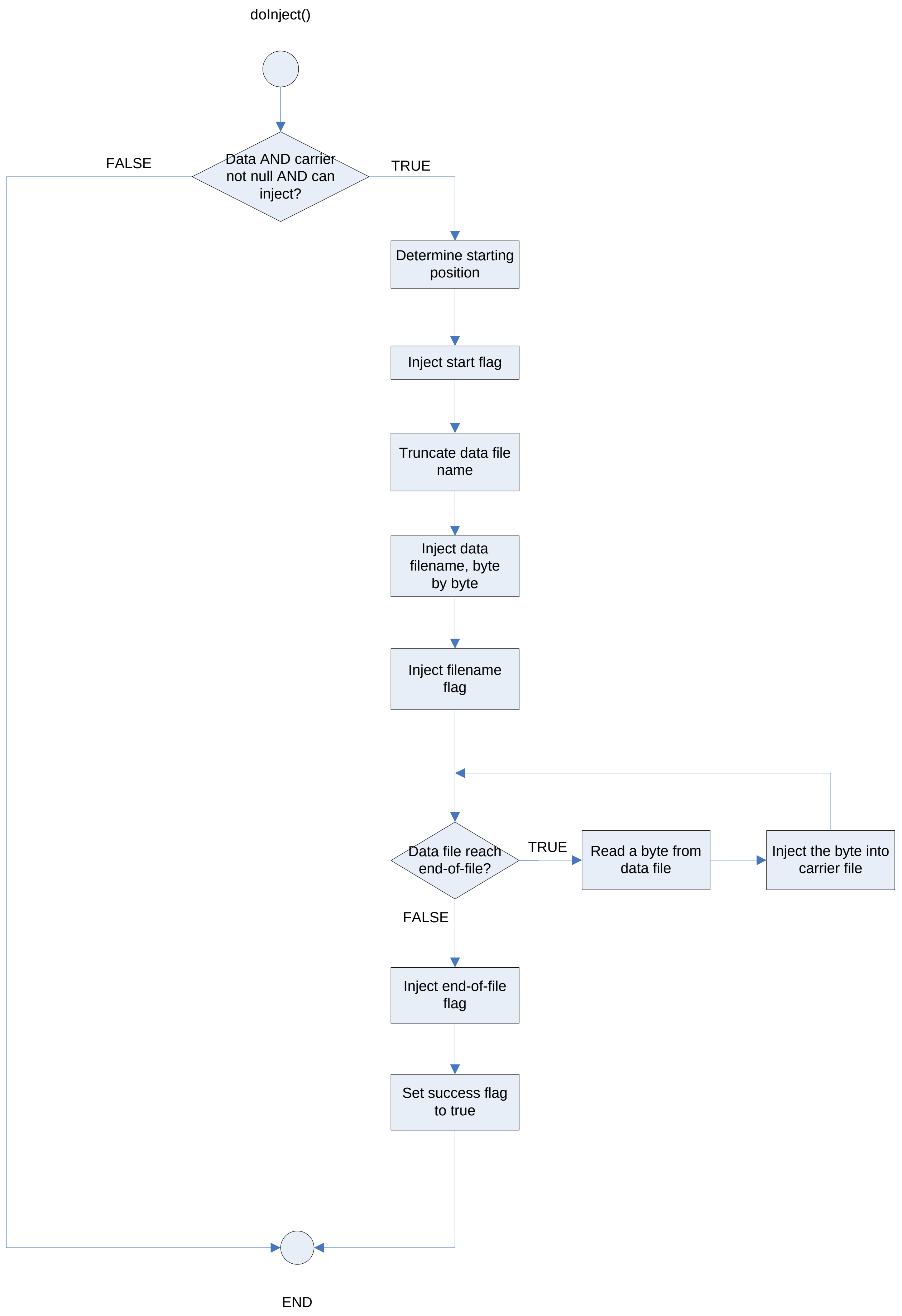

Zip()
readIn != null
AND
writeOut != null
FALSE
TRUE
Call
compressFile ()
Close output zip
stream
END

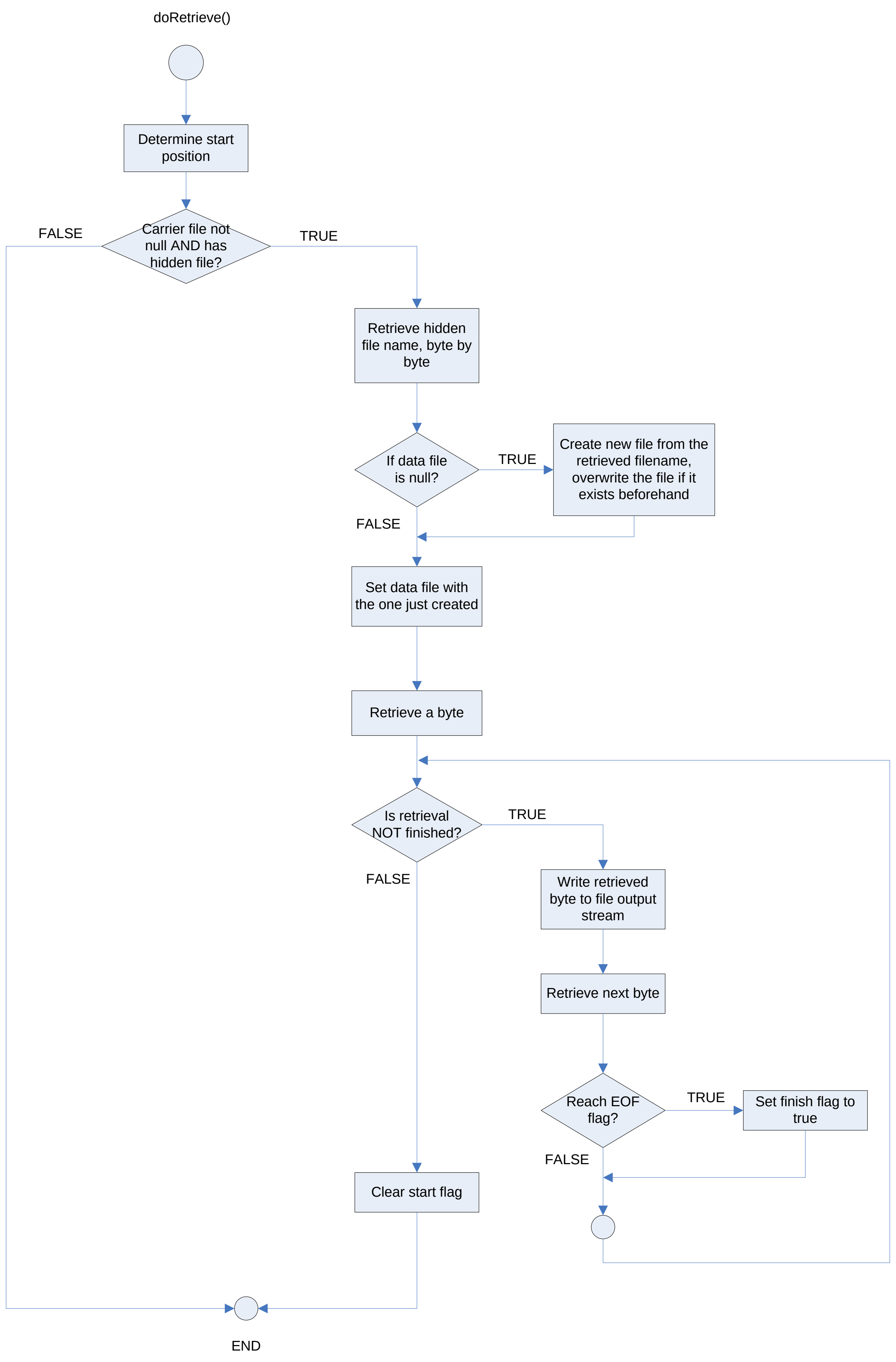

compressFile()
FALSE
If the data file is a file
TRUE
Set the next ZipEntry object to be compressed
Create next ZipEntry object from data file
Read from data file
Read next data from data file
Compress data and write to zip output stream
TRUE
Reach end-of-file?
FALSE
Close all file I/O stream
END


Unzip()
readIn != null
AND
readIn is a file
FALSE
TRUE
Get next ZipEntry object from zip input stream
Still has ZipEntry from zip input stream
TRUE
Call extractFile ()
FALSE
Close all file I/O stream
END

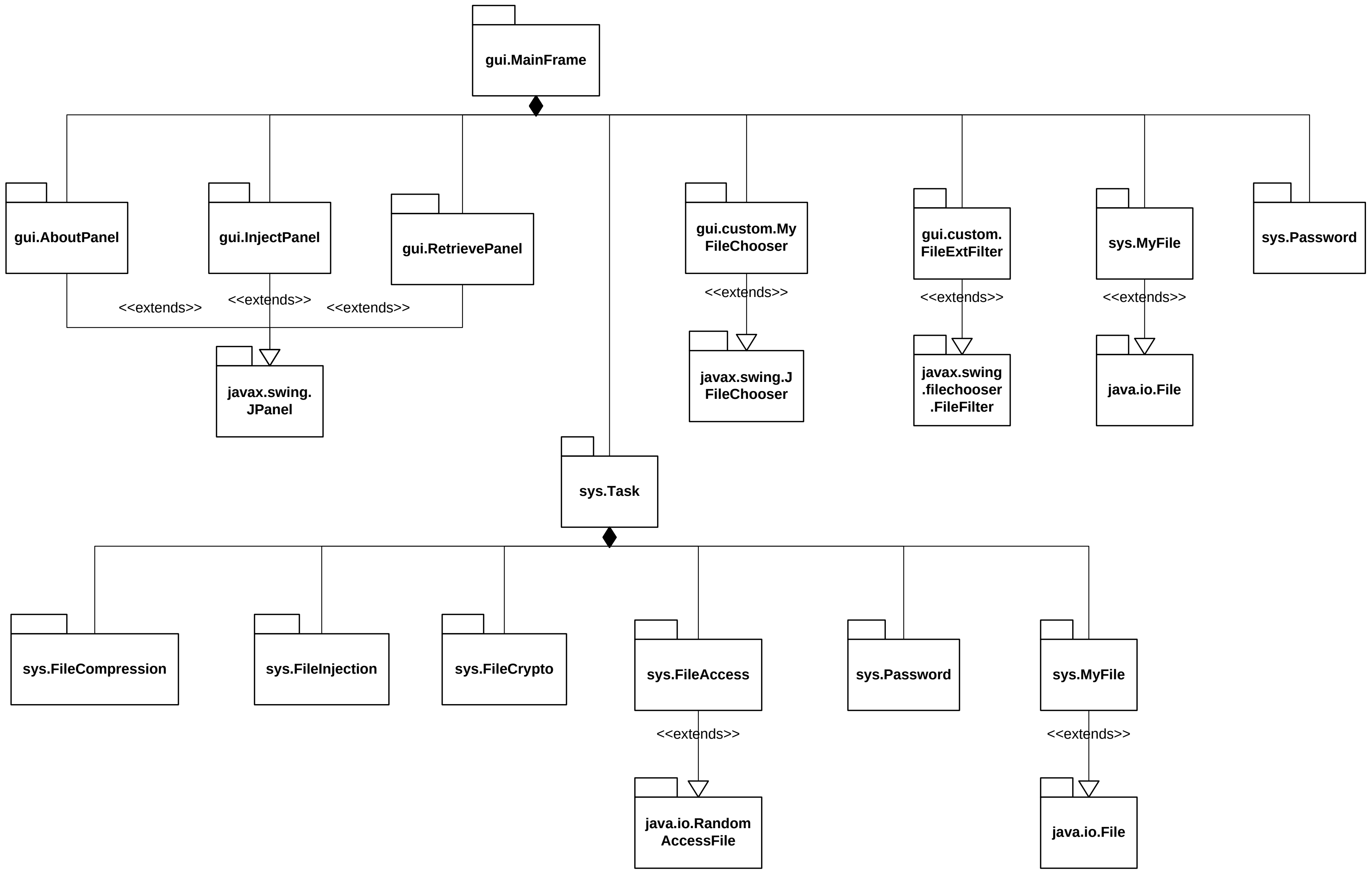

extractFile()
Parent != null AND Parent not exists
TRUE
Create all necessary directory for output file
FALSE
Read data from zip input stream and decompress it
Reach end-of-file?
TRUE
Write the decompress data to file output stream
FALSE
Flush remaining data in file output stream
Close all file I/O stream
END


doCrypto()
File input AND output stream not null?
FALSE
TRUE
Initialize KeyGenerator and Cipher object
Read data from data file
Still has more data?
TRUE
Encrypt/decrypt the data
Write encrypted/ decrypted data to file output stream
FALSE
Encrypt/decrypt remaining data
Write and flush the encrypt/decrypt data to file output stream
Set success flag to true
Close all file I/O stream
END

