



& THE WRENCH OF DOOM

**Digisalt Lufia Translation Tools**  
a (hopefully somewhat) quick howto

# Index

Index.....	2
1Introduction.....	3
2Tool/File Usage.....	3
2.1Working with the text dumper.....	3
2.2The bulkdumper.....	9
2.3Translating text into your own language.....	9
2.4The checker tool.....	12
2.5Using the inserter.....	13

# 1 Introduction

As mentioned in the readme of the archive, this document will attempt to tell you how to work with the tools I created and how to create your own localized version of Lufia1. The tools created as I was translating (and learning a bit of C programming alongside) so they're not the really high sophisticated solutions to any problem faced. Instead they just fit my needs at the very time... Actually, they're even quite dumb. They don't automatically adjust any pointers (which was necessary sometimes) at all... so the good old hex editor will still be needed from time to time. Before actually releasing the tools alongside their sourcecode, I rewrote (most of) them to improve their code quality and enable other hackers to furtherly enhance them if they wish. Only the inserter tool which does the recompression of the script and inserts the correct hex bytes into the rom – this tool wasn't recreated as I somehow lost the original sourcecode for the tool... so it's only available as a windows exe compiled from C code :/

Oh, btw: I don't intend on furtherly developing the tools so don't expect any bugfixes or anything... in case you want to see something improved, you'll have to take care of it yourself. Sorry for the inconvenience ;)

## 2 Tool/File Usage

On the following pages, I'll explain the various tools, I used to translate the game into German tongue. Those are in particular:

- **The text dumper**
- **The bulkdumper**
- **The checker tool**
- **The text inserter**
- **The savaram fixer tool**

I'll also cover the names of the textfiles used in combination with the tools and the roms which are:

- **The original dictionary table – AllOriginal.tbl (see archive)**
- **The translated working dictionary table – SortedNew.tbl (see archive)**

The following chapter contains

### 2.1 *Working with the text dumper*

First of all, I'll have to describe the various text-files used by the tools as you'll want touse them as well using my tools. First of all, let me clarify one thing: The german translation of Lufia hardly changes any pointers within the game code... that is: the tools do NEVER change any pointers automatically at all. How can that be I hear you ask - well... you're right. German sentences are usually longer than their english counterparts so it is undeniable that most of the textboxes from the german translation are in fact larger than the english ones. How does the game enable this without modification of any pointers? It's simple. It uses compression. That's right folks. Lufias text is compressed. Lufia - like many other games of said generation uses something called dictionary or DTE compression. That means: the game contains a list of words that happen to appear regularly... it goes like this: the game contains multiple lists where all those words and syllables are standing right next to each other. Before that list of works, there is a pointertable which contains a pointer to each of the values. Let's just pretend the list of words contains such the word blade. That means that every dialog throughout the game containing either of the words will just reference the one occurrence in the rom where it is actually written in clear text. For example the following dialog with the number of characters written next to each line for your pleasure:

We need to find the	19
dual blade to defeat	20
the Sinistrals.	15

Then the size in the rom would be  $19 + 20 + 15$  bytes plus two additional bytes for the line breaks after the first and second line and one additional byte as the end of dialog marker. That would be  $19 + 20 + 15 + 2 + 1 = 57$  bytes, right?! Wrong! As Lufia uses the DTE compression, the word blade is only referenced in this dialog. That means instead of five bytes, it only takes two bytes as there's just a pointer stored instead of the whole word. That reduces the complete dialog to 54 bytes. Now imagine that blade is not the only word in the dictionary but most of the words you normally need to use. That means that you can store loads of text within only a few bytes. During the time when Lufia was released, it made sense to do such things as rom chips were expensive and developers strived to save space in order to keep production costs low. For the aspiring romhacker/translator this means he has to fiddle around with the DTE algorithm to first decompress the original text of the rom, then translate the text, create a quite good new dictionary and finally insert the compressed and translated text back into the rom. This can be quite tedious work. And granted, the first passages of text translated were just that. Hard work. I had to do it all by hand with Translhexion. This nice hexeditor supports tablefiles so you can at least read the original text in a good way but you have to compress all the new dialogs by hand. Obviously this wasn't the best way to do it so I decided to create some tools assisting me on my way to finishing my translation. The first one was a dumper tool. As I wasn't able to completely figure how the pointer schema worked pointing to the dialogs, I created a semi-automatic dumper. In order to dump all the dialogs from the rom, you need the original rom, the dumper.jar and the file AllOriginal.tbl which contains the complete dictionary of Lufia & The Fortress of Doom in tbl file format. Please take a look at the content of the file so you get an impression of how large the games dictionary really is. It IS quite extensive. That's for sure. Anyhow, if you want to dump the text of the rom, you need to run the dumper like this:

**`java -jar dumper.jar LufiaUs.smc 114538 AllOriginal.tbl`**

This will tell the dumper to open the rom at address 114538 (decimal) and to attempt decompression of the bytestream at that position. It will do so until it reaches an end of dialog marker (hex 04) so it might be that the program goes into an infinite loop if there is no such byte found thereafter. You might also want to use a hex address instead of a decimal one by adding one of the following characters in front of the address itself: h, x or \$. So the address 114538 may also be written as \$1BF6A, x1BF6A or h1BF6A instead. The result would always be the same. The hex string would be converted to the decimal address value and the rom would be opened at that position. What you'll see on the screen next is the following:

```

C:\WINDOWS\system32\cmd.exe - dumper001.exe LufiaE.smc 114538 AllOriginal.tbl
Lufia-dumper v1.0 by FloBo                                     (c) 2005 Digisalt
-----
input:      y      x      c      v      b      m      q
function: <<      <      >      >>      jump dump quit
-----

114538
Ma? W c escape!
114554

Maxin? We can't escape!
input: _

```

The first line is always the same. It just tells you some stuff about the program you're just using. The lines below show the characters you might want to press in order to handle the dumper. Below, you can see a preview of the dump of the current record the dumper has decompressed using the supplied dictionary file. The first number shows the start offset of the record. After that, there's a line of the compressed string. That is: The word Maxim for example is compressed to x0704. That is: it is two bytes long. Therefore the first two characters of the word are printed in the line. Both the exclamation mark and the space sign are uncompressed characters and are therefore the following two characters printed. Next, we have only a capital W as the word we is a dictionary word as. However it is even furtherly compressed than Maxim as it only takes one byte space in compressed for (x91=we). The same applies for the word can't. The word escape is not compressed at all so it appears in clear text in the overview line and in the decompressed text. Following the starting offset and the compressed line, there's a second number which represents the end of the record (that is the position of the end of dialog marker x04). Underneath the end offset, the decompressed text is being displayed. In this case all has worked fine and the dialog is well readable. So we want to dump this record into a textfile and advance to the next record. We can do so by pressing m and confirm by hitting return. The next screen looks a bit more weird:

```

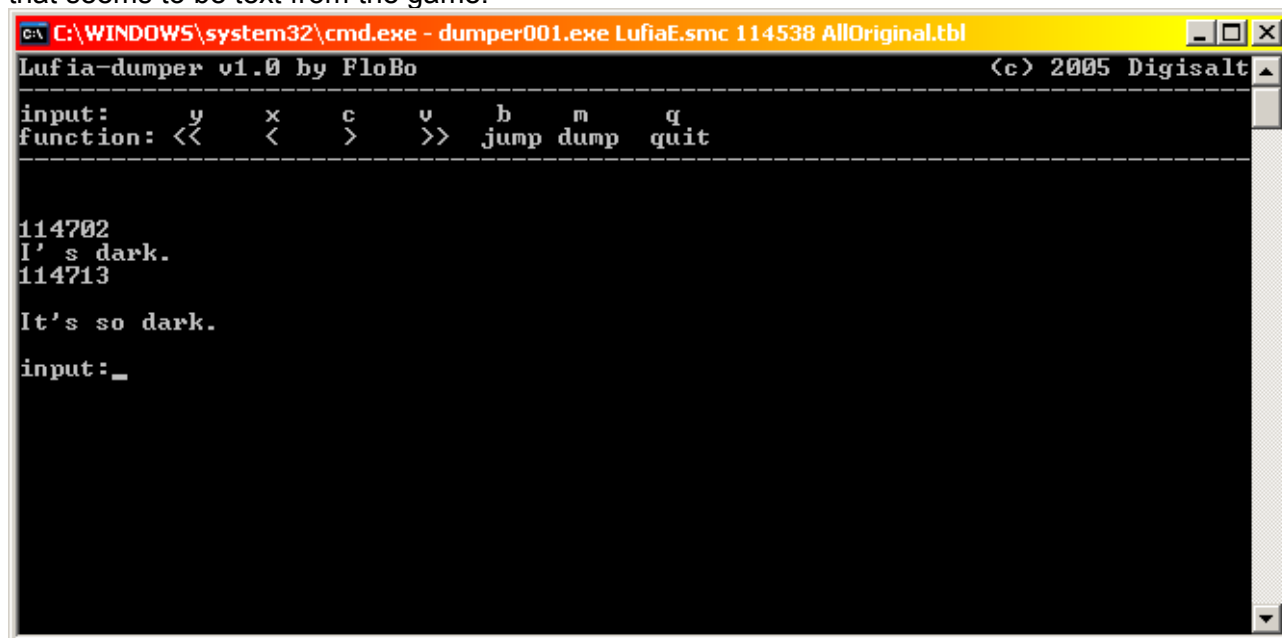
C:\WINDOWS\system32\cmd.exe - dumper001.exe LufiaE.smc 114538 AllOriginal.tbl
Lufia-dumper v1.0 by FloBo                                     (c) 2005 Digisalt
-----
input:      y      x      c      v      b      m      q
function: <<      <      >      >>      jump dump quit
-----

114555
<A_____P_____
114572
____<Alumina____P_____
input:

```

This is because it seems like this part of the rom doesn't contain any real dialog. You can notice this because of the underscores everywhere. Whenever a byte value was found that isn't

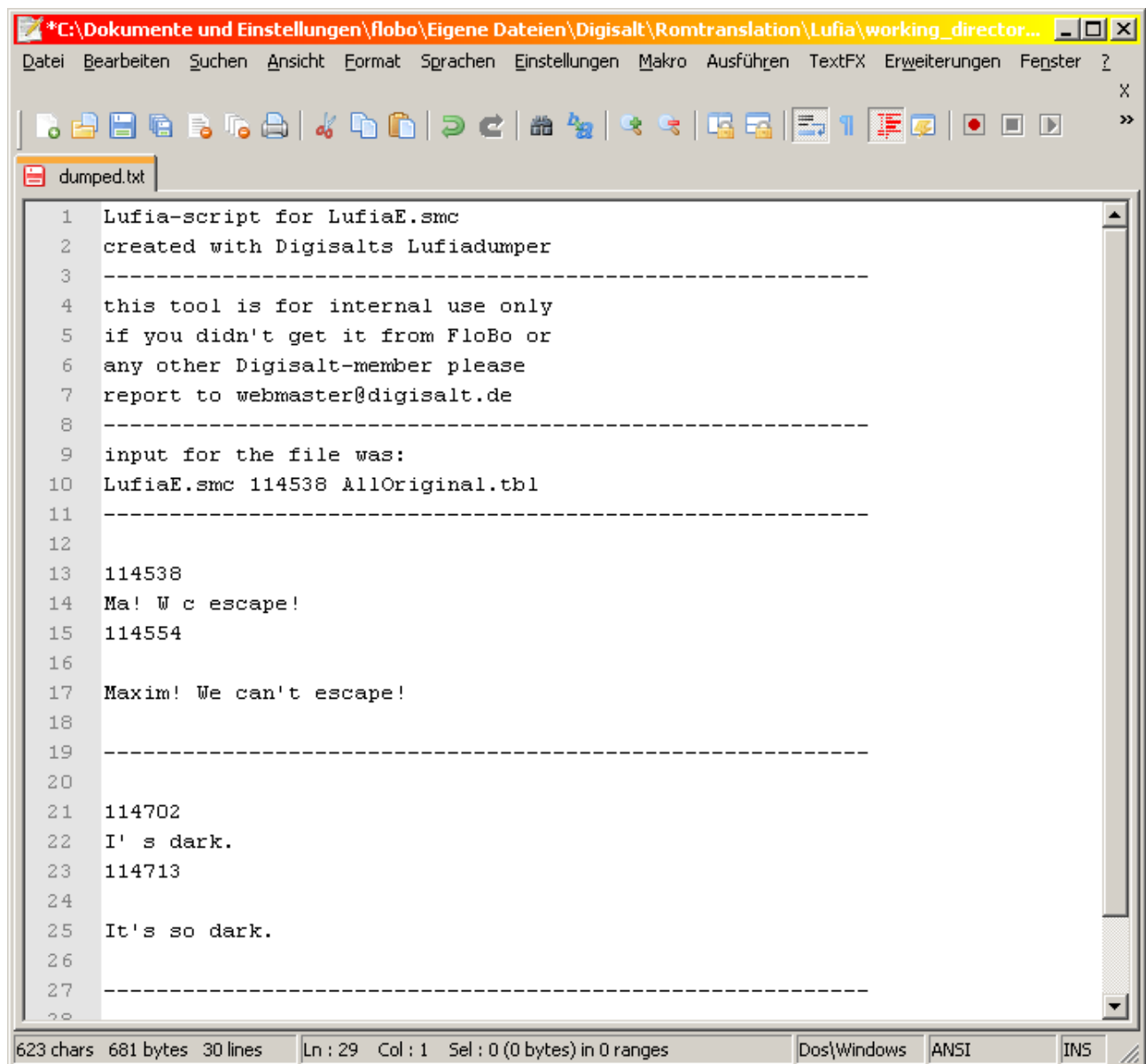
contained in the dictionary table, an underscore is printed. As also the last few characters of this blog are underscores, chances are near 100% this is no dialog text blog. So what we want to do now is skip over the bytes which don't contain any actual text. We can seek forward and backward we can use the keys x and c. If we want to seek more quick, we have to use y and v (note that these keys are all next to each other on a german keyboard, therefore they were chosen). If we really just want to skip the complete current blog and advance to the position after the current end of dialog marker, we can do so by pressing b. Having done that twice, we can see another string that seems to be text from the game:



Here we can press m again to dump the current record into the textfile. The dumper will add the record to our output textfile and advance to the next record. This procedure has to be done until you don't want to do it anymore or until you have finished dumping all records. Just for your information, when I translated Lufia, I dumped the complete script in seven parts so that it didn't always take forever to test some minor updates having to insert ALL dialogs again...

Phew. Now this is how dumping works. Once done with it make sure to terminate the dump by pressing q (NOTE: Do NOT just close the window as your dump might get messed up and your computers memory almost certainly will not be cleared completely). Update: as the dumper was rewritten in java, this shouldn't be the case anymore... nevertheless, it's still recommended to quit using q :)

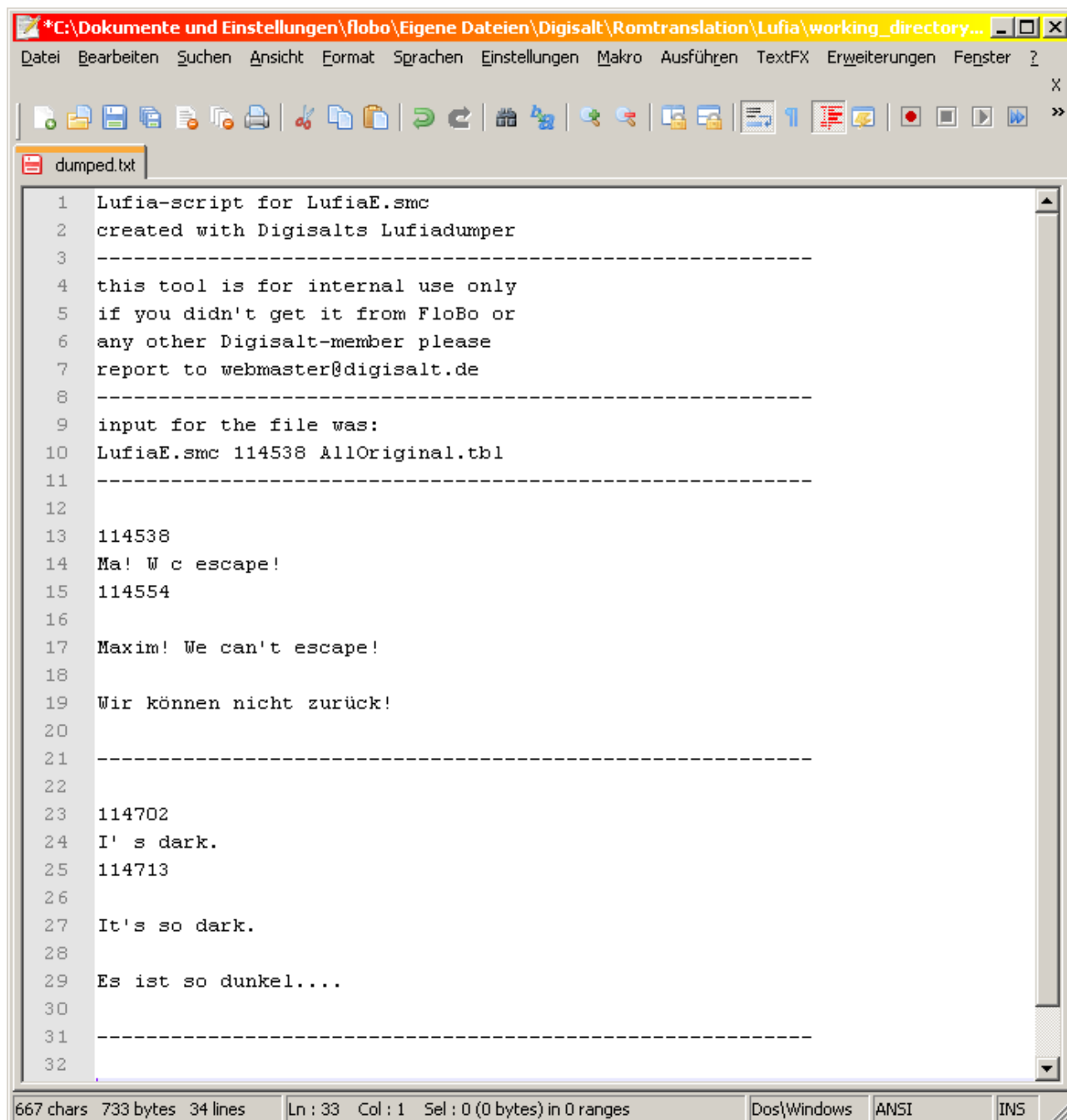
Once done, a new textfile should be in the directory your dumper.exe is located in. Its name should be dumped.txt. You might want to rename it to anything that makes sense to you. I just named my script dumps part000\_Lufia\_script.txt to part006\_Lufia\_script.txt... Opening the dumped.txt file, you'll see the following:



```
1 Lufia-script for LufiaE.smc
2 created with Digisalts Lufiadumper
3 -----
4 this tool is for internal use only
5 if you didn't get it from FloBo or
6 any other Digisalt-member please
7 report to webmaster@digisalt.de
8 -----
9 input for the file was:
10 LufiaE.smc 114538 AllOriginal.tbl
11 -----
12
13 114538
14 Ma! W c escape!
15 114554
16
17 Maxim! We can't escape!
18
19 -----
20
21 114702
22 I' s dark.
23 114713
24
25 It's so dark.
26
27 -----
28
29
```

