

ramble.pl – Neko's Amazing Dialogue Editor, version two

Contents

ramble.pl – Neko's Amazing Dialogue Editor, version two.....	1
What it does.....	1
What it does not.....	1
Changes since version one.....	2
To run it.....	2
First steps: creating a template file from playing around in the Construction Set.....	3
Generating an importable .tescs.txt file.....	5
Generating a proper .esp file packed with nice dialogue.....	7
Structure of the template file.....	7
Elements that can appear in the template.....	7
Tags that can appear inside the “info” block.....	9
Tags that can appear inside the “dialogue” block.....	9
Using Variables.....	12
Example file: “soulgems.csv”.....	13
Example file: “SoulGems.template”.....	13
Bigger example file: “SoulGems.template”.....	14
Cool example file: “SoulGems.template”.....	15
Credits.....	15

What it does

- Happy, simple command-line program for Morrowind dialogue. It's very useful, for a very limited set of uses.
- Generate tab-delimited .txt files that TESCS can import with it's **File>Import Data>Dialogue** option
- Generate actual, factual **.esp files** for you to use, either by merging into your main mod or running it alongside your mod.
- Really make your wrists happy if you were about to copy-and-paste 131 lines of dialogue for each skill-book in the game and then fill in all the book names and IDs and what they teach by hand.
- Oh, and you'd be able to run your favourite spellcheck program on the template files. Or any other text tools you like.

What it does not

- It is not a GUI program. So it might be awkward to use. Especially if you're stuck with cmd.exe as your shell. There, I said it. I love bash and gnome-terminal. I can edit the .template files for this program with my favourite text editor - if I made a GUI to do that, it would be very limiting.
- It does not teach you how to make Morrowind Dialogue. If you're new to this, my program will only confuse you further. Play around with the proper editor and a good guide first. Come back to ramble.pl when you crave more power.

- While version two can now make .esp files (*Wheee!*), it cannot read them or edit them. It spits out a binary .esp lump that you can merge into your mod later, or use as a 'patch' plugin to be loaded after your main mod.
- It does not attempt to keep the dialogue IDs (those enormous numbers) consistent each time you generate dialogue. The new entries you create will be given a random ID, and it will be random each time you run it. Unless of course you want to fill in all those IDs by hand. This has a couple of implications if you're trying to merge in dialogue that you've already generated previously. Hence, not for the faint of heart.
- It does not write compelling, dramatic quest dialogue for you. Only version three does that.

Changes since version one

- Can now write an actual, factual **.esp file** filled with dialogue!
- Supports an **info....endinfo** block in the template to describe the generated .esp file (author, description, master file)
- Aliases for dialogue tags that are convenient for writing Journal entries – **quest name**, **quest finish**, **quest restart**, and **journal_index**
- A **set** element to easily set some global variables without resorting to making a .csv file specially for them
- A **choice....endchoice** block inside dialogue to make my life easier when writing heavily branching dialogue choices & responses.
- An **include** directive to use if you're really ambitious and want to spread your dialogue out over multiple files
- Better documentation! slightly

To run it

First thing: it's a command-line program. I don't have time or mana to make a complex GUI for this, my favourite text editor (NEdit) and a terminal window do just fine. Open a terminal (*windows XP users: you want "Command Prompt" on the start menu, or choose Run and type cmd.exe*), navigate to the directory you're using to keep ramble.pl (or .exe) and type

```
./ramble.pl --help      (bash)
ramble.exe --help      (dos)
```

...unless of course you find ramble.pl so helpful that you want to put it in your PATH for easy access =)

If all is working, this should spit out the license and usage information:

```
james@yin: ~/source/esp_generation/dialogue_generation
$ ./ramble.pl
Neko's Amazing Dialogue Generator, version two
```