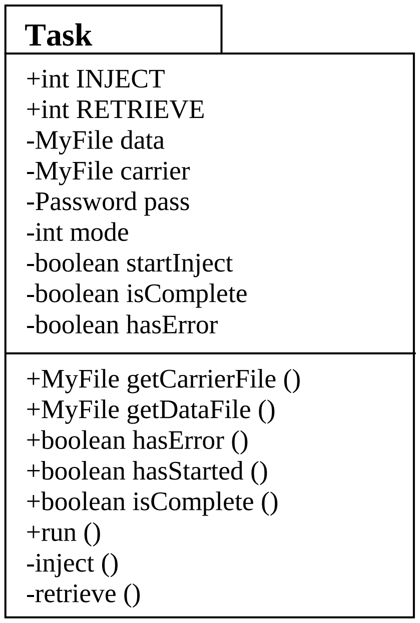

doInject()
Data AND carrier not null AND can inject?
FALSE
TRUE
Determine starting position
Inject start flag
Truncate data file name
Inject data filename, byte by byte
Inject filename flag
Data file reach end-of-file?
TRUE
Read a byte from data file
Inject the byte into carrier file
FALSE
Inject end-of-file flag
Set success flag to true
END


doRetrieve()
Determine start position
Carrier file not null AND has hidden file?
FALSE
TRUE
Retrieve hidden file name, byte by byte
If data file is null?
TRUE
Create new file from the retrieved filename, overwrite the file if it exists beforehand
FALSE
Set data file with the one just created
Retrieve a byte
Is retrieval NOT finished?
TRUE
FALSE
Write retrieved byte to file output stream
Retrieve next byte
Reach EOF flag?
TRUE
Set finish flag to true
FALSE
Clear start flag
END

3. Class hierarchy



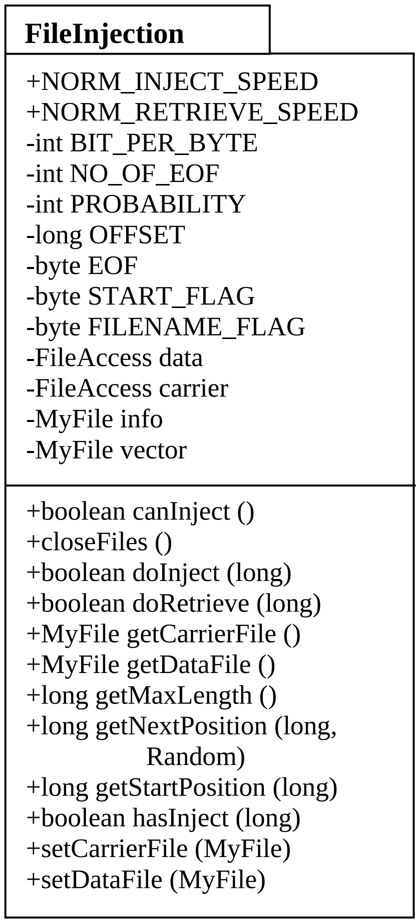
4. Class summary



Task

+int INJECT
+int RETRIEVE
-MyFile data
-MyFile carrier
-Password pass
-int mode
-boolean startInject
-boolean isComplete
-boolean hasError

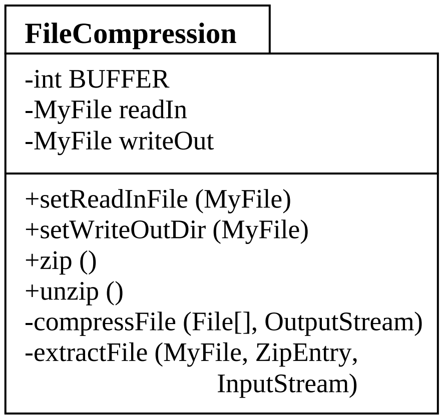
+MyFile getCarrierFile ()
+MyFile getDataFile ()
+boolean hasError ()
+boolean hasStarted ()
+boolean isComplete ()
+run ()
-inject ()
-retrieve ()



FileInjection

+NORM_INJECT_SPEED
+NORM_RETRIEVE_SPEED
-int BIT_PER_BYTE
-int NO_OF_EOF
-int PROBABILITY
-long OFFSET
-byte EOF
-byte START_FLAG
-byte FILENAME_FLAG
-FileAccess data
-FileAccess carrier
-MyFile info
-MyFile vector

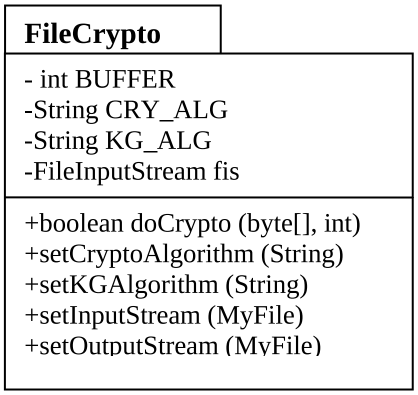
+boolean canInject ()
+closeFiles ()
+boolean doInject (long)
+boolean doRetrieve (long)
+MyFile getCarrierFile ()
+MyFile getDataFile ()
+long getMaxLength ()
+long getNextPosition (long, Random)
+long getStartPosition (long)
+boolean hasInject (long)
+setCarrierFile (MyFile)
+setDataFile (MyFile)



FileCompression

-int BUFFER
-MyFile readIn
-MyFile writeOut

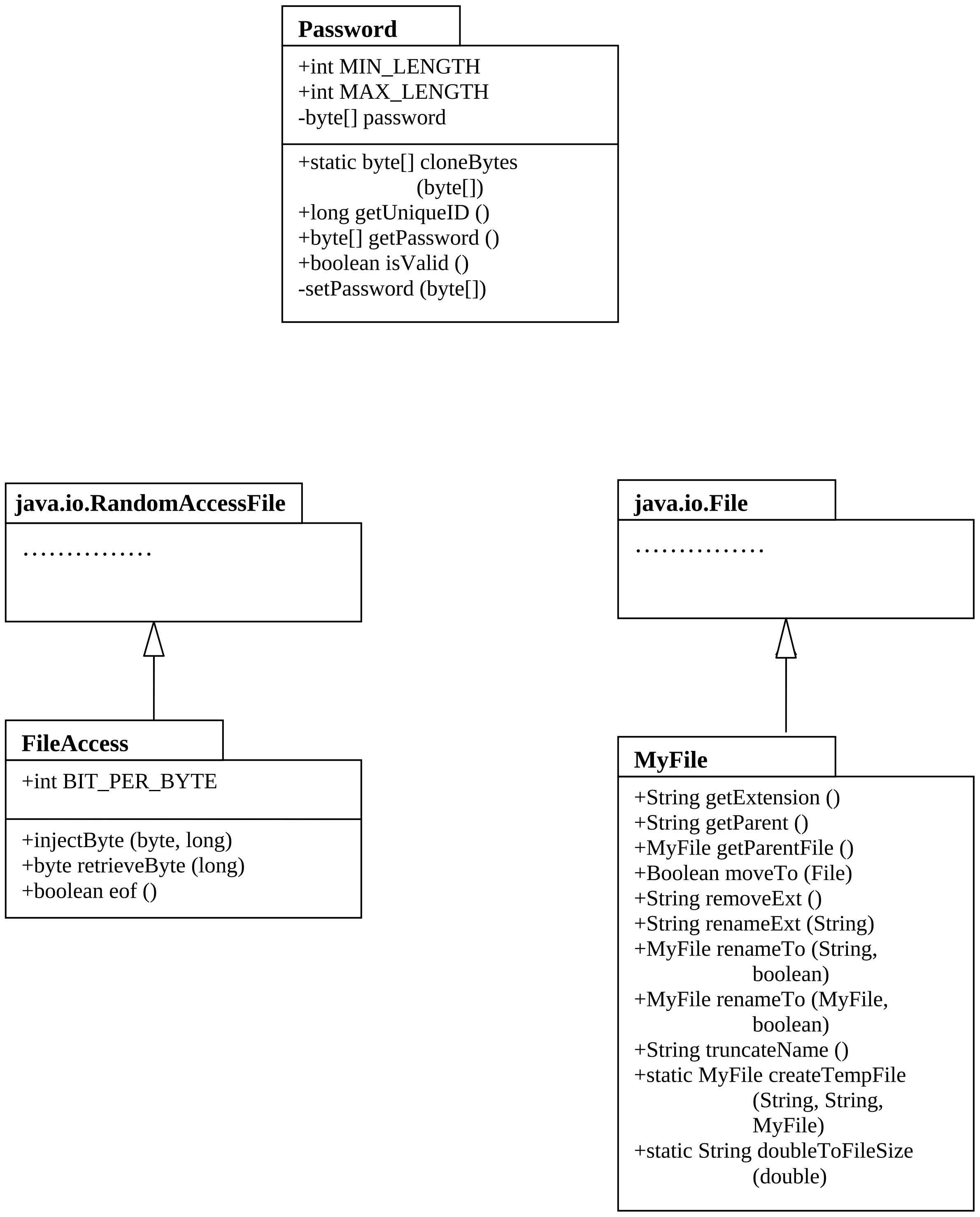
+setReadInFile (MyFile)
+setWriteOutDir (MyFile)
+zip ()
+unzip ()
-compressFile (File[], OutputStream)
-extractFile (MyFile, ZipEntry, InputStream)



FileCrypto

- int BUFFER
-String CRY_ALG
-String KG_ALG
-FileInputStream fis

+boolean doCrypto (byte[], int)
+setCryptoAlgorithm (String)
+setKGAlgorithm (String)
+setInputStream (MyFile)
+setOutputStream (MyFile)


Password
+int MIN_LENGTH
+int MAX_LENGTH
-byte[] password
+static byte[] cloneBytes (byte[])
+long getUniqueID ()
+byte[] getPassword ()
+boolean isValid ()
-setPassword (byte[])
java.io.RandomAccessFile
……………
FileAccess
+int BIT_PER_BYTE
+injectByte (byte, long)
+byte retrieveByte (long)
+boolean eof ()
java.io.File
……………
MyFile
+String getExtension ()
+String getParent ()
+MyFile getParentFile ()
+Boolean moveTo (File)
+String removeExt ()
+String renameExt (String)
+MyFile renameTo (String, boolean)
+MyFile renameTo (MyFile, boolean)
+String truncateName ()
+static MyFile createTempFile (String, String, MyFile)
+static String doubleToFileSize (double)

5. Pseudo code

5.1 Password.java

CLASS: Password.java
INHERITS: none
PURPOSE:
The Password class serves to contain password with its properties and behaviours. It also provides some necessary methods to verify and manipulate the password.

PROPERTIES:
```
      +int MAX_LENGTH
      +int MIN_LENGTH
      -byte[] password
```

BEHAVIOUR:
```
      +Password (byte[])
      +byte[] cloneBytes (byte[])
      +byte[] getPassword (void)
      +long getUniqueID ()
      +boolean isValid ()
      -void setPassword (byte[])
```

ALGORITHM:

```
Constructor #1: Password ()
Purpose: to construct a Password object
Import: byte[] pass
Export: none
START
    Invoke setPassword () with pass
END


Module: cloneBytes ()
Purpose: to make a deep copy of a byte array
Import: byte[] source
Export: byte[] target
START
    FOR 1 to source.length – 1, increment by 1, DO
      Set target[i] to source[i]
    END FOR
END


Module: getPassword()
Purpose: to return the password to the calling method
Import: none
Export: byte[] password
START
    Return password
END
```

```
Module: getUniqueID ()
Purpose: to return the unique ID of this Password object
Import: none
Export: long uniqueID
START
    Convert password to String, and initialize the uniqueID to be its hashcode
    DO
        uniqueID *= password[a random int between 0 and password.length]
        uniqueID += password[a random int between 0 and password.length]
        uniqueID /= password[a random int between 0 and password.length]
        uniqueID -= password[a random int between 0 and password.length]
    WHILE nextBoolean() is true
    END DOWHILE

    Set uniqueID to its absolute value
END


Module: isValid ()
Purpose: to test if this Password object is valid or not
Import: none
Export: Boolean isValid
Comment: a valid password is of any character with length between 2 and 20 (inclusive)
START
    Return (MIN_LENGTH <= password.length <= MAX_LENGTH)
END


Module: setPassword()
Purpose: to set a password to this Password object
Import: byte[] pass
Export: none
START
    IF pass != null THEN
        Invoke cloneBytes() with pass, and assign the returned value to password
    END IF
END
```

5.2 MyFile.java

CLASS: MyFile.java
INHERITS: java.io.File
PURPOSE:
The MyFile class serve to provide some extra behaviours/methods that are required for FIRA2 but are not present in the Java built-in package, java.io.File.

