# Message system HowTo

The message system, like everything else, is external. The message system is an array of characters that is 100x25 in size. Each row is for one set of messages. Each message can be 21 characters / spaces in length. You can have up to 100 messages per array. You should have a basic idea on how the sprintf method works to do this.

Basics:

You have to hand code in each line of the array yourself, but it isn't that hard. There is a file that exists that has the array coded out, but it completely blank, so all you have to do is put in the words. Lets take a look at the array in general. Here is an example:

sprintf(msgs[12],"Today is a nice day. 1\0");

The above array stores the message "today is a nice day." into the 12th row. At the end of the phrase, you see a space, the number 1, and \0. The space is used as a buffer area. The number is there for a reason, which I will explain in a bit. The "\0" is a terminator, meaning that it signals the end of the string. Make sure you have this, otherwise you could end up with garbage displaying. You will end up with hundreds of warnings, because of them, but ignore the warnings.

The number after the buffer and before the string terminator (\0) is used to tell AFW what to do with the string. There are 4 numbers it uses: 1, 2, 3, and 6. 1 stands for a single string and for the fact that no other string will be displayed after it. 2 signifies that there are 2 strings in the message. 3 signifies that there is more to be displayed, so pause and wait for 2nd to be press and when it is pressed, then continue onto the next set of strings. 6 means that there are no more strings to be displayed, so stop. The number 6 is the terminator number when you use the number 2.

The message system is set up so that up to 2 sets of strings can be displayed at a time. If you want to display more strings, then the first 2 have to be erased and the new set displayed. Below is how you can display 2 and more lines of a message that some character is trying to say.

Here is an example of how to use the number 2 and 6:

sprintf(msgs[13],"You must rescue the 2\0"); - this tells afw that there is another string to display.
sprintf(msgs[14],"princess! 6\0"); - 6 tells afw that this is the end of the message

Here is an example of how to use the number 2,3, and 6:

sprintf(msgs[30],"Hey! If you steal 2\0"); - 2 strings to be displayed
sprintf(msgs[31],"anything then I will 3\0"); - there are more strings to be displayed.
sprintf(msgs[32],"send you to the 2\0"); - 2 strings to be displayed
sprintf(msgs[33],"dungeon! 6\0"); - end of strings.

The above is a good example of how to write 4 lines. This is the method you use if you want to write an even number of lines; however, if you want to write an odd number of lines, then you must use 2,3,1

in order to do it. For example:

sprintf(msgs[25],"There is a guy out 2\0"); - 2 strings to be displayed
sprintf(msgs[26],"west who can teach 3\0"); - there are more strings to be displayed.
sprintf(msgs[27],"you how to use swords 1\0"); - only 1 string to display.

If they have more than 4 lines of things to say, then use the 2,3,2,3... 2,6 pattern. If you fail to put the numbers in, then the message system will fail to display right and may not even display what you want to see. If you go over 21 characters for the array, then the message will be displayed outside of the message box (in the game) and you may not even have it displayed correctly, due to missing numbers.

The number at the end must always be in the 23rd spot. If it is not, then the string will not be parsed correctly. Even if you have to fill in the rest of the line with spaces, then you must do so to make sure the number is in the correct spot.

Your row numbers are important and you must remember them. The row numbers will be the ID numbers later on when you create the storyline files. You will use the row number to decide which message to display.

Message creation:

You will need to create a new header file. The array is placed inside of a routine. You will have to rename the routine to whatever you want it to be. Then you will have to go to the map_saver.c file and modify the save_mess() routine. It will need to be modified at the top. Where the following code exists:

```
if (i==4){update_messages4();}
else if (i==5){update_messages5();}
```

Actually, this is the code you will need to modify. Don't touch anything else, not unless you know what you are doing. You will need to modify the code into your own code. My naming conventions in the above code make things easy for me. You might want to do the same.

You will need to start with the number 3 and continue on. You cannot use the numbers 1 and 2, because they are used for the system messages. I did not include the system messages, because they do not need to be edited, because they are part of the code and are needed as is.

Background info: the save_mess() routine is called by passing a variable to it. This variable is checked by the if then statements (you can change this into a case statement if you wish). If the variable is true, then it will call the routine you have your messages in. This will fill the array then export the array to an external file.

When you create your new file, you need to modify the if, then, else statement to make sure it calls the routine you created, so your messages are loaded into the array. Keep in mind that whatever number you choose, will be the file_id number, too. Try to keep the number between 3 and 99. The file will be exported as messX, where X is the number you passed into the save_mess() routine. The number X is used to identify the file, so afw knows to pull it up and use it. That way you can access the messages

inside that array.

Then add the #include statement and include your header file in the map_saver.c file. After your code modifications, compile and run the program. You will need to make sure the messX files end up in the msgs folder (on your calculator) and make sure they are archived.

This guide should be fairly straight forward. See the storyline howto for more information on messages. It will tell you how they are displayed. This guide just tells you how to make them.

See my messages for examples.

## File list:

map_saver.c – this saves your message array to an external file
itemedit.c – the main routine
mapping_sys.h – important routines
messages4.h – my messages modify or delete them if you wish
messages5.h – my messages modify or delete them if you wish
messages-DEFAULT.h – this is the blank array that I created to make it easier for me to create message arrays. Just copy this and rename the copy. Modify the routine name and then have fun editing.