(C) 2006 James Clark - This program comes with ABSOLUTELY NO WARRANTY;
it is distributed under the terms of the GNU General Public License, version 2.
This is free software, and you are welcome to redistribute it under certain
conditions - for details see the file "COPYING" that came with this program.

```
Usage: ./ramble.pl --generate <.template file> <.esp plugin file>
Usage: ./ramble.pl --generate-esp <.template file> <.esp plugin file>
Usage: ./ramble.pl --generate-tescs <.template file> <importable .tescs.txt file>
Usage: ./ramble.pl --ungenerate <exported .tescs.txt file> <.template file>
--verbose : outputs a little more info during the process.
```

If something goes wrong:

- Using the .pl version?
 - Do you have perl installed on your system? (or ActivePerl for windows)
 - Do you have the extra libraries it needs? You may need to check CPAN, your package manager, or the ActivePerl package thingamagig
 - DBI - should come as standard
 - DBD::CSV - debian package is *libdbd-csv-perl*
 - Text::CSV_XS - *libtext-csv-perl*
 - SQL::Statement - *libsql-statement-perl*
 - Is the file marked as executable? `chmod +x ./ramble.pl` to make it so.
 - Is perl installed in some weird place on your system? Maybe Mac OS X puts it somewhere strange, or something, I don't know. Just run `perl ramble.pl` instead of `ramble.pl` by itself, that should fix it.
- Using the win32 .exe version?
 - I made it using PAR, it's a self-extracting magical executable, so it takes a little while to start the first time. It seems to be faster on the second run. If it breaks, I really can't help much.
 - Although I promise it was a completely harmless executable when I packaged it, I can't promise it won't do bad things to your system, and I can't promise that some asshole hasn't infected it with a virus and repackaged it - only download from sites you trust!
 - Plus, *Windows Does Not Like Me*, it could break for other reasons I don't know about.

First steps: creating a template file from playing around in the Construction Set

To get started, you need a template file to feed into my program. But how to get one? You could read through all this documentation and learn how, but it's quicker to get you started by just basing it off dialogue you set up in the Construction Set.

- Load TESCS, select whatever ESMs you normally use, and set up some trivial little dialogue. Maybe some Greeting text. I don't know. You need to have some experience with making Morrowind dialogue the normal way. If you're new at the dialogue thing, put my toy away for a rainy day - it won't help you get started, and might even confuse you a bit ~_~"

- When you've made some changes, click **File > Export Data > Dialogue > New Dialogue**.
 - It asks you to save it's tab-delimited text file. I suggest you make a directory for your little dialogue project and put it in there.
 - The text file it makes is a bunch of dialogue lines, with each field separated by tab characters. You can load these files up in most spreadsheet programs (*e.g. Gnumeric*). I generally refer to them as **.tescs.txt** files.
 - Important note: you can tell the Construction Set to save this file with the name of a file that already exists. Maybe you're doing this a couple of times to get things "Just Right". The thing to remember is, if you tell it to save with the name of a file that's already there, the Construction Set doesn't overwrite the file (*as you might expect*), it appends the new dialogue onto the end. That might be useful in some circumstances, but it might trip you up if you were expecting it to replace the file.
- Now, go to your terminal window, cd to wherever you're keeping these files, and run something like this (*filenames and paths may vary - this example assumes you're running ramble.pl from the directory one level up*)
 - `../ramble.pl --ungenerate MyFirst.exported.tescs.txt MyFirst.template (bash)`
 - `..\ramble.pl --ungenerate MyFirst.exported.tescs.txt MyFirst.template (dos)`
- The `--ungenerate` option takes two filenames, an input filename and an output filename. The input is the tab-delimited text file that the Construction Set produced for you, and the output is a template file you can tweak and use in the next steps.
- So, after running the program, you should find that there's a new file in your working directory, `MyFirst.template`. Open it up with your favourite text editor - I like NEdit and Notepad++ myself. (*You can call it **MyFirst.txt** instead if it's convenient for you for doubleclicking purposes, ramble.pl doesn't care - but I call them .template files because that's what they are, and TESCS already uses .txt for it's own exported dialogue format*)
- What do you see? Should look a little like this:

```
# Neko's Amazing Dialogue Generator, version two
# test.template generated on Mon Oct 30 03:04:24 2006
# -----
info
    author "Neko"
    description
        something something
    enddescription
endinfo

# -----
section "Greeting"
topic "Greeting 5"

dialogue
    id_prev "410817711684828723"
    id_this "3222913824823928800"
    id_next "1452031409191968389"

    npc_id pemenie
```

```

condition Journal MV_TraderAbandoned < 10.00

text
    Hey there! Yes, you. Interested in a making a deal?
endtext
result
endresult
enddialogue

dialogue
    id_prev "3222913824823928800"
    id_this "1452031409191968389"
    id_next "261631255676123743"

    npc_id "teris raledran"

condition Journal MV_RichTrader = 80.00

text
    Shall we travel together to the shop? It'll be lonely without
    Rollie, but we'll make it!
endtext
result
endresult
enddialogue