PROPERTIES:
```
      (none)
```

BEHAVIOUR:
```
      +MyFile (String)
      +MyFile (File, String)
      +static String doubleToFileSize (double)
      +static createTempFile (String, String, MyFile)
      +getExtension ()
      +String getParent ()
      +MyFile getParentFile ()
      +boolean move (File)
      +String removeExt ()
      +String renameExt (String)
      +MyFile renameTo (String, boolean)
      +MyFile renameTo (MyFile, boolean)
      +String truncateName ()
```

ALGORITHM:

```
Constructor #1: MyFile ()
Purpose: to construct a MyFile object
Import: String filename
Export: none
START
    Invoke super() constructor with filename
END


Constructor #2: MyFile ()
Purpose: to construct a MyFile object
Import: File parent file, String child filename
Export: none
START
    Invoke super() constructor with parent and filename
END


Module: createTempFile()
Purpose: to over-ride the super-class method to return MyFile object
         instead of File object
Import: String file name prefix, String file name suffix, MyFile the directory of
        the temporary file to be created
Export: MyFile object for the temporary file created
START
    Invoke createTempFile() from super-class, File with all required parameters
    Create and return the MyFile object describing the just created temp file
END
```

```
Module: doubleToFileSize ()
Purpose: to format a double value to a file size representation in String
Import: double num
Export: String a string representation of num to show the file size
START
    IF (num >= 1000) THEN
        num /= 1000
        set multiplier to “kb”
    END IF

    IF (num >= 1000) THEN
        num /= 1000
        set multiplier to “Mb”
    END IF

    IF (num >= 1000) THEN
        num /= 1000
        set multiplier to “Gb”
    END IF

    Format the num to 2-decimal point and concatenate it with the multiplier
END


Module: getExtension()
Purpose: to return the file extension of this file object
Import: none
Export: a String containing the file extension
START
    Find the last occurrence of the period, ‘.’ in this file name
    IF the index of the last occurrence is greater than 0 THEN
        Return all String after the period, ‘.’
    ELSE
        Return NULL
    END IF
END


Module: getParent ()
Purpose: to return the absolute path of the parent file
Import: none
Export: String parent absolute path
START
    Get the absolute path of this File object
    Find the last index of the absolute pathname separator, ‘/’
    Remove the child filename of this file (left the parent absolute path)
END


Module: getParentFile ()
Purpose: to return the parent File of this File object
Import: none
Export: MyFile parent
START
    Invoke the getParent() method
    Construct a new MyFile object from the returned value of getParent ()
    Return the newly constructed MyFile object
END
```

```
Module: moveTo ()
Purpose: to move this file to specified location, and rename to specified
         specified name
Import: File object, describing the destination location and filename
Export: boolean to determine if the file move is a success or not.
START
    Invoke renameTo() from the superclass with destination File object, and return
END


Module: renameTo()
Purpose: to rename this file and return the new MyFile object, describing the
         newly renamed file
Import: MyFile object, describing the new file name, boolean flag to force rename
        this file or not.
Export: MyFile object for the newly renamed file
Note: When force rename flag is true, the rename process will always be success,
      since if the file already exists prior to file rename, the file will be
      OVERWRITTEN
START
    Invove the rename() with the imported File object name and the force rename
      flag, then return
END


Module: renameTo()
Purpose: to rename this file and return the new MyFile object, describing the
         newly renamed file
Import: the new file name in String, boolean flag to force rename
        this file or not.
Export: MyFile object for the newly renamed file
Note: When force rename flag is true, the rename process will always be success,
      since if the file already exists prior to file rename, the file will be
      OVERWRITTEN
START
    Create a new File object with this file's parent and the new file name

    IF the file already exists AND force rename THEN
        Delete the file
    END IF

    Invoke the renameTo() in super-class, File with the new File object
    IF rename is success THEN
        Create and return the new MyFile object describing the renamed file
    ELSE
        Return this file object
    END IF
END


Module: renameExt ()
Purpose: to return the filename of this File object with its extension renamed
Import: String new extension
Export: filename with extension renamed
START
    Invoke removeExt (), and assign the returned value to filename
    Concatenate filename with "." and new file extension
END
```

```
Module: removeExt ()
Purpose: to return the filename of this File object with its extension removed
Import: none
Export: String filename without file extension
START
    Determine the index for the last occurrence of '.'
    IF the index is > 0 THEN
        Get and return all the String before the period
    ELSE
        Simply return the file name
    END
END


Module: truncateName ()
Purpose: to set a password to this Password object
Import: none
Export: String shortName
START
    Separate the filename into filename and file extension
        Get the full name of this file
        Get the index of last occurrence of '.' in this filename
        IF no extension THEN
            The name of the file is equals to the full name
        ELSE
            The name of the file is all String before the period, '.'
            The extension is all String after the period, '.'
        END IF

    IF (filename > MAX_LENGTH) THEN
        Truncate the filename to 20 characters
    END IF

    Return (filename + file extension)
END
```

5.3 FileAccess.java

CLASS: FileAccess.java
INHERITS: java.io.RandomAccessFile
BRIEF DESCRIPTION:
To provide methods for injecting and retrieving a byte from a stream of data in bytes. It is inherited from java.io.RandomAccessFile because it requires some of the properties and behaviour from that super-class.

PROPERTIES:

```
      +int BIT_PER_BYTE
```

BEHAVIOUR:

```
      +FileAccess (File, String)
      +FileAccess (String, String)
      +void injectByte (byte, long)
      +byte retrieveByte (long)
      +boolean eof ()
```

ALGORITHM:

```
Constructor #1: FileAccess ()
Purpose: to construct a FileAccess object
Import: File file, String mode
Export: none
START
    Invoke super() constructor with file and mode
END


Constructor #2: FileAccess ()
Purpose: to construct a FileAccess object
Import: String filename, String mode
Export: none
START
    Invoke super() constructor with filename and mode
END


Module: injectByte ()
Purpose: to inject a byte into a file at the location pointed by pos
Import: byte value, long pos
Export: none
START
    Create the mask for the associate BIT_PER_BYTE.
        e.g. if BIT_PER_BYTE is 2, then mask = %0000 0011
             if BIT_PER_BYTE is 4, then mask = %0000 1111

    FOR 0 to (8/BIT_PER_BYTE)-1, DO
        Set the file pointer to pos
        Read a byte as pointed by the file pointer and assign it to temp
        Mask the BIT_PER_BYTE number of least significant bit of temp to 0,
          the rest remain untouched

        Set temp = temp OR (value AND mask)
        Logical right shift value by 1
        Set the file pointer to pos
        Write temp back to the file
        Increment pos by one
    END FOR
END


Module: retrieveByte ()
Purpose: to retrieve a byte from a file at the location pointed by pos
Import: long pos
Export: byte retrieved byte
START
    Create the mask for the associate BIT_PER_BYTE.
        e.g. if BIT_PER_BYTE is 2, then mask = %0000 0011
             if BIT_PER_BYTE is 4, then mask = %0000 1111

    Initialize retrieved to 0

    Set the file pointer to pos
    FOR i=0 to (8/BIT_PER_BYTE)-1, DO
        Read a byte as pointed by the file pointer and assign it to temp
        Retrieved = retrieved OR ((temp AND mask) << i*BIT_PER_BYTE)
    END FOR
END


Module: eof ()
Purpose: to determine if end-of-file has been reached or not
Import: none
Export: true if end-of-file is reached, false otherwise
START
    Get the current file pointer
    Return (current file pointer >= file length)
END
```

5.4 FileCompression.java

CLASS: FileCompression.java
INHERITS: none
PURPOSE:
To have a class that provides high level access to file compression/ decompression (also known as file zipping/unzipping), so as to encapsulate all the details for the file compression/decompression within this class

NOTE: This FileCompression.java is modified slightly from a stand-alone file zipping/unzipping application written by the author when learning the Java Api in file zipping/unzipping

PROPERTIES:
```
      -int BUFFER
      -MyFile readIn
      -MyFile readout
```

BEHAVIOURS:
```
      +void setReadInFile (MyFile)
      +void setWriteOutFile (MyFile)
      +void unzip ()
      +void zip ()
      -void compressFiles (File[], ZipOutputStream)
      -void extractFiles (MyFile, ZipEntry, ZipInputStream)
```

ALGORITHM:

```
Constructor #1: FileCompression ()
Purpose: to construct the FileCompression object
Import: MyFile object specifying the source file, MyFile object specifying the
        destination directory
Export: none
START
    Invoke setReadInFile () with source file and setWriteOutDir () with
      destination directory
END


Constructor #2: FileCompression ()
Purpose: to construct the FileCompression object
Import: String specifying the source file, String specifying the destination
        directory
Export: none
START
    Invoke constructor #1 with all necessary parameters
END


Module: setReadInFile ()
Purpose: to set the source file for where the zip stream will read from
Import: MyFile object specifying the source file
Export: none
Note: Will only set the file iff it exists.
START
    IF imported MyFile object is not null AND it exists THEN
        Set readIn to be file
    END IF
END
```

```
Module: setWriteOutDir ()
Purpose: to set the output/destination directory to which the zipped files will be
         extracted/decompressed
Import: MyFile object specifying output/destination directory
Export: none
START
    IF imported MyFile object is not null THEN
        Create a new MyFile object with the absolute path described by the
          imported MyFile object.
        Set writeOut to be the newly created MyFile object
    END IF
END


Module: unzip ()
Purpose: to unzip the compressed file(s)
Import: none
Export: none
START
    IF source file (readIn) not null AND readIn is a file THEN
        Create all required zip I/O stream
        Create the destination directory if it does not exists

        WHILE there still has ZipEntry object DO
            Determine the output destination of extracted file
            Invoke extractFiles () with the determined destination file, current
              zip entry object, and zip input stream object
        END WHILE
    END IF
END


Module: extractFiles ()
Purpose: to extract a zipped file, specified by the imported ZipEntry object to
         its specified destination
Import: MyFile object specifying the destination of the extracted file, ZipEntry
        object specifying the file to be extracted, ZipInputStream object
        specifying the input stream for the file extraction
Export: none
START
    IF the parent directory into which the file should be extracted does not
    exists THEN
        Create the directory
    END IF

    WHILE there is still zipped data DO
        Read zipped data stream to buffer and decompress it (done automatically by
          the read() method)
        Write data in buffer to destination file (which is the extracted file)
    END WHILE

    Flush and close file output stream
END
```

```
Module: zip()
Purpose: to zip/compress a file or all files in a directory
Import: none
Export: none
START
    IF source file/directory not null AND destination directory not null THEN
        Invoke compressFiles() with the list of files and ZipOutputStream object

        Close the ZipOutputStream object
    END IF
END


Module: compressFiles ()
Purpose: to compress the given list of files into specified ZipOutputStream object
Import: File object specifying the source file to be compressed, ZipOutputStream
        object for file compression
Export: none
START
    IF source file is a file THEN
        Create the new ZipEntry object for this file
        Put it as next zip entry and compress it (done automatically)

        WHILE still has data to read from files[i] DO
           Read data from files[i] into the buffer
           Write data in buffer to ZipOutputStream and compress the file
             (The file compression is done automatically)
        END WHILE

        Close file input stream from source file
    END IF
END
```

5.5 FileCrypto.java

CLASS: FileCrypto.java
INHERITS: none
PURPOSE:
To have a class that provides high level access to file encryption/ decryption, so as to encapsulate all the details for the file encryption/ decryption within this class

PROPERTIES:

```
      -int BUFFER
      -String CRY_ALG
      -String KG_ALG
      -FileInputStream fis
      -FileOutputStream fos
      -Cipher c
      -KeyGenerator kg
```

BEHAVIOUR:

```
      +void setCryptoAlgorithm (String)
      +void setKGAlgorithm (String)
      +void setInputStream (MyFile)
      +void setOutputStream (MyFile)
      +boolean doCrypto (byte[], int)
```

ALGORITHM:

```
Constructor #1: FileCrypto ()
Purpose: to construct the FileCrypto object
Import: Myfile source file, MyFile destination file
Export: none
START
    Invoke setInputStream(), setOutputStream(), setCryptoAlgorithm() and
      setKGAlgorithm() with their corresponding parameter
END


Module: setInputStream ()
Purpose: to set the file input stream for this file cryptography
Import: MyFile specifying the source file
Export: none
START
    Create a FileInputStream from imported source file and assign it to fis
END


Module: setOutputStream ()
Purpose: to set the file output stream for this file cryptography
Import: MyFile specifying the destination file
Export: none
START
    Create a FileOutputStream from imported destination file and assign it to fos
END
```

```
Module: setCryptoAlgorithm ()
Purpose: to set the cryptography algorithm for this file cryptography
Import: String representing the cryptography algorithm
Export: none
START
    Create a Cipher instance from Cipher.getInstance() with the imported
      cryptography algorithm
END


Module: setKGAlgorithm()
Purpose: to set the algorithm for the key generator
Import: String specifying the KeyGenerator algorithm
Export: none
START
    Create a KeyGenerator instance from KeyGenerator.getInstance() with the
      imported KeyGenerator algorithm
END


Module: doCrypto ()
Purpose: to actually perform the encryption/decryption task, depending on its mode
         of operation
Import: byte[] specifying the password, int specifying the mode of operation
Export: true if encryption/decryption process succeed, false otherwise
START
    IF file input AND output stream are not null THEN
        Initialize the KeyGenerator with the SecureRandom object using the
          imported password
        Initialize th Cipher object with imported mode and the SecretKey object

        WHILE there is still data to be read by file input stream DO
            Read data from file input stream
            Store the data in buffer
            Encrypt/decrypt the data in buffer
            Write the encrypted/decrypted data from buffer to file output stream
        END WHILE

        // finishing work
        Encrypt/decrypt and finalize the final data in the buffer
        Write and flush the data from buffer to file output stream

        Set the success flag to true

        Close all file I/O stream after finish
    END IF
END
```

5.6 FileInjection.java

CLASS: FileInjection.java
INHERITS: none
PURPOSE:
To provide a class that will perform the file injection and retrieval task. It includes all the required checking before file injection and retrieval.