623 chars 681 bytes 30 lines Ln : 29 Col : 1 Sel : 0 (0 bytes) in 0 ranges Dos\Windows ANSI INS

It's actually just a file containing exactly the data you previously saw at the console window printed into a textfile with a blabla-header above. And that's effectively the basis of your work thereafter. You'll just use these textfiles, insert a translated version of the text below the original one (separated by a blank line) and try to insert them using the inserter tool. It's as easy as that... well almost. The real challenges will be to create the correct DTE for your language and to fit all the necessary text in the limited space available. Anyhow, in principle, this is what your prepared script could look like before you try to compress it back into the rom:



```
*C:\Dokumente und Einstellungen\llobo\Eigene Dateien\Digisalt\Romtranslation\Lufia\working_directory...
Datei Bearbeiten Suchen Ansicht Format Sprachen Einstellungen Makro Ausführen TextFX Erweiterungen Fenster ?
X
dumped.txt
1 Lufia-script for LufiaE.smc
2 created with Digisalts Lufiadumper
3 -----
4 this tool is for internal use only
5 if you didn't get it from FloBo or
6 any other Digisalt-member please
7 report to webmaster@digisalt.de
8 -----
9 input for the file was:
10 LufiaE.smc 114538 AllOriginal.tbl
11 -----
12
13 114538
14 Ma! W c escape!
15 114554
16
17 Maxim! We can't escape!
18
19 Wir können nicht zurück!
20
21 -----
22
23 114702
24 I' s dark.
25 114713
26
27 It's so dark.
28
29 Es ist so dunkel....
30
31 -----
32
667 chars 733 bytes 34 lines Ln: 33 Col: 1 Sel: 0 (0 bytes) in 0 ranges Dos\Windows ANSI INS
```

Actually, this is exactly how I translated the text so this is no made up example. You can see that you can just insert the text below the original one. It just has to be separated by a blank line. Also be aware that you can use any number of lines for the dialog from one to four. It just has to fit into the original area that was assigned for that block of text.

One more thing that might come handy when working with the text: The inserter also allows you to add another line between the two addresses. That is you MUST add it only under the dumped one as the dumped one is taken to calculate the number of bytes available for insertion. This would be all valid for reinsertion:



```

11 -----
12
13 114538
14 Ma! W c escape!
15 Blabla some text! Lololo!
16 114554
17
18 Maxim! We can't escape!
19
20 Wir können nicht zurück!
21 -----
22

```

Please note that even if I wanted to provide the dumped scripts of the original game, their content is still copyright by Taito (by now SquareenixTaitoMegacorpOrSomething) so it would be illegal to do so. Therefore I am writing this document so you know how to easily extract them yourself. Maybe someone wants to modify the dumper a bit more so it can dump all records automatically without any manual work to do here... would be nice to see things done here...

