Tebas Script

Documentation for Tebas Script, by Dumbelfo.
https://github.com/Dumbelfo08/Tebas

Scripts

Scripts are files, usually with the *.tbscr* extension.

Each file is divided by lines and each line is a **sentence**.

There are some special sentences:

Labels

This sentences are special, used for the control of the **flow** of the program. They must be at the start of the line and start with ':', such as:

```
:start
```

Here, it is named start.

Comments

Comments are started with '//', and they can be placed is any place of the sentence, except inside quotes or brackets.
For example:

```
//This is a valid comment
command arg1 arg2 //This is a valid comment
command "//This comment will be absorbed into arg1" //This is a valid comment
```

Table

Before explaining normal sentences, we first have to understand the central concept of the table. When the script executes, the table is created. It is a dynamic array of strings that can contain any values. There are sentences that add content into the table(always in the last position) and sentences that use the content of the table. The table is 0 indexed, so the first element is element 0. Because its impossible to control where things are added or to know where they are added, we can access things starting from the end instead of the start. Element -1 is the last element, element -2 is the previous and so forth.

Sentences

Normal sentences are divided into command and arguments. The parts are divided by whitespace, but whitespace is allowed inside of quotes or brackets.
There are 3 types of arguments:

Literals

These can be thought of as strings, they are text. However, they can have parts replaced by the contents of the table. The following literal, for example:

```
“I am a [0]”
```

That ‘[0]’ will be replaced with the value in index 0 of the table. We can do this with any number, as seen earlier. The square brackets are mandatory for this. If we want to use the opening square bracket, we can do so with the escape char: ‘\[‘. Notice that if we want to write the escape char inside a literal, we can only do it like ‘\\’.

Literals are usually written between double quotes ‘”’.

Table indexes

Some sentences use directly the values of the table, and we need to provide the index directly. Indexes are usually numbers written between square brackets ‘[]’.

For example:

```
command [0] [3] [-1]
```

Here the three args are table indexes.

Lists

Lists are collections of literals, a list of them. They are enclosed in curly brackets ‘{}’ and inside, literals are separated by commas. The literals inside can also get their values replaced as seen earlier.

For example:

```
command {“value1”, “value2”, “value[0]”}
command {  “value”  ,  “othervalue”   }
```

Sentence list

Now, a list of all sentences will be provided along with what they do and their format. Quotes will be used for literals, square brackets for table indexes, and curly brackets for lists. Comments will be used if additional information is necessary.

cmd

Syntax: cmd "command to be executed"
Explanation: Executes a command in the command line. The command should be enclosed in quotes to treat it as a literal.
Example:

```
cmd "echo Hello, World!"
```

ask

Syntax: ask "prompt message"
Explanation: Prompts the user for input and stores it in the table.
Example:

```
ask "What is your name?"
```

print

Syntax: print "message to print"
Explanation: Prints a message to the console.
Example:

```
print "Welcome to the script!"
```

run

Syntax: run "program path" "arguments"
Explanation: Runs a specified program or script with arguments.
Example:

```
run "C:\Program Files\app.exe" "--help"
```

load

Syntax: load "value"
Explanation: Loads a special value or variable into the table. Possible values explained later
Example:

```
load "%d"
```

set.literal

Syntax: set.literal "value"
Explanation: Sets a literal value into the table.
Example:

```
set.literal "My Value"
```

set.lower

Syntax: set.lower [table index]
Explanation: Converts a string from the table to lowercase and stores it in the table.
Example:

```
set.lower [0]
```

set.upper

Syntax: set.upper [table index]
Explanation: Converts a string from the table to uppercase and stores it in the table.
Example:

```
set.upper [0]
```

set.choose

Syntax: set.choose [table index] {list of values} "value if match" "value if no match"
Explanation: Chooses a value based on whether the value at the table index is in the list.
Example:

```
set.choose [0] { "option1", "option2", "option3" } "valid
option" "invalid option"
```

set.sumup

Syntax: set.sumup [table index]
Explanation: Increments a numeric value from the table and stores it in the table.
Example:

```
set.sumup [0]
```

set.sumdown

Syntax: set.sumdown [table index]
Explanation: Decrements a numeric value from the table and stores it in the table.
Example:

```
set.sumdown [0]
```

set.replace

Syntax: set.replace [table index]
Explanation: Gets the string from the table and replaces the '[X]' values, then adds it into the table. Useful for loaded values
Example:

```
set.replace [0]
```

set.copy

Syntax: set.copy [table index]
Explanation: Copies a value from the table and stores it back in the table.
Example:

```
set.copy [0]
```