PROPERTIES:

```
      +float NORM_INJECT_SPEED
      +float NORM_RETRIEVE_SPEED
      -int NO_OF_EOF
      -int PROBABILITY
      -long OFFSET
      -byte START_FLAG
      -byte FILENAME_FLAG
      -byte EOF
      -MyFile info
      -MyFile vector
      -FileAccess data
      -FileAccess carrier
```

BEHAVIOUR:

```
      +boolean canInject ()
      +boolean hasInject (long)
      +boolean doInject (long)
      +boolean doRetrieve (long)
      +long getMaxLength ()
      +long getStartPosition (long)
      +long getNextPosition (long, Random)
      +MyFile getDataFile ()
      +MyFile getCarrierFile ()
      +void setDataFile (MyFile)
      +void setCarrierFile (MyFile)
      +void closeFiles ()
```

ALGORITHM:

```
Module: canInject ()
Purpose: to check if the given data file can be injected into the given carrier
         file
Import: none
Export: none
START
    Return (data file size < the value returned by getMaxLength())
END
```

```
Module: hasInject ()
Purpose: to check if there is any injected file at the position given by the
         imported ID
Import: long specifying the password ID
Export: boolean flag to specify if there is injected file or not
START
    Determine the staring position in the carrier file using the imported ID
    Retrieve a byte at the starting position
    IF the retrieved byte EQUALS the START_FLAG THEN
        There is injected file
    ELSE
        There is no injected file
    END IF
END


Module: doInject ()
Purpose: to actually perform the file injection task iff it can be performed. This
         means that necessary checking will be done before the file injection
         process starts
Import: long specifying the password ID
Export: true if file injection process succeed, false otherwise
START
    IF data file AND carrier file not null AND injection process can be performed
      (determined by the returned value of canInject()) THEN

        Determine the starting position within the carrier file
        Inject the START_FLAG into carrier file
        Determine next injectible position

        Truncate the data filename
        FOR 0 to (truncated filename length – 1) DO
            Inject the truncated filename into carrier file, one byte by one byte
            Determine next injectible position
        END FOR

        Inject FILENAME_FLAG into carrier file
        Determine the next injectible position

        WHILE data file have not reach end-of-file DO
            Read a byte from data file
            Inject the byte into carrier file
            Determine the next injectible position
        END WHILE

        FOR 0 to (NO_OF_EOF - 1) DO
            Inject EOF into carrier file
            Determine next injectible position
        END FOR

        Set success flag to true
    END IF
END


Module: doRetrieve ()
Purpose: to perform the actual file retrieval process iff there exist a hidden
         file in the specified carrier file. There will be necessary checking
         before the file retrieval process is performed
Import: long specifying the password ID
Export: true if retrieval process succeed, false otherwise
START
    Determine the start position from the imported password ID

    IF carrier file is not null AND there is injected file in carrier file THEN
        // retrieve injected filename
        Determine the next position (to skip the START_FLAG)
        Retrieve the first byte
        WHILE FILENAME_FLAG not encountered yet DO
            Concatenate retrieved byte to the filename
            Determine next position
            Retrieve next byte for the filename
        END WHILE

        IF data file is null THEN
            Create the data file from the retrieved filename, overwrite if the
              file already exists beforehand
            Set the data file to this newly created file
        END IF

        Determine next position
        Retrieve a byte
        WHILE have not finish retrieving the hidden file DO
            Write the retrieve byte to output/destination file
            Determine next position
            Retrieve a byte
            Determine if hidden file has retrieved completely
        END WHILE

        Reset/Delete the START_FLAG
    END IF
END


Module: getMaxLength ()
Purpose: to calculate the max file size for a data file to be safely injected to
         the given carrier file
Import: none
Export: long specifying the max length
START
    Return (carrier file size - OFFSET)/(8/BIT_PER_BYTE + 2)
END


Module: getStartPosition ()
Purpose: to determine the staring position for file injection/retrieval based on
         imported password ID
Import: long specifying password ID
Export: long specifying the starting posiiton
START
    Start = (Imported ID % (value returned by getMaxLength())*8/BIT_PER_BYTE)
              + OFFSET
END


Module: getNextPosition ()
Purpose: to determine the next position for file injection and retrieval
Import: long specifying current position, Random object to generate a random int
Export: long specifying next position
START
    Next position = current position + 8/BIT_PER_BYTE
                    + (a random int between 0-10)/PROBABILITY

    IF next position >= (the carrier file size - 8/BIT_PER_BYTE) THEN
        Set the next position to OFFSET
    END IF
END
```

```
Module: getDataFile ()
Purpose: to return the data file object
Import: none
Export: MyFile specifying the data file for this FileInjection object
START
    Return info
END


Module: getCarrierFile ()
Purpose: to return the carrier file object
Import: none
Export: MyFile specifying the carrier file for this FileInjection object
START
    Return vector
END


Module: setDataFile ()
Purpose: to set the source/data file
Import: MyFile specifying the source/data file
Export: none
START
    IF imported source/data file not null THEN
        data = new FileAccess object created from imported data file
        info = imported data file
    END IF
END


Module: setCarrierFile ()
Purpose: to set the carrier file
Import: MyFile specifying the carrier file
Export: none
START
    IF imported carrier file not null THEN
        carrier = new FileAccess object created from imported carrier file
        vector = imported carrier file
    END IF
END


Module: closeFiles()
Purpose: to close all the file I/O streams
Import: none
Export: none
START
    IF data file is not null THEN
        Close the data file stream

    IF carrier file is not null THEN
        Close the carrier file stream
END
```

5.7 Task.java

```
CLASS: Task.java
INHERITS: java.util.Thread
PURPOSE:
To combine all the 2x3 process for FIRA2, i.e.
   1. For file injection
      Process: file compression, file encryption and file injection;
   2. For file retrieval
      Process: file retrieval, file decryption and file decompression

PROPERTIES:
      +int INJECT
      +int RETRIEVE
      -MyFile data
      -MyFile carrier
      -Password pass
      -int mode
      -boolean startInject
      -boolean isComplete
      -boolean hasError

BEHAVIOUR:
      +void run ()
      -void inject ()
      -void retrieve ()
      +MyFile getDataFile ()
      +MyFile getCarrierFile ()
      +boolean hasStarted ()
      +boolean isComplete ()
      +boolean hasError ()


ALGORITHM:

Module: run ()
Purpose: to perform a task in separate thread
Import: none
Export: none
START
    IF operation mode is INJECT THEN
        Invoke inject()
    ELSE IF operation mode is RETRIEVE THEN
        Invoke retrieve()
    END IF
END


Module: inject ()
Purpose: to combine all the required process for file injection
Import: none
Export: none
START
    Create a temp file
    Perform file compression of data (original) file to the temp file just created
    Rename the temp file to the data file (only change the file name, the location
      of the temp file remains)

    Determine if the compressed file can be injected into the given carrier file
    IF can THEN
        Create another temp file
        Perform file encryption of data (not original, but is the compressed file
          from previous file compression process) file to temp file
        Delete the data file
        Rename the temp file to data file

        Set startInject flag to true

        Perform file injection of data file into given carrier file
    ELSE
        Set startInject flag to true
        Set hasError flag to true
    END IF

    Set the isComplete flag to true
END


Module: retrieve ()
Purpose: to combine all the required process for file retrieval
Import: none
Export: none
START
    Retrieve the hidden data file from given carrier file

    IF the hidden file is successfully retrieved THEN
        Get the data file from FileInjection object
        Close all file I/O stream for the FileInjection object

        Create a temp file
        Perform decryption of data (retrieved data) file to temp file
        Delete the data file
        Rename temp file to data file

        Perform the file decompression to the same directory as the carrier file
        Delete the data file (this data file IS NOT the decompressed file, but is
          the compressed file after the file decryption process)
    ELSE
        Close all file I/O stream for the FileInjection object
        Set hasError flag to true
    END IF

    Set isComplete flag to true
END


Module: getDataFile ()
Purpose: to return the data file of this Task object
Import: none
Export: MyFile data file
START
    Return data file
END
```

```
Module: getCarrierFile ()
Purpose: to return the carrier file of this Task object
Import: none
Export: MyFile carrier file
START
    Return carrier file
END


Module: hasStarted ()
Purpose: to determine if the file injection process has started (exculsive of file
         compression and encryption process). It is used to update the progress
         bar in the GUI
Import: none
Export: boolean flag specifying if the file injection process has started or not
START
    Return startInject
END


Module: isComplete ()
Purpose: to determine if the overall task has been completed or not (a finish task
         is a task that has been completed its execution, regardless if it is
         successful of failure)
Import: none
Export: boolean flag specifying the task completion
START
    Return isComplete
END


Module: hasError ()
Purpose: to determine if the task has finished with error or not
Import: none
Export: boolean flag specifying if an error exist or not
START
    Return hasError
END
```