```

I've highlighted the keywords here, and made comment lines (any line starting with a # mark) blue. If you've got a decent text editor, you can set up some syntax highlighting for yourself pretty easily, or you can bug me for my .neditrc file ;)

Generating an importable .tescs.txt file

Next step is to tweak your **MyFirst.template** file and then turn it back into Morrowind dialogue. There are two ways you can do this now - the first is to use the - - generate-tescs option to create another tab-delimited **MyFirst.importme.tescs.txt** file which you can tell the Construction Set to import.

- A simple example might look like this:

```

# Neko's Amazing Dialogue Generator, version two
# MyFirst.template
# -----
info
    author "Neko"
    description
        Silly little test.
        Adds a line of dialogue in Greeting 5
    enddescription
endinfo

# -----
section "Greeting"
topic "Greeting 5"

dialogue
    id_prev "410817711684828723"
    id_this "3222913824823928800"
    # comment id_next out or remove it entirely - ramble.pl will link for you.
    # id_next "1452031409191968389"

    npc_id pemenie

```

```

condition Journal MV_TraderAbandoned < 10.00

text
    Hey there! Yes, you. Interested in a making a deal?
endtext
result
endresult
enddialogue

# let's add this dialogue entry between the two mw-original ones
dialogue
    npc_id "fargoth"
    condition Function "Talked To PC" = 0

    text
        Greetings %PCName! I am Fargoth, ruler of this land!
    endtext
    result
        Goodbye
    endresult
enddialogue

dialogue
    # comment id_prev out or remove it entirely - ramble.pl will link for you.
    # id_prev "3222913824823928800"
    id_this "1452031409191968389"
    id_next "261631255676123743"

    npc_id "teris raledran"

    condition Journal MV_RichTrader = 80.00

    text
        Shall we travel together to the shop? It'll be lonely without
        Rollie, but we'll make it!
    endtext
    result
    endresult
enddialogue

```

- Now to run it through ramble.pl and get something we can import into TESCS. We give ramble.pl the `--generate-tescs` option, and again, we have two filenames following it, an input file and an output file. This time, our input is the template file you just tweaked, and the output is a tab-delimited file for the Construction Set to chew on. Examples might look like:
 - `ramble.pl --generate-tescs MyFirst.template MyFirst.importme.tescs.txt`
(bash, assuming ramble.pl in your path, perhaps your ~/bin directory)
 - `D:\Neko_Ramble_v2\ramble.exe --generate-tescs MyFirst.template MyFirst.importme.tescs.txt`
(dos prompt, assuming you don't have ActivePerl and are using the bundled .exe, put somewhere on your 'D' drive. If you can be bothered to set up ActivePerl, I recommend it, it's much faster than the .exe. I've never had much luck setting the windows PATH variable in xp properly, you might wanna google for it)
- my Amazing Dialogue Generator should do some muttering to itself, and if all goes well, it will give you a new file - **MyFirst.importme.tescs.txt** in this case. It's a bit less readable for humans, but exactly what TESCS can import.
- Open the Construction Set again. In this case, we probably want to set up this dialogue as a brand new mod, so load Morrowind.esm etc as usual, then use the **File > Import Data > Dialogue** menu option. This can take a long time, and there are almost certainly going to be 'errors' reported. It's Bethesda software after all, it's a little finicky.