## 2.2 The bulkdumper

As already mentioned distributing the text of the original game would be illegal as it's copyrighted by Taito. In order to still be able to help you with all the work, I created a branch of the dumper containing a hardcoded list of all offsets, where the text dialogs are stored in the rom. Once you have started it correctly, it will cycle through the list of offsets dumping all the content into a single script textfile.

```

C:\Eingabeaufforderung
Lufia-bulkdumper v1.0 by FloBo                                     (c) 2011 Digisalt
-----
Wrong number of input-parameters.
Type
java -jar bulkdumper.jar <Rom-filename> <Start Offset> <Tablefilename> <Dumpfile
name>
to use this tool
C:\Dokumente und Einstellungen\flo\Desktop>_

```

This is how it can be executed. The start offset is actually being ignored but I was too lazy to remove it ;)

Just like all the other rewrites, the tool is written in Java and needs a Java 1.6 runtime installed.

## 2.3 Translating text into your own language

Of course, there might be some specific prerequisites for you to be able to actually translate the game into your preferred language. For example it might be a good idea to include language specific characters into the charset of the game. For me, that meant removing some of the unnecessary ones (like the hash symbol and the dollar sign) and inserting the german umlauts (äöüÄÖÜ) and the SZ (ß). You'll have to figure that out on your own. However it shouldn't be a real problem as the character set(s) are available within the rom in an uncompressed format.

Once you have done that, it makes sense to think about a dictionary for your desired language that makes sense. The most important parts are the tiny building blocks of words which make up most of your languages words. For example, in german the character sequences **es en ie ei im am es in er um** and quite some more are occurring VERY often. So it made sense to add them as sequences which will all be reduced to only one byte within a text block. The same applies to the following character sequences which are three letters long: **ung igt die ein und sie der wir des ver hab den das**. So those have also been added to the german dictionary table. Let's also note one thing here: you have to be smart in what you're doing! Just imagine you want to translate into english (I know this doesn't make any sense but just imagine for this case here). We all know the character sequence **ing** will appear quite regularly. So it might make sense to add it as a three character sequence to the dictionary... but maybe it also makes sense to NOT do so. Because mostly ing is just the end of a word like fishing, swimming and in most of the cases such a word is followed by another word which is just separated by a blank. So you can save even one more character by making **'ing '** a four character dictionary word. The same applies for the comma sign. Most of the cases a comma is being followed by a blank. So why not add **','** as a two character entry into your dictionary?! It will reduce almost every sentence using a comma by one byte! Also think about the three dots **'...'**. They're used in games alot. So why not reduce the space taken (three bytes... one byte for each dot) to two bytes?! It is easily done by making a custom character which is just two dots next to each other **'..'** taking up one byte. So instead of writing something..., you should now write something.... (with four dots) and they'll be only taking up two bytes in compressed form.

I think you got the point what to think about creating your first dictionary. As a side note, you may want to refine it completely having translated some of the first towns of the game. I did that having translated all dialogs until you leave Sheran for Treck. I experienced I was regularly missing the character sequence **war** which means 'was' in german. As it wasn't contained in my original DTE, it wasn't compressed every time it occurred within the translated script. So there was a huge potential to improve the compression ratio by taking out some of the other three-letter dictionary words which was less often used and replace it by the sequence war. Aside of this char sequence, there were others which I also noticed. So I searched my script for the number of times each of the dictionary character sequences was used within them and noted that number down. I did the same for the sequences that were not in the dictionary but about which I had the feeling they were more regularly used than I assumed initially. I got something like this:

Char Sequence	Occurances	In Dictionary
Und	50	Yes
Von	12	Yes
War	42	No

The example above clearly shows it would make more sense to include the word war instead of the word von into the dictionary. So this is what I did back then. Not only for the two and three character sequences but for every character sequence length available. In the end, this optimized compression ratio alot.

One more thing to note: whenever you change anything about the translated dictionary, you'll have to do so in your external dictionary tablefile AND within the translated rom. The dictionary inside the rom is split into multiple parts. Their offsets are the following:

- Item names are located between x55C00 and x57A70
- Short character sequences compressed to ONE byte are here x54D42 - x54E17
- Long character sequences compressed to TWO bytes are here x55019 - x555CC