6. GUI design

6.1 GUI layout

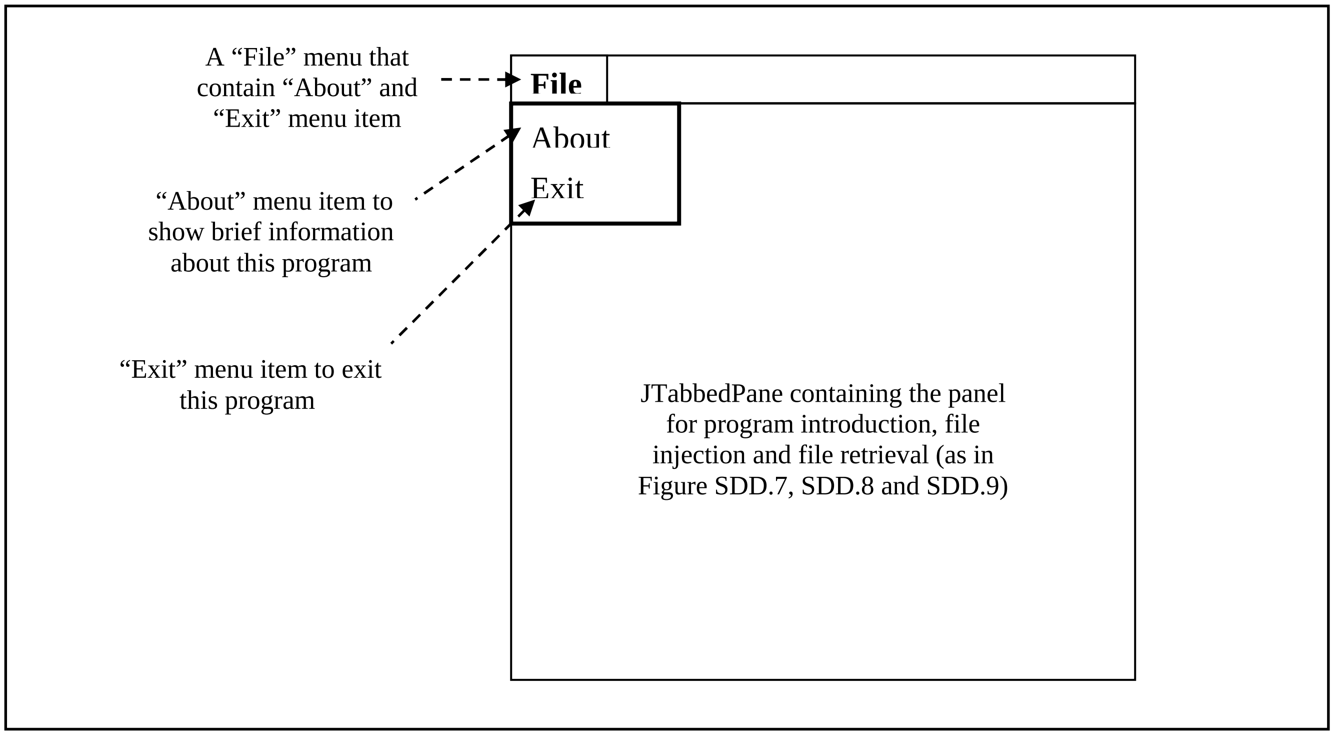
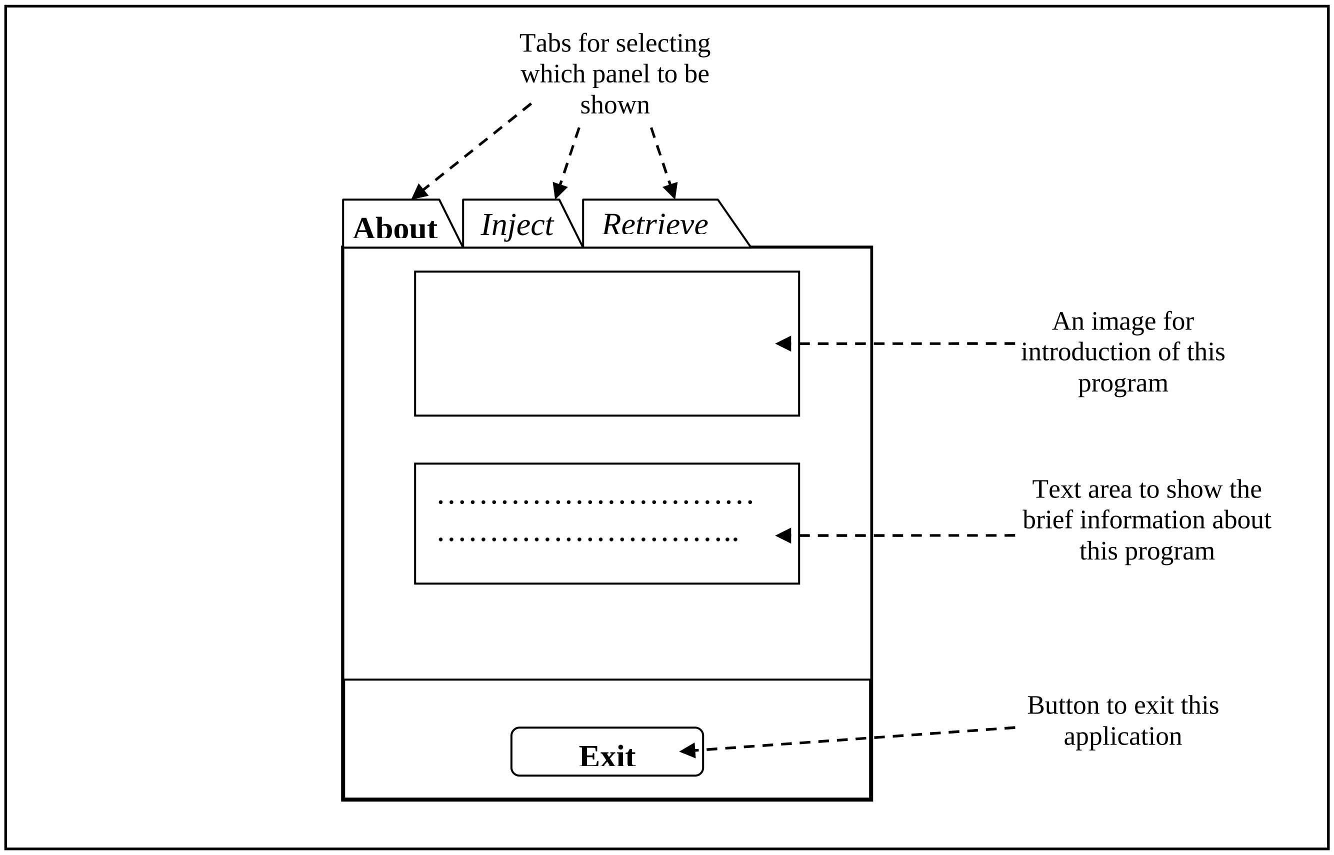
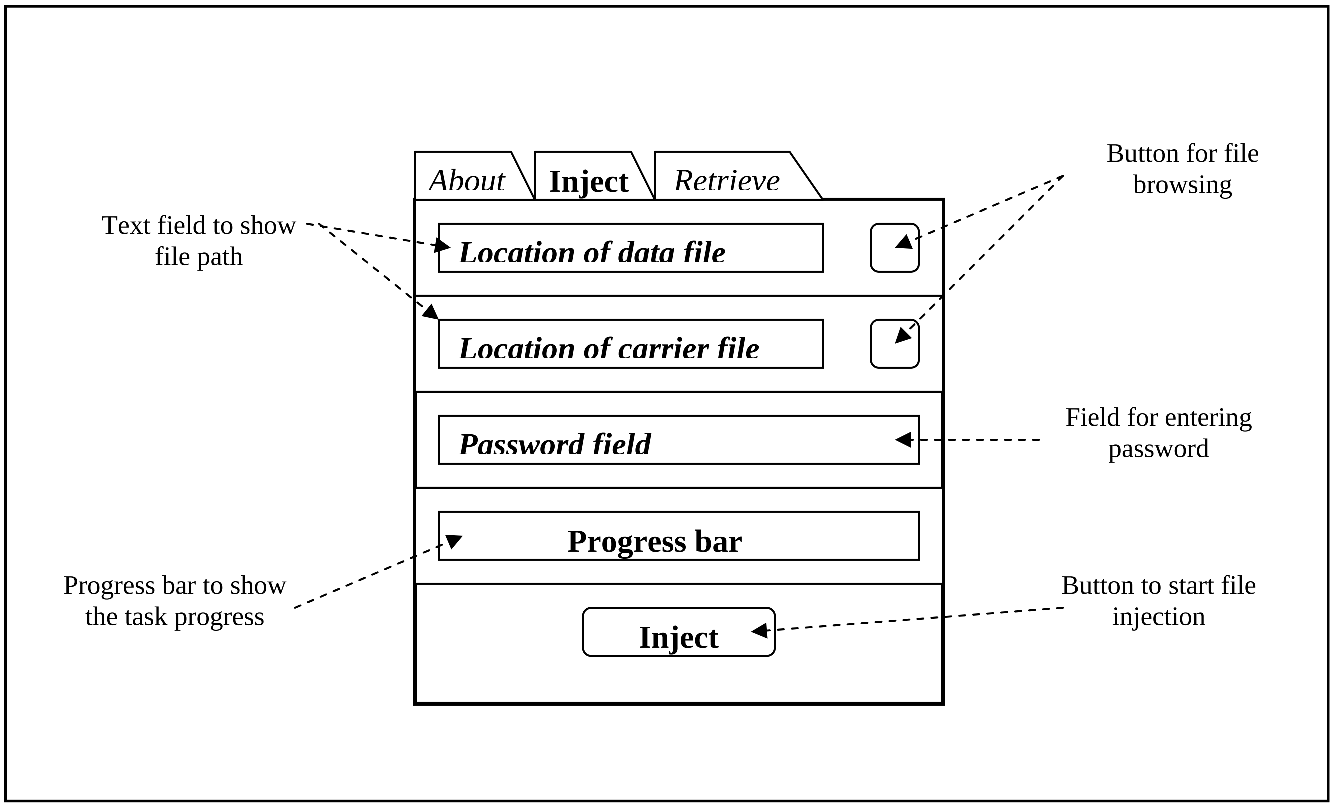

A “File” menu that contain “About” and “Exit” menu item
File
About
Exit
“About” menu item to show brief information about this program
“Exit” menu item to exit this program
JTabbedPane containing the panel for program introduction, file injection and file retrieval (as in Figure SDD.7, SDD.8 and SDD.9)

Figure SDD.6 – GUI layout for the main frame



Tabs for selecting which panel to be shown
About
Inject
Retrieve
An image for introduction of this program
…………………………
………………………..
Text area to show the brief information about this program
Exit
Button to exit this application

Figure SDD.7 – GUI layout showing the “About” panel



About
Inject
Retrieve
Button for file browsing
Text field to show file path
Location of data file
Location of carrier file
Password field
Field for entering password
Progress bar
Progress bar to show the task progress
Inject
Button to start file injection

Figure SDD.8 – GUI layout showing the “Inject” (for file injection) panel

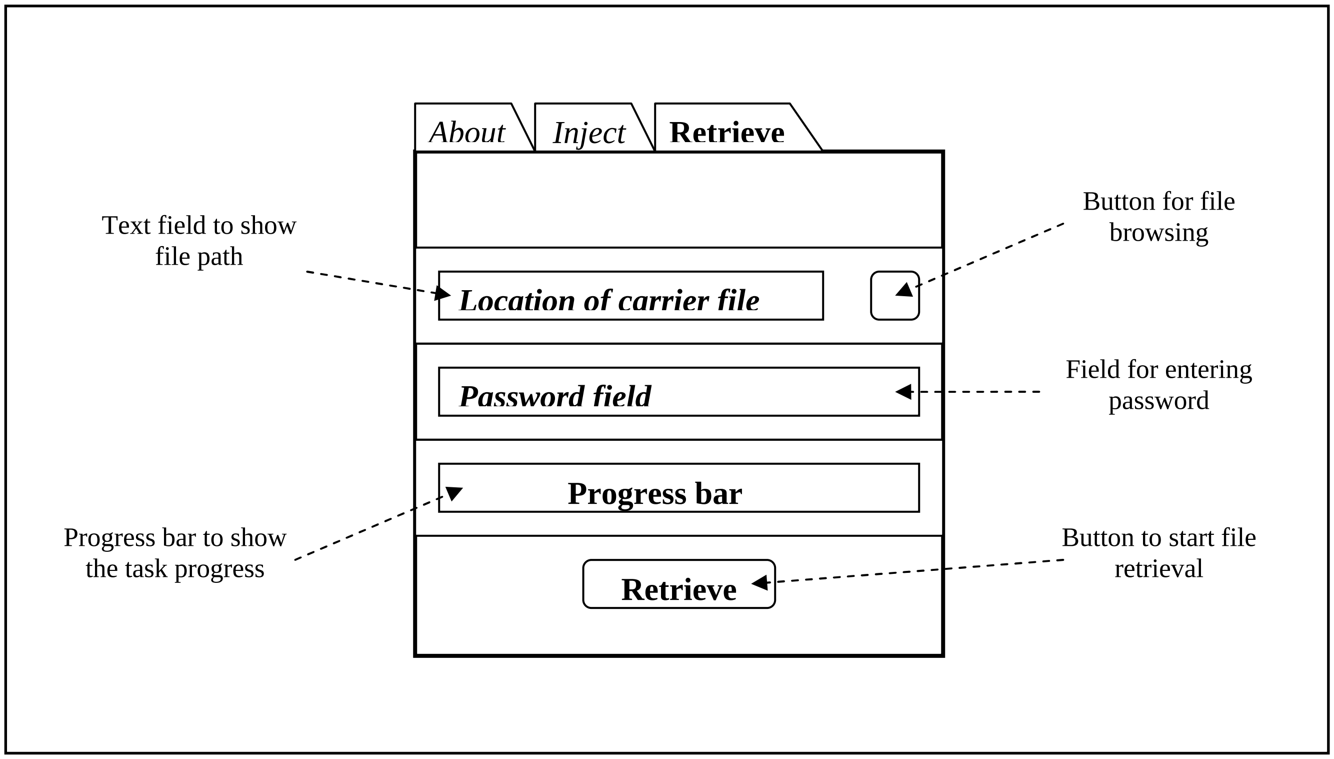

About
Inject
Retrieve
Text field to show file path
Location of carrier file
Button for file browsing
Password field
Field for entering password
Progress bar
Progress bar to show the task progress
Retrieve
Button to start file retrieval

Figure SDD.9 – GUI layout showing the "Retrieve" (for file retrieval) panel

6.2 Object component table

Table SDD.4 – Components for MainFrame

Object name	**Class (type)**	**Functions**
fileMenu	JMenu	- to contain the aboutItem and exitItem JMenuItem
aboutItem	JMenuItem	- to display brief information about this program
exitItem	JMenuItem	- to exit this program
tabPane	JTabbedPane	- to contain all the aboutPanel, injectPanle and retrievePanel of the program
aboutPanel	AboutPanel	- custom panel that contains all the component for the introduction panel of the program
injectPanel	InjectPanel	- custom panel that contains all the components for the file injection GUI
retrievePanel	RetrievePanel	- custom panel that contains all the components for the file retrieval GUI

Table SDD.5 – Components for AboutPanel

Object name	Class (type)	Functions
label	JLabel	- to display the introduction icon for this program
textArea	JTextArea	- to show some brief information about this application
exit	JButton	- to exit the application

Table SDD.6 – Components for InjectPanel

Object name	Class (type)	Functions
fileDisplayData	JTextField	- to display the selected data file from the file chooser
fileDisplayCarrier	JTextField	- to display the selected carrier file from the file chooser
browseData	JButton	- to open up the file chooser dialog for selecting a data file
browseCarrier	JButton	- to open up the file chooser dialog for selecting a carrier file
passField	JPasswordField	- the field for user to enter a password
pb	JProgressBar	- to show the user about the current status of a running task
inject	JButton	- to start the file injection task
fc	MyFileChooser	- provide GUI for user to select a file

Table SDD.7 – Components for RetrievePanel

Object name	Class (type)	Functions
fileDisplayCarrier	JTextField	- to display the selected carrier file from the file chooser
browseCarrier	JButton	- to open up the file chooser dialog for selecting a carrier file
passField	JPasswordField	- the field for user to enter a password
pb	JProgressBar	- to show the user about the current status of a running task
retrieve	JButton	- to start the file retrieval task
fc	JFileChooser	- provide GUI for user to select a file

Appendix

A. Bitmap file format

In a standard bitmap file, the first 54 bytes are the bitmap header information and must not be modified to avoid bitmap corruption. For example, a 80 x 80 pixels blank (all white) bitmap file has the following data in hexadecimal, as shown in Table A.1.

Table A.1 – Content of a carrier bitmap file in hexadecimal representation

Address	Data
`00000000`	`42 4d 36 4b 00 00 00 00 00 00 36 00 00 00 28 00`
`00000010`	`00 00 50 00 00 00 50 00 00 00 01 00 18 00 00 00`
`00000020`	`00 00 00 4b 00 00 00 00 00 00 00 00 00 00 00 00`
`00000030`	`00 00 00 00 00 00 ff ff ff ff ff ff ff ff ff ff`
`00000040`	`ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff`
...	...

From Table A.1, the first 54 bytes are the heard information for the bitmap file. The header file contains the type of the file, the height and width of the bitmap image and so on. The actual colour data starts with the first 'ff' value. As long as the header information is not illegally modified (modified externally), the bitmap file will not corrupt. As of FIRA2, it is actually the colour data that is being modified to inject a file.

File Injection and Retrieval Application version 2 (FIRA2)

Software Test Document (STD)

<u>Written by:</u>

Full name: Lai Chiong Ching

Nickname: LCC Corps

Last updated: July 18, 2006

Table of content

Contents ***Page***

1. Test design specification

Each major class (excluding the GUI class) for FIRA2 has been incorporated with their own unit test class. The FIRA2 major classes with their corresponding unit test class are as shown in Table STD.1. Table STD.1 also shows the corresponding test method developed to test the methods in the major class.