- The Construction Set also has a habit of blanking out any fields it doesn't know about - for example, if your dialogue refers to an NPC, Journal entry, item etc that is in Tribunal, but you only have Morrowind.esm loaded, it will be blank when you import it. This is important to know if you're writing dialogue that you intend to “piggy-back” on top of your own .esp plugin.

Generating a proper .esp file packed with nice dialogue

As mentioned in the last section, there are two ways to turn your **MyFirst.template** file back into Morrowind dialogue. The second, really cool way now available in version two is to use the `--generate-esp` (or just `--generate` is fine too) option to create an actual factual **MyFirst.esp** plugin file which you can use in all sorts of ways. Exciting!

- If you have a mod already, you can load the dialogue .esp after your main mod .esp, and use it to add the extra dialogue to your main mod.
- Or you could use some tool, or the Construction Set itself to merge the dialogue into your main mod.
- Or you could just distribute two .esps, or use the dialogue .esp as a base if it's mostly dialogue and not much else is required, or... all sorts of things.
- So take your template file, and run it through `ramble.pl` and get an .esp file in return. We give `ramble.pl` the `--generate` option, and again, we have two filenames following it, an input file and an output file. Our input is the template file you just tweaked, and the output is the .esp plugin. Examples might look like:
 - `ramble.pl --generate MyProject/MyFirst.template MyProject/MyFirst.esp`
(this example assumes ramble.pl in your path, and you've made a subdirectory called MyProject/ to keep all the files related to your dialogue)
 - `ramble.exe --generate MyProject\MyFirst.template MyProject\MyFirst.esp`
(lalala same example using dos paths and the PAR self-contained exe, so boring. assumes you've got ramble.exe in the current directory, and you've made a subdirectory called MyProject to keep all the files related to your dialogue)
- my Amazing Dialogue Generator should do some muttering to itself *(by the way, you can add a `--verbose` option to the commandline to get a bit more muttering, helpful if you're trying to debug something)*, and if all goes well, it will give you a new file - **MyFirst.esp**. Lovely.

Structure of the template file

I've tried to keep the format of the .template files as clean as possible. It's plain text (*no .doc files for the love of Sheogorath...*) and line based. First word on a line is typically some kind of element or tag, and the following part of the line is the arguments or parameters for that tag. These arguments can optionally be enclosed in double-quotes, which is necessary if you want to supply an argument that contains spaces. Finally, there are a few special 'blocks', such as `dialogue....enddialogue`, `text....endtext`, and `result....endresult`, which contain other tags, or plain text. Indentation is mostly optional, but I highly recommend it to make things more readable. Any line that starts with a `#` is a comment and will be ignored by my program. Hopefully the examples included in the package will make things clearer.