If I forgot to mention some here, you'll be on your own to find them. Shouldn't be too hard to do so yeah. What I have just created for you (and believe me it took me AGES to do so) is the startout file for your translated dictionary. As mentioned, when replacing a dictionary word from the original game with your own one, you'll have to do so within the lists of the rom (see addresses above) and also within your translated dictionary table. The file we're talking about is named SortedNew.tbl. At first glimpse it looks quite like any thingy table. However it has a well defined structure. That is: it

contains dictionary words sorted by their number of characters instead of the way they used to occur within the rom. This will make it easier for you to introduce new words of a given length into the dictionary. Just scroll down until you find the section headed by // X CHARACTERS // where X is the number of characters your new word should have. Now you may pick any word to replace. However, there are a few different types within each section (whereas a section is made up by all dictionary words of X characters length). Those types are the following:

1. Items, Spells, etc.: These dictionary words mostly are Items or Spells. Their hexcode ALWAYS starts with 09 or 0A. There is no automatic uppercase version included (more on this later). You'll mostly not use them within dialogs but they'll be displayed within battles and in the menu. So you'll most likely go straight through them and try to translate them all at once as they really don't have any context associated...
2. General two character words: They mostly contain longer words (usually 4 or more characters long) and ALWAYS start with 0C or 0D. In contrast to the group before, each value corresponds to the lower case / upper case version of a word. So for example x0CED stands for **best** while x0DED stands for **Best**. If you want to introduce a new dictionary word of this type, please make sure to overwrite BOTH the x0C AND the x0D version of the word. They're always located directly above each other in the SortedNew.tbl file. Within the rom, there's only the lower case version hardcoded. This is one of the major groups which will be edited in the course of translating Lufia. As there's plenty of those words (256 upper and lowercase) it only makes sense to leverage that source.
3. General one character words: They mostly contain short words (at most six characters long) and are represented by x80 – xFF. They also – just like group two always contain a corresponding upper case and lower case version of the word. For a given lower case version of a word represented by x80, the upper case version will be xC0, the next pair would be x81 and xC1 and so on and so forth until we reach the pair xBF and xFF. The corresponding pairs are always located directly above each other in the SortedNew.tbl file. Also for this group of words make sure to edit BOTH occurrences in the tbl file and the single occurrence within the rom (also in lower case here). Even though this group is even better than group 2 because it compresses to only one byte, you should be very careful before you decide to take one of these words into your translation. There are still quite some of them available but their number is much lower than that of group two. So if it also works with two byte words, give it a try.
4. Then there's some special two character words which don't have the upper-lower case thing that group 2 and 3 have. Those are for example the names the group of adventurers the player gets to control. For example x0700 is linked to the word Hero – which will be automatically replaced by the name of the Hero, the player has chosen. Another example would be x070B which is the code for Gades. Names of persons start with x07. Another example of such special words are the village names which always start with the byte x0B. x0B01 is Alekia for instance. This group is similar to group one. You'll hardly want to edit those values. And if so, you'll do it all at once as the words don't have any context associated with them.
5. Then there's some more strange one character words. They also don't contain any upper-lowercase switching and therefore are just as they are. Those should be the most frequently used words you have in your translation. Their hex values are x10 – x1F. So there's really only a few of them. Make sure not to waste them easily.
6. Finally, there's the standard character set of the game. So letters from a-z in both lowercase and uppercase version, some functional characters like x04 for the linebreak and x05 for the end of dialog marker and also some numerical as well as symbolic signs. They are represented by the hex values of x20 – x7E, x04 and x05. These would have to be changed if you'd like to introduce special characters like i did for my german translation (read Umlauts, etc.).

Alright then. Let's try to actually get some translated text into the game. It's not that hard after all. First of all, we'll have to get how to actually use the inserter.

## 2.4 The checker tool

The checker tool was rewritten in java as it's pretty much straight forward. It loads all the content of a standard table file (tbl) into ram and answers your questions... why we need it? Good question. The answer is quite easy. To find the optimum compressed version of a dialog is not always a trivial task. Take the following example:

...blabla blubblub...

Now imagine we had the following dictionary words:

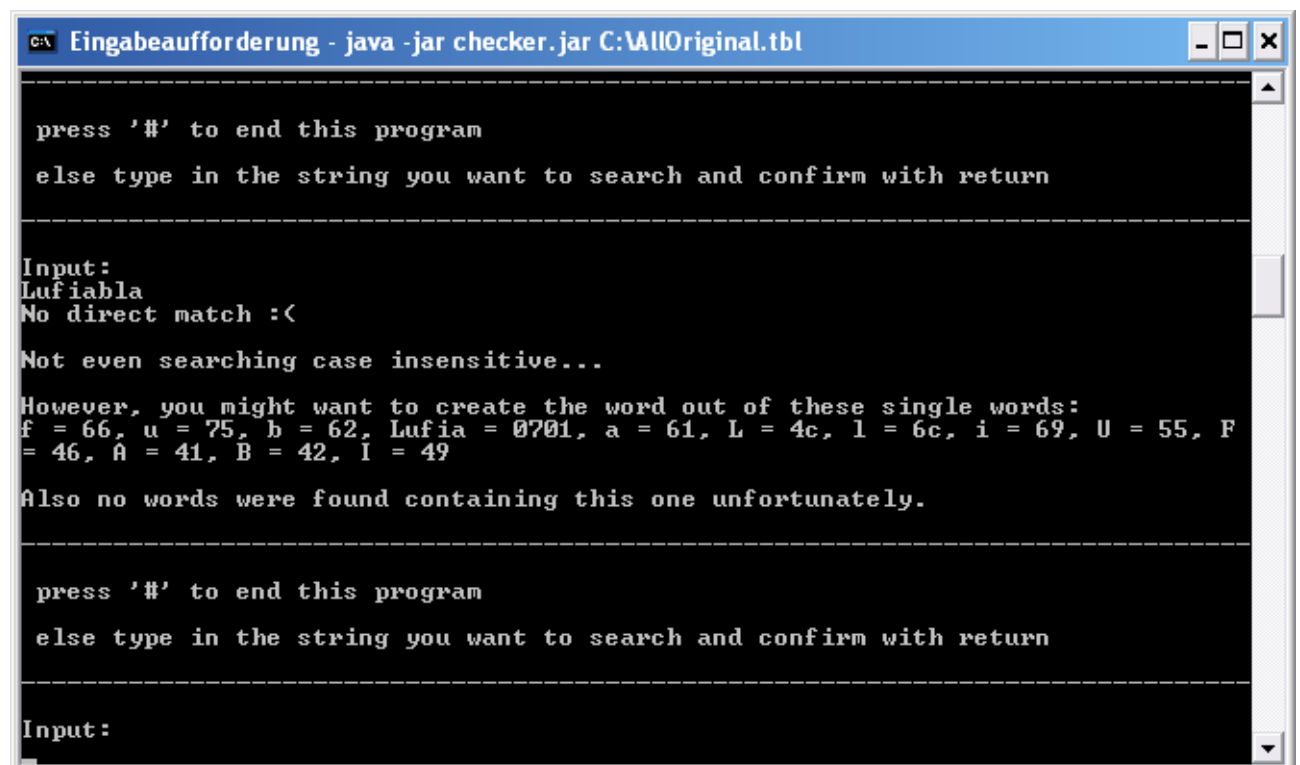
.	bla	blabla	blub	blubblub	abla blub	...
1E	1F	0C2D	0C2E	0C2F	0C30	2B

Creating the optimum compressed version out of these words is not an easy task. Some of the words can be constructed in various ways. Some of the ways exclude other options which were available in other version (you won't be able to use blabla (0C2D) in the same compressed version as abla blub (0C30)). In order to create the optimum compressed string, you might need to create quite complex graphs within the compression algorithm. Long story short: the algorithm in the inserter isn't the optimum one and will not return the best compression in every case.

So in some cases it might make sense to attempt to find the optimum compression with manual work. That's why there is the checker tool.

It was also completely rewritten in java and also enhanced with some more hints if there were some partial hits but no direct hit for what you searched for. It works like this: you start the tool, it loads the table internally, then you type in some string (can be anything).

The tools then compares the string you typed againsts what is present in the tablefile. If there's a direct match, it will tell you alongside the hexcode representing the text. If not, it will try to look for the characters contained or any smaller words which can be used to construct the word you entered and will finally display them.



```
C:\> java -jar checker.jar C:\AllOriginal.tbl

press '#' to end this program
else type in the string you want to search and confirm with return

-----

Input:
Lufiabla
No direct match :(<
Not even searching case insensitive...
However, you might want to create the word out of these single words:
f = 66, u = 75, b = 62, Lufia = 0701, a = 61, L = 4c, l = 6c, i = 69, U = 55, F
= 46, A = 41, B = 42, I = 49
Also no words were found containing this one unfortunately.

-----

press '#' to end this program
else type in the string you want to search and confirm with return

-----

Input:
```

This is an example output, the program will produce when searching for Lufiabila in the original table of the game:  
It tells us that the word itself was not found. However, Lufia is 0701 and bla could be constructed out of 62, 6C and 61. So here it's quite obvious the optimum compressed bytes will be 0701626C61.

## ***2.5 Using the inserter***

### **TO BE DONE**

Up to now, the inserter is only available as is (no source just a precompiled windows exe)... as I intended to rewrite the tool in java, this chapter has to be written at a later point in time.