Table STD.1 – Major classes in FIRA2 with their test class

Major class	Unit test class
Password.java	PasswordTestCase.java
MyFile.java	MyFileTestCase.java
FileAccess.java	FileAccessTestCase.java
FileCompression.java	FileCompressionTestCase.java
FileCrypto.java	FileCryptoTestCase.java
FileInjection.java	FileInjectionTestCase.java
Task.java	TaskTestCase.java

1.1 PasswordTestCase

1.1.1 testPasswordConstrutor()

This test method is to test if the Password object has been constructed as expected. The test is performed by constructing a Password object using a prototype array of byte. Then, get the password using the Password object and compared with the prototype.

1.1.2 testCloneBytes()

This test method is used to test the cloneBytes() in Password.java. The test is done by copying a prototype array of byte and then compared them both (the prototype and the copied).

1.1.3 testGetUniqueID()

This test method is to test the getUniqueID(). There are two cases in this test. The first case of the test is performed by constructing two similar Password object and compare their corresponding unique ID. The second case is to create two different Password object and compare their corresponding unique ID.

1.1.4 testIsValid()

This test method is to test the isValid(). There are three cases in this test, namely to test the validity of a valid password (with length 2-20 characters) and two invalid passwords (with length less than 2 and greater than 20 characters).

1.2 MyFileTestCase

1.2.1 testGetExtension()

This test is performed by creating a MyFile object with a dummy filename and a prototype extension. Then, the file extension from MyFile object is being compared with the prototype.

1.2.2 testGetParent()

This test is performed by first creating a MyFile object. Then, the String returned my the getParent() is being compared with the String returned by the getAbsolutePath() with the filename removed.

1.2.3 testGetParentFile()

This test need the getParent() to work as expected. The test is done by comparing the MyFile object returned by the getParentFile() with the new MyFile object created from the String returned by the getParent().

1.2.4 testRemoveExt()

This test is performed by creating a MyFile object with a prototype filename (without extension) and a dummy file extension. Then, the value returned by the removeExt() will be compared with the prototype filename.

1.2.5 testRenameExt()

This test is performed by creating a MyFile object with a dummy filename and an old file extension. Then, the value returned by the renameExt(new extension) is compared with the concatenation of dummy filename and new extension.

1.2.6 testRenameTo()

This test is done by creating two different MyFile object (and make sure that both of the files do exists). Then, the first file is being renamed to the second file. After that, both the MyFile object is being compared.

1.2.7 testTruncateName()

This test is performed by first creating a MyFile object with a very long filename (more than 20 characters long). Then, compare the value returned by truncateName() with the the one manually truncated (using substring()).

1.3 FileAccessTestCase

1.3.1 testEOF()

There are two cases associated with this test. The first test is to purposely make the FileAccess object reach end-of-file by using the seek() and compared the value returned by eof() with the expected result. The second case is to use the seek() to NOT causing the end-of-file, and compare the result with expected value.

1.3.2 testInjectByte()

This test is done by injecting a test byte into a test file. Then, all (only four) the bytes that contain the information of the injected test byte is read into a buffer, and have the original byte rebuilt back. Then, the rebuilt byte is compared with the original byte.

1.3.3 testRetrieveByte()

This test need the injectByte() in FileAccess to be working properly. The test is done by injecting a test byte using the injectByte(). Then, the injected byte is retrieved back using the retrieveByte() and has its returned value compared with the original value.

1.4 FileCompressionTestCase

1.4.1 testZip()

This test method is to test both the zip() and compressFile() in FileCompression since both of them are related. The test is done by compressing a file using the zip(). Then, the zip entry is obtained from the resulting compressed file and is compared with the original test file (Note that only their file name is being compared. Besides, the resulted compressed file can be opened using any zip tools, like WinZip or WinRAR).

1.4.2 testUnzip()

This test method is to test the unzip() and the extractFile() in FileCompression. This test method uses the resulted compressed file from testZip(). It will decompress the file and then compare the decompressed file content with the original file content.

1.5 FileCryptoTestCase

In order to test the output for encrypted and decrypted file, a stand-alone application related to file encryption/decryption is required. This stand-alone application has already been written by the author during the development of FIRA2 as part of self-learning about javax.crypto.* packages. From this stand-alone application, a test file and its encrypted file are used as comparison samples.

1.5.1 testEncrypt()

This test is done by encrypting a file using the doCrypto() in FileCrypto. The output, encrypted file content is being compared with the comparison sample mentioned above.

1.5.2 testDecrypt()

This test is done by decrypting an encrypted file using the doCrypto() in FileCrypto. The output, decrypted file content is being compared with the comparison sample mentioned above.

Once both testEncrypt() and testDecrypt() is successful, they can be modified to encrypt a test file, and then decrypt it. The output file (from file decryption) is then compared with the original test file (before file encryption).

1.6 FileInjectionTestCase

1.6.1 testGetMaxLength()

The only test for getMaxLength() is to test that the returned value is a positive numbers (never a negative value).

1.6.2 testGetStartPosition()

This test is to determine that the value returned by getStartPosition() is always between 60 to max length (inclusively). The value of max length is determined by getMaxLength().

1.6.3 testGetNextPosition()

This test is to determine that the value returned by getNextPosition() will be between 60 to max length (inclusively), and that the values will not overlap one another.

1.6.4 testCanInject()

This test is done by comparing the result/value returned by canInject() with the expected value. There are two cases associated with this test. The first case is to have a small data file with sufficiently large carrier file so that the data file can be injected. The second case is to have a very large data file so that it cannont be injected.

1.6.5 testHasInject()

The test is done by comparing the value returned by hasInject() with expected value. The three cases related to this test are:

1. a carrier file with hidden file AND correct password
2. a carrier file with hidden file BUT with wrong password
3. a carrier file without any hidden file (password is unrelevant)

1.7 TaskTestCase

1.7.1 testInjectRetrieve()

This test method will test both doInject() and doRetrieve(). The test is performed by first inject a test file into a dummy carrier file. Then, the hidden test file is retrieved from the dummy carrier file. The content of the retrieved test file is compared with the content of the original test file.

2. Test procedure

2.1 Unit testing

The development method adopted for FIRA2 is hierarchical development[1]. Hence, the unit test has been carried out in parallel to the FIRA2 development. In other words, as a method is being developed, its corresponding unit test is also developed to test the method. Only after the unit test succeeds then the next method required for FIRA2 is being developed. The purpose of adopting this test procedure is to enable early bug/error detection before the bug/error creep deeper into the program. Most of the test performed is only black-box and gray-box testing.

2.2 System and Regression test

Once the first workable FIRA2 is completed, it is being run and modified multiple times to correct some minor tweaks as well as to remove some redundant code. Hence, during this phase, regression test is being vigorously performed to ensure that the modification will not cripple the program and the program will continue to work as expected.

2.3 Acceptance test

Once the FIRA2 program has been finalized (no more modification to its source codes), acceptance test will be performed to determine that FIRA2 will perform according to its requirements as stated in SRS.

[1] In hierarchical development, all the small components/classes at the bottom of the hierarchy are being developed first before they are combined to form a larger component at the top of hierarchy and finally form a complete program.

3. Test result

3.1 PasswordTestCase

```
=== Test cloneBytes() ===
result   = abc@123
expected = abc@123
testCloneBytes().....OK

=== Test PasswordConstructor() ===
result   = 0123456789abcdefghijklmno
expected = 0123456789abcdefghijklmno
testPasswordConstructor().....OK

=== Test isValid() ===
Test #1:
result   = true
expected = true
testIsValid().....OK

Test #2:
result   = false
expected = false
testIsValid().....OK

Test #3:
result   = false
expected = false
testIsValid().....OK

=== Test getUniqueID() ===
Test #1: Similar ID
result   = true
expected = true
testGetUniqueID().....OK

Test #2: Different ID
result   = false
expected = false
testGetUniqueID().....OK
```

3.2 MyFileTestCase

```
=== Test removeExt() ===
result   = test0123456789abcdefghijklmnopqrstuvwxyz
expected = test0123456789abcdefghijklmnopqrstuvwxyz
testRemoveExt ().....OK

=== Test renameExt() ===
result   = test0123456789abcdefghijklmnopqrstuvwxyz.datum
expected = test0123456789abcdefghijklmnopqrstuvwxyz.datum
testRenameExt ().....OK

=== Test truncateName() ===
result   = test0123456789abcdef.dat
expected = test0123456789abcdef.dat
testGetShortName ().....OK

=== Test getParent() ===
result   = C:\Documents and Settings\aurora.lai\Desktop\My
Works\SOSC\FIRA2\GUI\FIRA2\src\testcase\testFile\testMyFile
expected = C:\Documents and Settings\aurora.lai\Desktop\My
Works\SOSC\FIRA2\GUI\FIRA2\src\testcase\testFile\testMyFile
testGetParent().....OK

=== Test getParentFile() ===
result   = C:\Documents and Settings\aurora.lai\Desktop\My
Works\SOSC\FIRA2\GUI\FIRA2\src\testcase\testFile\testMyFile
expected = C:\Documents and Settings\aurora.lai\Desktop\My
Works\SOSC\FIRA2\GUI\FIRA2\src\testcase\testFile\testMyFile
testGetParentFile().....OK

=== Test renameTo() ===
Before rename   = C:\Documents and Settings\aurora.lai\Desktop\My
Works\SOSC\FIRA2\GUI\FIRA2\src\testcase\testFile\testFileInjection\abc.txt
After rename   = C:\Documents and Settings\aurora.lai\Desktop\My
Works\SOSC\FIRA2\GUI\FIRA2\src\testcase\testFile\testFileInjection\hello.world
Target rename  = C:\Documents and Settings\aurora.lai\Desktop\My
Works\SOSC\FIRA2\GUI\FIRA2\src\testcase\testFile\testFileInjection\hello.world
testRenameTo().....OK

=== Test getExtension() ===
result   = dat
expected = dat
testGetExtension().....OK
```

3.3 FileAccessTestCase

```
=== Test eof() ===
Test 1
result   = true
expected = true
testEOF ().....OK
Test 2
result   = false
expected = false
testEOF ().....OK

=== Test injectByte() ===
Result :
 1 1 1 0 1 0 1 0
Expected :
 1 1 1 0 1 0 1 0
result = 87
Expected = 87
testInjectByte ().....OK

=== Test retrieveByte() ===
Result :
 0 1 0 1 0 1 1 0
Expected :
 0 1 0 1 0 1 1 0
result = 106
expected = 106
testRetrieveByte ().....OK
```

3.4 FileCompressionTestCase

```
*=*=* Test zip() a single file *=*=*

=== Test zip() ===
Result :test.bmp
Expected :test.bmp

testZip().....OK

*=*=* Test unzip() file to "testUnzip" folder *=*=*

=== Test unzip() ===
Checking the content of extracted file against the original file before the
compression...
testUnzip().....OK
```

3.5 FileCryptoTestCase

```
=== Test decrypt() ===
Contents of encrypted file and expected sample file are the SAME.
testEncrypt().....OK

=== Test decrypt() ===
Contents of decrypted file and expected sample file are the SAME.
testDecrypt().....OK
```

3.6 FileInjectionTestCase

```
=== Test getStartPosition() ===
startPos = 160 (between 60-6000): true
testGetStartPosition().....OK

=== Test getNextPosition() ===
Test with 100 sample (generated automatically)
nextPos = 65 (between 60-6000): true
nextPos = 69 (between 60-6000): true
nextPos = 74 (between 60-6000): true
nextPos = 80 (between 60-6000): true
nextPos = 84 (between 60-6000): true
............
nextPos = 524 (between 60-6000): true
nextPos = 530 (between 60-6000): true
nextPos = 535 (between 60-6000): true
nextPos = 540 (between 60-6000): true
nextPos = 544 (between 60-6000): true
testGetNextPosition().....OK

=== Test getMaxLength() ===
maxLength = 660 (greater than zero): true
testGetMaxLength().....OK

=== Test canInject() ===
Test #1: Able to be injected (with small file)
result = true
expected = true
testCanInject().....OK

Test #2: Unable to be injected (with big file)
result = true
expected = true
testCanInject().....OK

=== Test hasInject() ===
Test #1: With injected file
result = true
expected = true
testHasInject().....OK

Test #2: With injected file (but wrong password)
result = false
expected = false
testHasInject().....OK

Test #3: Without injected file
result = false
expected = false
testHasInject().....OK

=== Test doInject() AND doRetrieve() ===
The original data file: test.txt
The retrieved file: retrieved.txt
The content of original and retrieved file is same: true
Test doInject() AND doRetrieve().....OK
```

3.7 TaskTestCase

```
=== Test inject() AND retrieve()===
NOTE: This test may take a while to complete...
The original test file: src\testcase\testFile\testTask\original\data.txt
The retrieved file: C:\Documents and Settings\aurora.lai\Desktop\
         My Works\SOSC\FIRA2\GUI\FIRA2\src\testcase\testFile\testTask\data.txt
The content of original and retrieved file are SAME.
testInjectRetrieve().....OK
```

4. Test summary

All the tests have been performed by the author, LCC Corps, along with the development of FIRA2.