Elements that can appear in the template

info endinfo	The info block is used to describe the template. The fields in here directly correspond to fields in the header of generated .esp files.
section "x"	All dialogue elements following this line will belong to dialogue section "x" unless the dialogue overrides it or you start a new section. x can be Topic, Voice, Greeting, Persuasion, or Journal.
topic "x"	All dialogue elements following this line will belong to dialogue topic "x" or journal entry "x", unless the dialogue explicitly overrides it or you start a new topic line. For example, you can set topic to "VERY special friend", "Greeting 5", "Admire Success", "Romance_Ahnassi" etc depending on the section.
dialogue enddialogue	Defines a single line of dialogue for Morrowind. Lots of different tags can be placed inside this block - see the next table to see what they all do.
loopCSV "filename" endloopCSV	<p>One of the fun perks of using Neko's Amazing Dialogue Generator. The loopCSV block can contain additional dialogue elements. It also gets a filename as an argument. What happens?</p> <p>The filename you supply should be the name of a .csv file - comma separated values. These can be constructed easily with any good spreadsheet program, and MWEdit is also good at spitting them out. The first row of the .csv file are the column headings. The remaining rows of data are processed one by one, and with each row from your .csv file, variables are set that you can use inside the loopCSV dialogues.</p> <p>The example templates should make this clearer, but essentially, in the dialogue inside the loopCSV block, you can use {ColumnHeading} anywhere you'd write normal dialogue text or conditions. This will get expanded into the appropriate field from the .csv file.</p>
set VariableName to "Something"	<p>A one-line element which takes three arguments. The first is the name of a variable you wish to define, the second is always the word 'to', and the third is some text you wish to put in the variable.</p> <p>This is basically a convenient way of setting some variables that you might want to use, without having to set up an entire .csv file for them. For example, you might write dialogue for your npc, aa_bob_the_guar, and want to have the option of changing all the dialogue you write to a different npc id later. You could do it like this:</p> <pre>Set MyNPC to "aa_bob_the_guar" dialogue npc_id {MyNPC} ...</pre>

	<pre> enddialogue dialogue npc_id {MyNPC} ... enddialogue </pre> <p>Later, if you wanted to change the ID in all those dialogue lines, you'd only have to change it in one place; the set element.</p>
<code>include "filename"</code>	If you're feeling particularly adventurous, this is the element for you. "filename" is the name of another .template file, which will be read and the contents 'included' into the list of elements ramble.pl will process. This lets you split up a particularly large project into smaller files, if you wish to.

Tags that can appear inside the "info" block

<code>author "me"</code>	Your name - appears in the "author" field of the generated .esp
<code>description</code> <code>enddescription</code>	All the lines of text between the description and enddescription lines will appear in the description field of the generated .esp file.
<code>master "esmfile"</code>	Defaults to Morrowind.esm unless you specify otherwise. Right now I only support adding one master file to the .esp header.
<code>mastersize n</code>	n is the size of the master file in bytes. Gets used by Morrowind as a "freshness check" I guess. Defaults to 79837557 because that's the size of mine... maybe other language versions have different sizes. You could always use Wrye Mash to update the final .esp file anyway.

Tags that can appear inside the "dialogue" block

<code>section "x"</code>	Explicitly sets which section this line of dialogue will go into: Topic, Voice, Greeting, Persuasion, or Journal. Before linking dialogues and writing the output file, ramble.pl will sort all the dialogue based on section and topic. Individual dialogue lines within the same (section&&topic) will stay in the same order they appear in your template.
<code>topic "x"</code>	Explicitly sets which section this line of dialogue will go into.
<code>id_prev "n"</code> <code>id_this "n"</code> <code>id_next "n"</code>	<p>As you may know, Morrowind uses large numeric IDs for it's dialogue lines. The id_ tags for a dialogue entry control how the linking between dialouges happens.</p> <p>When you use the editor, these numbers are generated for any new dialogue you create, and the prev and next numbers are updated to place each line of dialogue in it's correct order.</p> <p>When you create a .template file for my program, you are free to use these tags to specify exactly how the dialogue should be linked, or you can leave them blank.</p>