goto

Syntax: goto "label"
Explanation: Jumps to a specified label in the script.
Example:

```
goto ":start"
```

branch

Syntax: branch [table index] {list of values} "label to jump to"
Explanation: If the value of the table at the specified index matches any value in the list, the script jumps to the specified label.
Example:

```
branch [1] { "A", "B", "C" } ":start"
```

pause

Syntax: pause
Explanation: Pauses the script execution and waits for a key press.
Example:

```
pause
```

exit

Syntax: exit
Explanation: Exits the script execution.
Example:

```
exit
```

table.empty

Syntax: table.empty
Explanation: Empties the contents of the table.
Example:

```
table.empty
```

file.read

Syntax: file.read "file path"
Explanation: Reads the content of a file and stores it in the table.
Example:

```
file.read "file.txt"
```

file.create

Syntax: file.create "file path"
Explanation: Creates a new empty file at the specified path.
Example:

```
file.create "newfile.txt"
```

file.delete

Syntax: file.delete "file path"
Explanation: Deletes the specified file.
Example:

```
file.delete "oldfile.txt"
```

file.rename

Syntax: file.rename "old path" "new path"
Explanation: Renames the specified file.
Example:

```
file.rename "oldfile.txt" "newfile.txt"
```

file.write

Syntax: file.write "file path" [table index]
Explanation: Writes table content to a file.
Example:

```
file.write "file.txt" [0]
```

file.append

Syntax: file.append "file path" "data"

Explanation: Appends table content to the end of a file.

Example:

```
file.append "file.txt" [0]
```

file.exists

Syntax: file.exists "file path"

Explanation: Checks if a file exists and adds 1 or 0 to the table.

Example:

```
file.exists "file.txt"
```

folder.create

Syntax: folder.create "folder path"

Explanation: Creates a new folder at the specified path.

Example:

```
folder.create "newfolder"
```

folder.delete

Syntax: folder.delete "folder path"

Explanation: Deletes the specified folder.

Example:

```
folder.delete "oldfolder"
```

folder.rename

Syntax: folder.rename "old path" "new path"

Explanation: Renames the specified folder.

Example:

```
folder.rename "oldfolder" "newfolder"
```

folder.exists

Syntax: folder.exists "folder path"
Explanation: Checks if a folder exists and adds 1 or 0 to the table.
Example:

```
folder.exists "folder"
```

template.read

Syntax: template.read "value name"
Explanation: Reads a template value and stores it in the table.
Example:

```
template.read "numberA"
```

template.write

Syntax: template.write "template path" [table index]
Explanation: Writes a value from the table to the template.
Example:

```
template.write "numberB" [0]
```

template.append

Syntax: template.append "template path" [table index]

Explanation: Appends a value from the table to the end of the template value.
Example:

```
template.append "C:\template.txt" "appended value"
```

template.run

Syntax: template.run "script name"
Explanation: Runs another script in the template.
Example:

```
template.run "debug"
```

project.read

Syntax: project.read "value name"
Explanation: Reads a project value and stores it in the table.
Example:

```
project.read "creationDate"
```

project.write

Syntax: project.write "project path" [table index]
Explanation: Writes a value from the table to the project.
Example:

```
project.write "name" [0]
```

project.append

Syntax: project.append "project path" "value"
Explanation: Appends a value from the table to the end of the project value.
Example:

```
project.append "name” [0]
```

Notes

There is a couple important things to say about the sentences.

File paths

When file path appear(this does not apply to the application path in the run sentence) they usually do inside of a literal.

You can type for example “file.txt” and it will reference the file at the working directory (usually the project directory, only the template in the installation).

You can also use the prefix “W/” to reference to the working directory, or “T/” to reference to the template directory (if available).

So you can do:

```
file.read “T/file.txt”
```

Referencing to a file in the template folder.

Labels

Wherever labels are mentioned, they come in the form of literals. You can either write it with or without the colon (it is recommended to write it with, so you know what you are referencing).

So the following lines would be equal:

```
goto start
goto :start
```

Load values

In the load sentence, it is said that it loads special values. They are short strings and they can go with the '%' symbol preceding them. Again, this is optional but it increases readability.

Full list of values:

```
load %p //Project name
load &pn //Project name

load %tn //Template name

load %w //Working directory
load %wd //Working directory

load %t //Template directory
load %td //Template directory

load %d //Current date and time

load %e //last error (don't use this)
```

Literals

You might have noticed that sometimes, literals do not come with quotes. For example in the load sentence, the literal argument is not between quotes. If the literal argument does not contain whitespace, it is an option to not write them. We only advise it when writing labels (they cannot contain spaces so it is never necessary) or when writing load values. You should use them in paths and every other literal.