File Injection and Retrieval Application version 2 (FIRA2)

User Manual

<u>Written by:</u>

Full name: Lai Chiong Ching

Nickname: LCC Corps

Last updated: July 22, 2006

Table of content

Preface

The original idea for this application is obtained from the article *Hiding a text file in bmp file* by Ahmed Osama (Adore C++), at http://www.codeproject.com/cpp/HideIt.asp and from *Hide a file in a BMP file (plus)* by Neil Xu at http://www.codeproject.com/cpp/HideIt2.asp.

This application is developed solely for educational purpose and of personal interests. Illegal misuse of this application by anyone is by NO means liable to the author.

1. Introduction

From times to times, people often require some means to convey their private messages or files to their friends or other acknowledged recipients. Although presently, there exist many applications that offers file encryption and password protection, these software lack of one feature, that is the ability of conveying data (files) without catching the attention of the prying eyes.

Therefore, this application '*File Injection and Retrieval Application*' version 2 or FIRA2 in short has been developed to solve this lack. It is capable of injecting data files into a carrier file (a bitmap image file). The carrier file containing the data file is completely indistinguishable by naked eyes, thus enable data to be conveyed without anyone's attention. Besides, the carrier file is also password protected, which required the authorized recipient to enter the correct password in order to retrieve the hidden data file in the carrier file.

2. Requirement

Minimum requirement:
Pentium II 200MHz
Window 98
64 MB RAM memory
2 MB hard disk space
[♣]Installed Java SDK 1.4

Recommended requirement:
Pentium III 500MHz or above
Window 98, Window 2000 or Window XP
128 MB RAM memory or above
2 MB hard disk space
[♣]Installed Java SDK 1.4.2 or above

[♣] Latest version of Java SDK can be obtained from java.sun.com

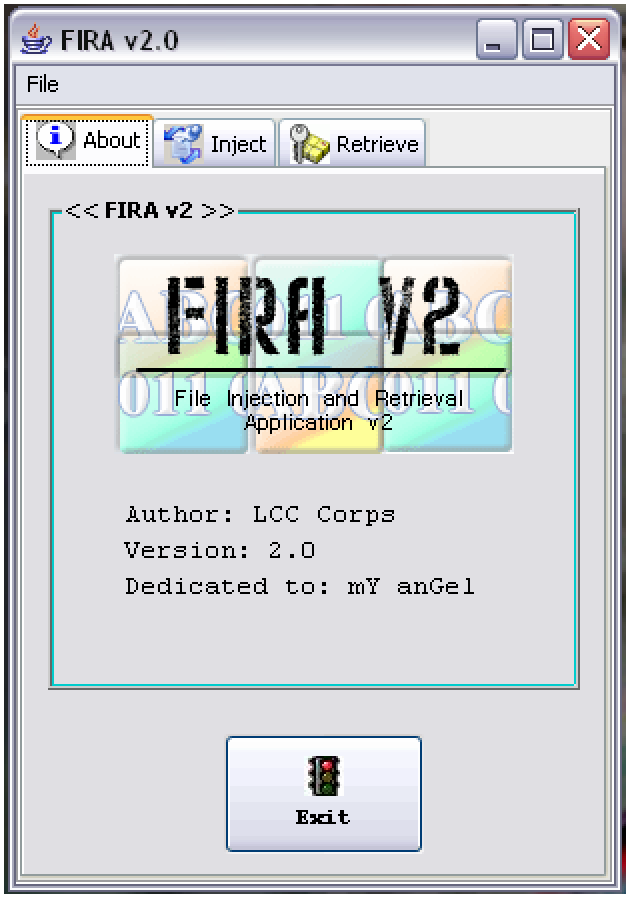
3. Installation

This project is zipped into a single file called FIRA2.zip. Within FIRA2.zip, there will be a “Program” folder, “Source code” folder and “Documentation” folder. To install the application, simply use WinZip (www.winzip.com) or WinRAR (www.winrar.com) to unzip FIRA2.zip to a folder named FIRA2. The actual program will be located in /FIRA2/Program folder.

4. Executing the application

4.1 Running the application

To executing FIRA2, simply go to the “Program” folder and double-click the “fira2.bat” to execute the program. It may take a while for the program to load. After that, the program window will be shown, as in Figure ???.



Author: LCC Corps

Version: 2.0

Dedicated to: mY anGel

Figure UM 1 – The main window for FIRA2

4.2 Injecting a file

Once the application is successfully executed, click on the “Inject” tab.

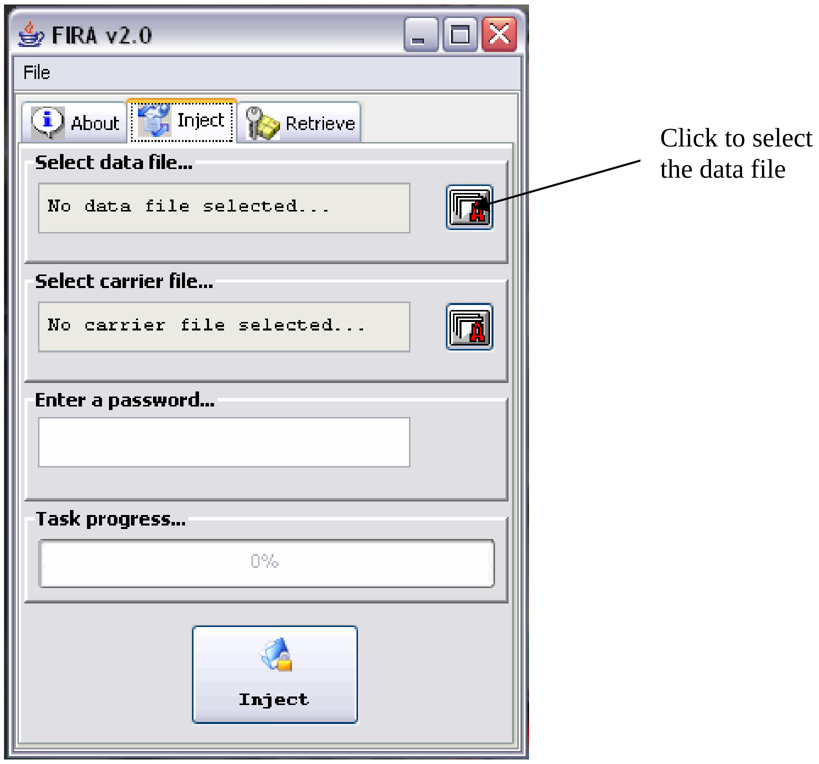


Figure UM 2 - The file injection panel for FIRA2, to select a data file

Then select the data file that you wish to be injected/hidden in the carrier file by clicking file browsing button. Then, a file chooser window will appear for the user to select the data file.

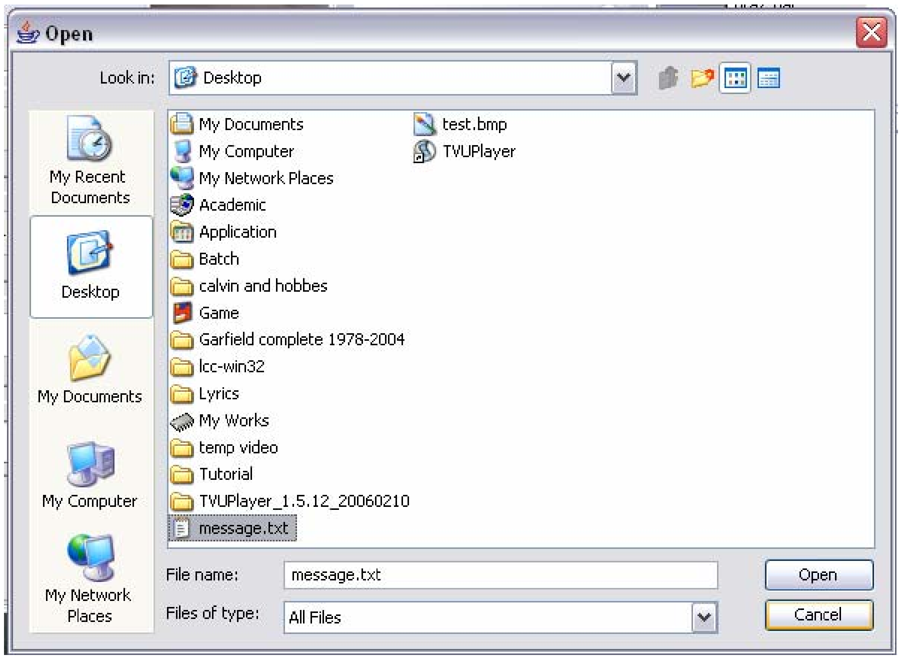


Figure UM 3 - File chooser window to select data file

After selecting the data file from the file chooser window, click the "Open" button.

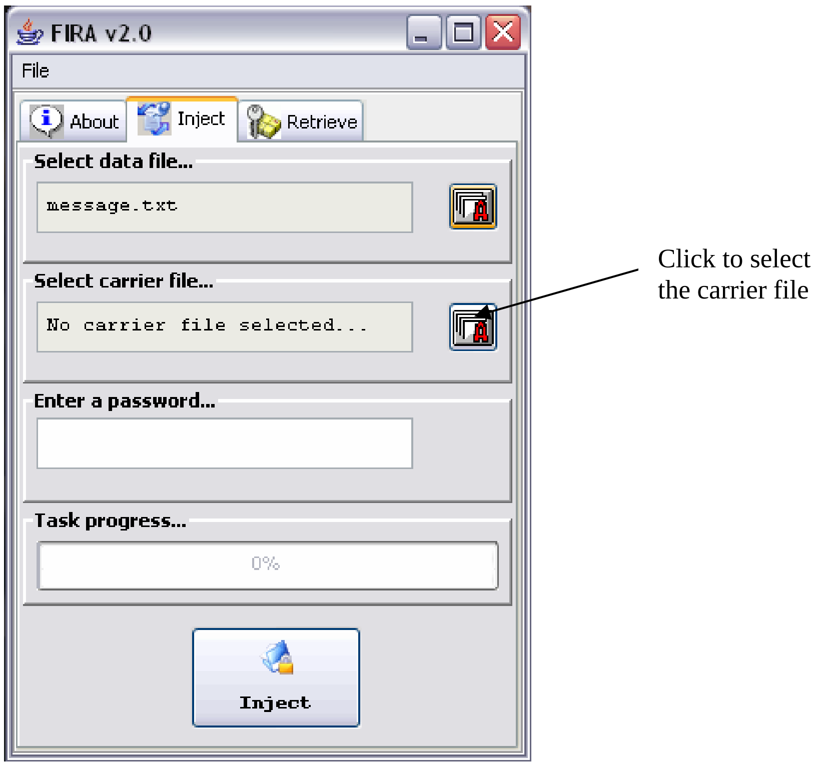


Figure UM 4 - Select a carrier file

Next, select a bitmap (carrier) file by clicking the file browsing button. Similarly, a file chooser window will appear where you can select a bitmap file as the carrier file.

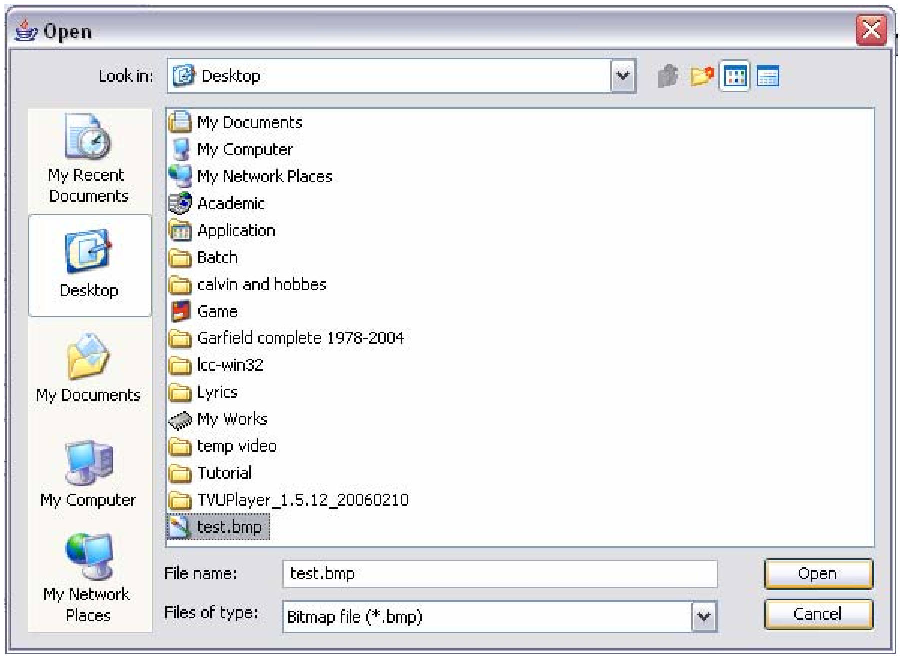
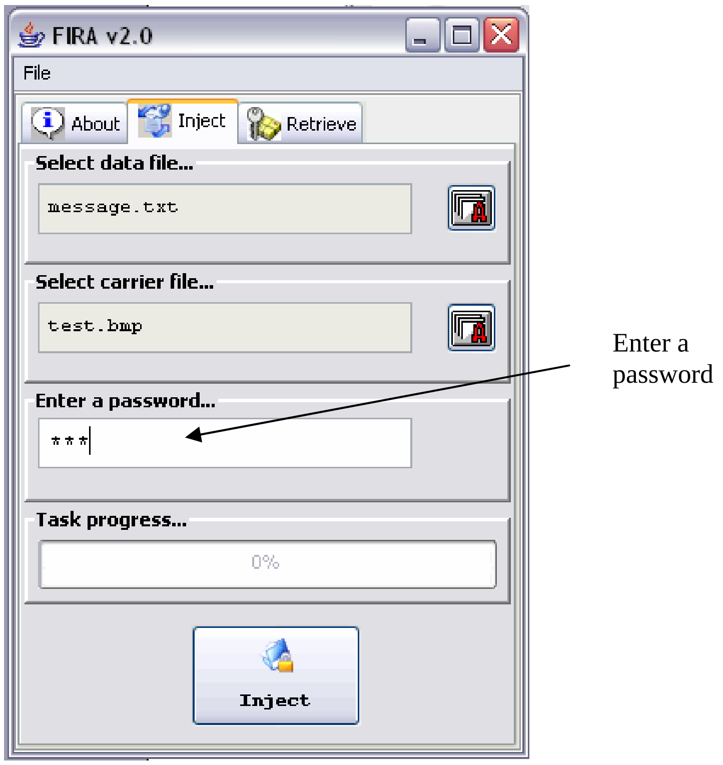


Figure UM 5 - File chooser window to select a carrier file

When finish selecting the carrier file, click 'Open'.