	A blank id_ this line will always be filled in by ramble.pl when you use the --generate option. It will be a new random number each time. Leaving id_prev and id_next blank will allow ramble.pl to figure out the linking itself, based on the sequence of dialogues you define in the template.
npc_rank n	Rank, according to what faction the npc belongs to. The required range for dialogue seems to be 1-10 according to the faction ranks, and -1 for “don't care” (default)
npc_faction “x”	Faction to check npc belongs to, e.g. “Census and Excise”
npc_sex “x”	“Male” or “Female”
pc_rank n	Same as npc_rank, but for the specified player faction.
pc_faction “x”	Faction to check pc for, e.g. “Ashlanders”
npc_id “fargoth”	The NPC ID this dialogue will filter to, if any. speaker can be used as an alias.
npc_race “x”	E.g. “Dark Elf”, “Khajiit”
npc_class “x”	E.g. “Apothecary Service”
cell_id	E.g. “Balmora”. cell can be used as an alias.
soundfile “x”	The mp3 played by voice greetings. E.g. "vo\b\f\Hlo_BF026.mp3" - BUT NOTE! Backslashes inside the arguments of tags inside the .template file need to be escaped, since backslash is an escape character. So you'd need to write soundfile "vo\\b\\f\\Hlo_BF026.mp3"
disposition n	Required disposition for the dialogue to activate. Journal entries use this same field for their index number - you can use journal_index as an alias for this.
specialflags “x”	Again, with Tribunal journals, you get three extra flags that can be set, “quest name”, “quest finish” and “quest restart”. In tescs.txt files, these appear as a string, “Q000”, “Q100”, “Q010” etc. The specialflags tag lets you set that string.
quest {name,finish,restart}	Alternative, more readable way of setting the Journal specialflags. quest takes one argument, which should be name , finish , or restart .
condition “cond” “name” “eq” “value”	<p>One-line tag with four arguments, and the most powerful of the dialogue filters. You can have up to six of these lines in any one dialogue block, and they directly correspond to the six dialogue conditions in the Construction Set.</p> <ul style="list-style-type: none"> • “cond” - The first parameter is the kind of condition to apply. This can be one of: Function, Global, Local, Journal, Item, Dead, Not ID, Not Faction, Not Class, Not Race, Not Cell, and Not Local. • “name” - The second parameter can vary wildly depending

	<p>on what you're looking to have in your condition. For "Function" you get things like Alarmed, Choice, PC Sex, etc. Probably the most useful of those is Function Choice. For Global, you get all the global variables at your disposal. Journal will have a journal entry ID here. Item will have an object ID here.</p> <ul style="list-style-type: none"> • "eq" - The third parameter is the comparator. Valid selections are != < <= = >= > , although I've put in a couple of aliases like == because I just know I'm going to use that by accident. • "value" - The fourth parameter is a number of some kind, relating to the condition and what you're checking. Examples will make all the confusion go away.
text endtext	<p>The most important block, which you will probably never be without. All the text you type between the <code>text</code> line and the <code>endtext</code> line is taken as literal text for your character to say (or Journal to write).</p> <p>Here's the mildly tricky thing which might not make sense to some: you can indent the text you type between the <code>text</code> line and the <code>endtext</code> line. In other words, feel free to use the [Tab] key to pretty-print the dialogue text inside the <code>text</code> block. My program will try to work out which bits of indentation to ignore and which to keep, based on how far you indent the first line after the <code>text</code> keyword.</p> <p>This may not seem that important for regular dialogue text, which isn't that sensitive about such things, but it will come in handy for the <code>result...endresult</code> block.</p>
result endresult	<p>You can use the alias <code>results...endresults</code> if you want, it's easy to do by mistake. Everything inside the <code>result</code> block gets entered into the dialogue's results script. Indentation in the template file is removed as cleverly as possible, to leave indentation in your morrowind script (if..else..endif, etc) untouched.</p> <p>As of version two, <code>result</code> blocks get a little smarter. You can actually include more than one result block in a dialogue line, and they will all be concatenated together to form the final results script for your dialogue. Why bother with this? Well, if you're a power user, you might find it very handy when combined with the next block, also new in version two: <code>choice</code>.</p> <p>You can indent the text inside the block any way you want, with spaces or tabs, but please be consistent. <code>ramble.pl</code> will get confused if you mix indentation styles.</p>
choice "x" n endchoice	<p>Choices in Morrowind dialogue gets a bit <i>spaghettified</i>. As I'm sure you're aware, dialogue expert that you are, the responses</p>

to a question asked in dialogue need to appear ~above~ the question. This is so that all the **Function Choice** conditions get processed first, and the final question gets asked if the npc hasn't asked the question yet.

I wanted a shorthand way of setting simple dialogue Choice questions and responses up, to let me write compelling, dramatic, heavily-branching plot dialogue. So, I quaffed a mana potion and wrote this. Here's how it works:

Where you'd normally put your **result** block, and add the Morrowind-script lines for the **Choice** command, you can instead place a **choice** block. The two arguments after the word choice are the question to be asked (or choice to be presented), and the number that you want to assign to the response. The fun part is, inbetween the **choice** line and the **endchoice** line, you can put **dialogue...enddialogue** elements! For example, the traditional:

```
result
  Choice "I love you" 1
  Choice "I hate you" 2
endresult
```

can become

```
choice "I love you" 1
  dialogue
    text
      Me too! Let's get married!
    endtext
  enddialogue
endchoice
choice "I hate you" 2
  dialogue
    text
      What a horrible thing to say! Let's fight!
    endtext
  enddialogue
endchoice
```

and let you have all your responses written in the body of your question dialogue. Note that those inner **dialogue...enddialogue** blocks look pretty sparse- there's no **npc_id**, no conditions specified, and there isn't even the

```
condition Function Choice = 1
```

line you would normally have to add. This is because the dialogues inside the choice block *inherit* their conditions from their parent dialogue, and **ramble.pl** adds the appropriate **Function Choice** condition automatically for you. And it also automatically puts the responses above the question for you. Which is just the way I like it ;)

See the **Choice_Test** example directory for more.

Using Variables

All this is well and good, but it's ultimately just another way to describe the dialogue. The new `choice` block is nice, certainly. Being able to split a big project into smaller files and still spit out an `.esp` at the end of it could come in handy. But you didn't come here wanting to write your first simple quest dialogue. You want power. You've used the Construction Set, and that's quite nice (*even though it drives you mad sometimes*), but there are some things you just can't do with it.

Variables are probably the biggest selling point of my silly little perl script. Anywhere you've been writing dialogue conditions like

```
npc_id "a shady smuggler"
condition Item "gold_001" >= 6000
```

and anywhere in the `text...endtext` and `result...endresult` block, you can use the special tag `{VariableName}`, where *VariableName* is whatever you want to name it. If you've defined a variable with that name previously in the file, with the

```
set VariableName to "some kind of text I don't know"
```

or by creating a `.csv` file and looping over it with the `loopCSV...endloopCSV` block, your variable's text will be substituted in place of wherever you use `{VariableName}`.

Now, I'm not sure I've explained that well. The examples I provide in different directories in this package should hopefully help with that. Here's one example:

Example file: "soulgems.csv"

SoulGemID	SoulGemName	ChoiceID
Misc_SoulGem_Grand	Grand Soul Gem	1
Misc_SoulGem_Greater	Greater Soul Gem	2
Misc_SoulGem_Common	Common Soul Gem	3
Misc_SoulGem_Lesser	Lesser Soul Gem	4
Misc_SoulGem_Petty	Petty Soul Gem	5

Example file: "SoulGems.template"

```
set MyNPC to "aa_some_guy"
section "Topic"
topic "give me a soul gem"

loopCSV "soulgems.csv"
  dialogue
    npc_id {MyNPC}
    condition "Item" "{SoulGemID}" >= 1

    text
      Did you find any soul gems for me?
      Ah! I see you have a {SoulGemName}! Give it to me!
    endtext
    result
      Choice "Give him the soul gem" {ChoiceID}
      Choice "Refuse" 99
    endresult
  enddialogue
endloopCSV
```

This is just a trivial example showing the npc, `aa_some_guy`, asking for a soulgem. There isn't any response dialogue set up to keep it simple. The set line at the top lets us use `{MyNPC}` in place of `aa_some_guy`, so if we change our minds, we can put all this dialogue in someone else's mouth instead.