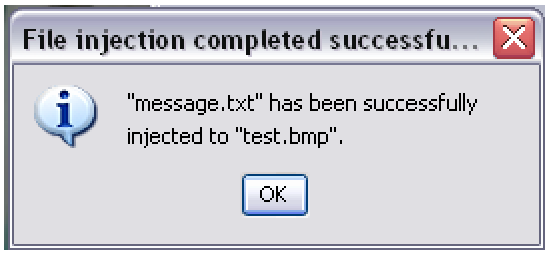
message.txt

test.bmp

***

Figure UM 6 - Enter desired password and ready for file injection

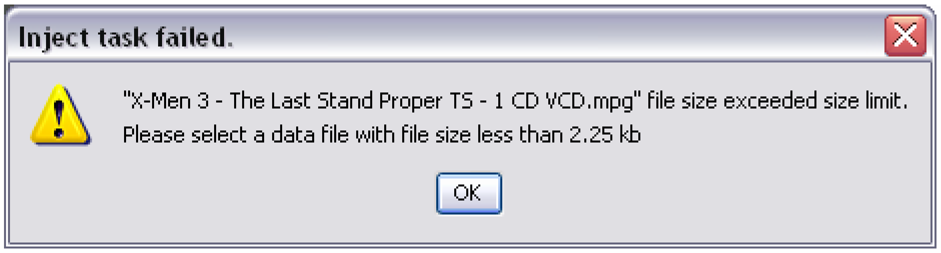
Then, enter your desired password in the provided field. The password needs to be between 2 to 20 characters long. Finally click the 'Inject' button to start the file injection process. When the file injection task has completed, a confirmation window will popup to inform you that the process is completed.



"message.txt" has been successfully injected to "test.bmp".

Figure UM 7 - Acknowledgement dialog on the completion of file injection process

If the file injection fails, the only reason may be that the data file size exceeds the injection capacity of the carrier file, meaning that the selected data file is too big to be injected into the selected carrier file. An error message will be shown to user if this failure occurs.



Inject task failed.

"X-Men 3 - The Last Stand Proper TS - 1 CD VCD.mpg" file size exceeded size limit.
Please select a data file with file size less than 2.25 kb

Figure UM 8 - Error message for file injection failure

Note that the indicated file size is only an estimate, since this program has the capability of file compression/decompression. As an example, for a text file with very high compression rate, a text file with file size of 3.0 mega-bytes can be safely injected into a bitmap file of only 300 kilo-bytes.

4.3 Retrieving a hidden data file

Click the 'Retrieve' tab.

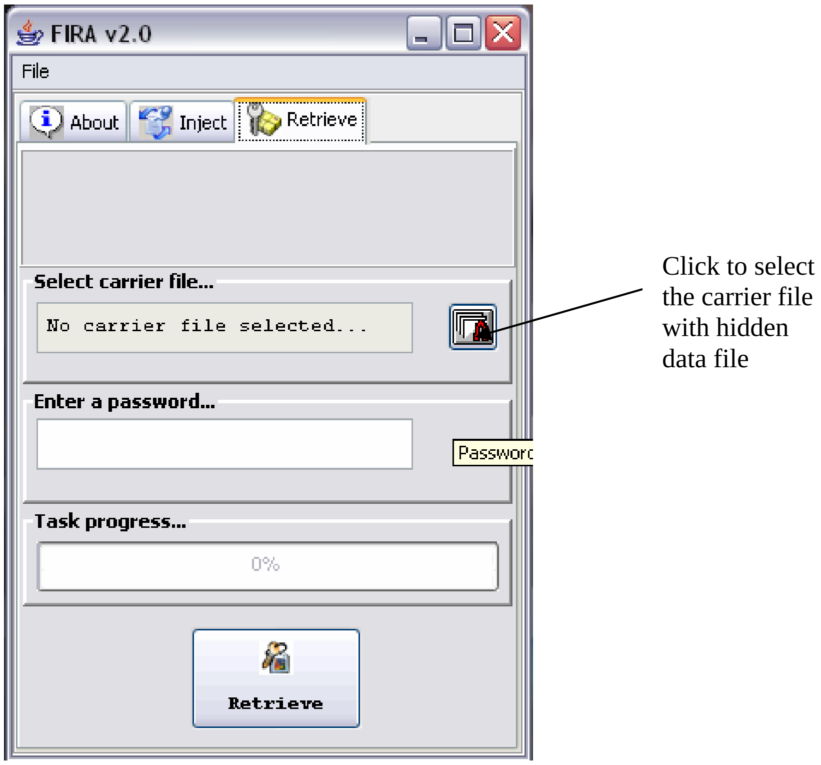
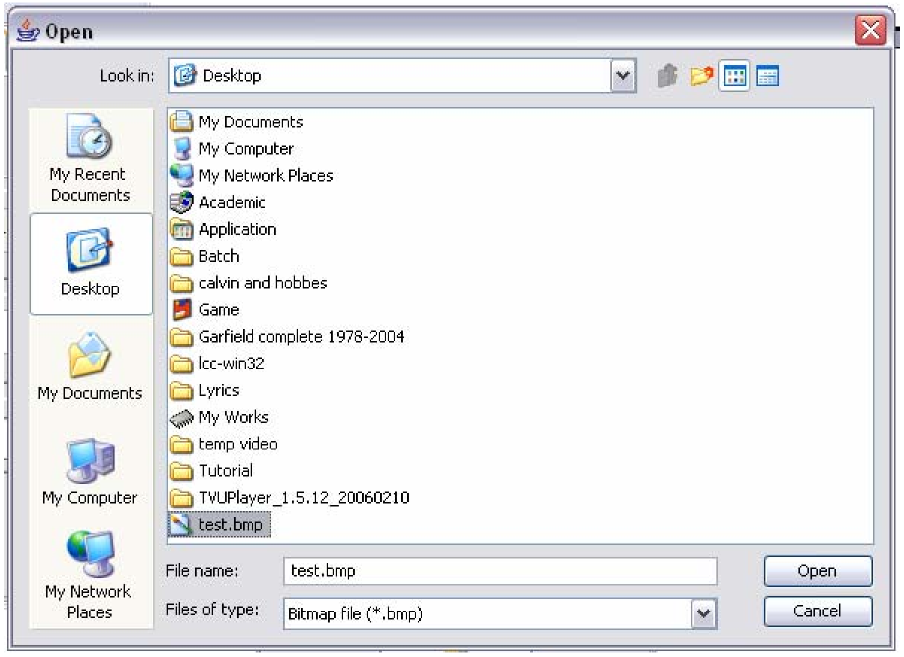


Figure UM 9 - File retrieval panel for FIRA2

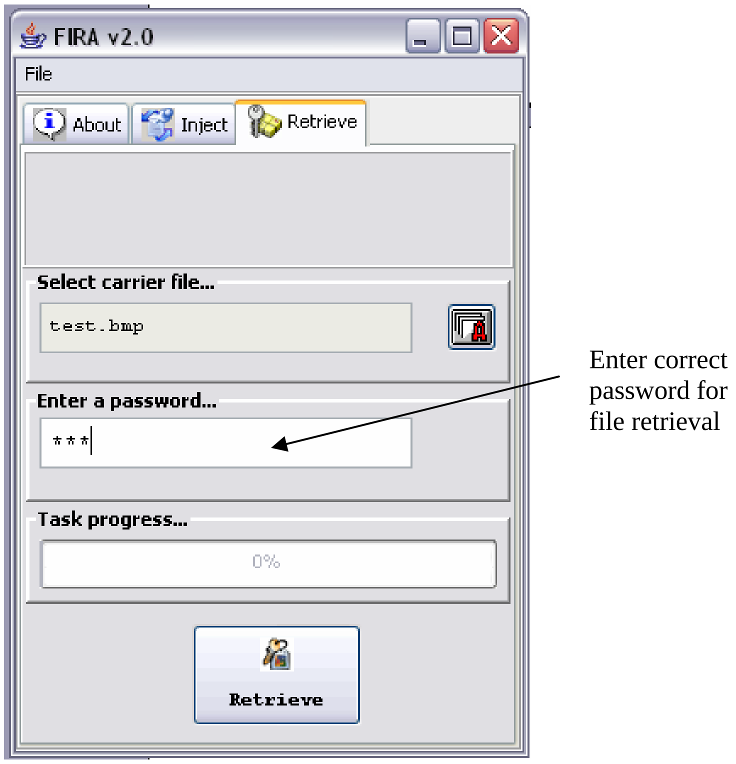
Click the file browsing to select the carrier bitmap file containing the hidden data file.



File name: test.bmp
Files of type: Bitmap file (*.bmp)

Figure UM 10 - File chooser window for selecting a carrier file

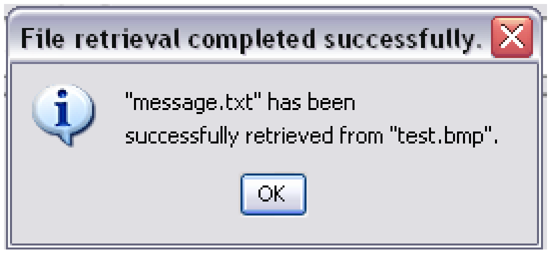
After finish selecting the carrier file with the hidden data file, click the 'Open' button.



Select carrier file... test.bmp
Enter a password... ***

Figure UM 11 - Enter correct password and ready for file retrieval

Next, enter the correct password and click the 'Retrieve file' button to begin hidden file retrieval process. Once the file retrieval process has completed, a confirmation window will appear.


File retrieval completed successfully.

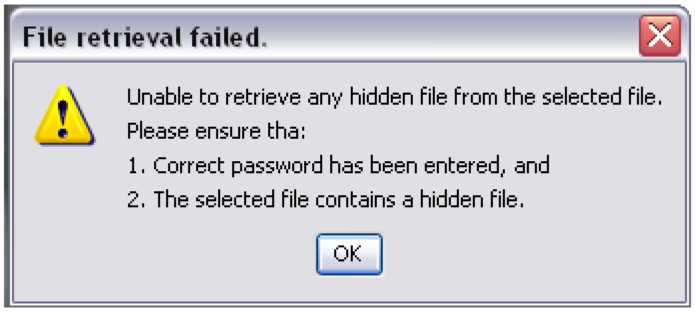
"message.txt" has been
successfully retrieved from "test.bmp".

Figure UM 12 - Acknowledgement dialog upon successful file retrieval process

If the file retrieval fails, there may only be two reasons for it.

1. An incorrect password is entered OR
2. The selected carrier bitmap file DOES NOT contain any hidden data file.

An error message dialog will appear to acknowledge the user about this failure.


File retrieval failed.

Unable to retrieve any hidden file from the selected file.
Please ensure tha:
1. Correct password has been entered, and
2. The selected file contains a hidden file.

Figure UM 13 - Error message for file retrieval failure

Troubleshooting/FAQ

Q1: Can other file types (e.g. *.jpg, *.gif, etc) be selected as carrier file?
A1: NO. This application is design to only utilize bitmap file (*.bmp) as carrier file.

Q2: What is a valid password to be used?
A2: A valid password is of any character, but must be between 2-20 characters in length.

Q3: I already retrieved the hidden data file from a carrier file, but I accidentally delete it (the retrieved file). I cannot retrieve the hidden data file again. Why?
A3: This application is design to enable only ONE successful retrieval per carrier file. That means for a carrier file, you can only retrieve the hidden data file once (successfully). After that the data is erased. The reason for this design is for security purpose.

Q4: I follow the maximum file size of the data file that can be injected, but I still get the same error message asking me to select a smaller data file. Why?
A4: The maximum data file size shown by the application is ONLY an estimation. It may NOT be 100% accurate. A good rule of thumb is to inject a data file that is 6 TIMES less than the carrier file. For example, if the carrier file is 3.0 Mb, then the data file should be best being at most 500 kb.

Q5: I finish retrieved the hidden data file successfully, but where is the file?
A5: The file is saved, by default, in the same directory as the selected carrier file.

Contact

For any comments, feedbacks and reporting bugs, please email to the author ‘LCC Corps’ at aurora.lai@gmail.com.