The `loopCSV...endloopCSV` block generates five different lines of dialogue for us, one for each (*non-heading*) row in the CSV file. It also provides the variables `{SoulGemID}`, `{SoulGemName}` and `{ChoiceID}` for us to use inside the loop. At each iteration of the loop, those variables will go from **Grand Soul Gem** down to **Petty Soul Gem**.

For the power users, there is also a special variable set inside every `loopCSV` block, called `{LoopIteration}`. This starts at 0 and goes up with each row of the CSV file that is read.

I'll leave you with a more complete example of the soul gem dialogue, including response text. Lots more examples are included with this package, some of them fully functional, some of them just examples you could play with in the editor.

Bigger example file: "SoulGems.template"

```
set MyNPC to "aa_some_guy"
section "Topic"
topic "give me a soul gem"

# first loop - make all our "Give him the soul gem" responses.
loopCSV "soulgems.csv"
  dialogue
    npc_id {MyNPC}
    condition "Item" "{SoulGemID}" >= 1
    condition Function Choice = {ChoiceID}

    text
      Excellent, excellent! Muahahahaha!
    endtext
    result
      Player->RemoveItem, "{SoulGemID}", 1
      AddItem, "{SoulGemID}", 1
    endresult
  enddialogue
endloopCSV

# a response for all the "Refuse" choices.
dialogue
  npc_id {MyNPC}
  condition Function Choice = 99

  text
    You dare defy me!?
  endtext
  result
    StartCombat, Player
    Goodbye
  endresult
enddialogue

# second loop - make all the questions asking for a particular gem.
loopCSV "soulgems.csv"
  dialogue
    npc_id {MyNPC}
    condition "Item" "{SoulGemID}" >= 1
```

```

text
    Did you find any soul gems for me?
    Ah! I see you have a {SoulGemName}! Give it to me!
endtext
result
    Choice "Give him the soul gem" {ChoiceID}
    Choice "Refuse" 99
endresult
enddialogue
endloopCSV

```

Oh, what the hell. Here's ~Another~ version of the soulgem-crazy maniac dialogue above, but this time, using my helpful magical choice....endchoice block.

Cool example file: "SoulGems.template"

```

set MyNPC to "aa_some_guy"
section "Topic"
topic "give me a soul gem"

# a response for all the "Refuse" choices.
dialogue
    npc_id {MyNPC}
    condition Function Choice = 99

    text
        You dare defy me!?
    endtext
    result
        StartCombat, Player
        Goodbye
    endresult
enddialogue

# second loop - make all the questions asking for a particular gem.
loopCSV "soulgems.csv"
    dialogue
        npc_id {MyNPC}
        condition "Item" "{SoulGemID}" >= 1

        text
            Did you find any soul gems for me?
            Ah! I see you have a {SoulGemName}! Give it to me!
        endtext

        # first option is embedded in the special choice block.
        choice "Give him the soul gem" {ChoiceID}
            dialogue
                text
                    Excellent, excellent! Muahahahaha!
                endtext
                result
                    Player->RemoveItem, "{SoulGemID}", 1
                    AddItem, "{SoulGemID}", 1
                endresult
            enddialogue
        endchoice
        # second option is handled the traditional way,
        # since we only want one Refuse response, the one I wrote up there.
        result
            Choice "Refuse" 99
        endresult
    enddialogue
endloopCSV

```

Credits

All the locals at Emma's Elder Scrolls Forum, for giving me the inspiration to make this.

Dave Humphrey, who wrote an excellent description of the ESM/ESP/ESS file format, available at http://www.uesp.net/text.shtml?morrow/tech/mw_esm.txt

Bethesda for making Morrowind!