DLL debugging with a Logger

What is a logger and why should I use one?

A logger is a small (or not so small) tool to write messages from inside your program. Think printf, but usually to a file.

All capabilities printf has are also available here, so you can dump variables or just simple messages whether a certain function of branch of code has been entered or is about to be left. Writing that into a file helps you understanding and debugging your code while or after running it.

OK, and why can I just use frpintf?

fprintf needs a stream, i.e. a file that it can write to. That file must be opened (i.e. accessed) and closed (i.e. freed) after usage. If you have a program with a defined entry and exit point like main() this is a piece of cake. In case of a DLL where you don't easily know which function will be called when, not so much.

So, what should I use instead?

I use a small tool called macrologger. As the name indicates it is purely macro based with no overhead. Output is written to the stderr stream. This is one of the system streams used especially for writing error messages, stdout is used for normal output and stdin is used for – you guessed it – input.

How does it work?

Easy. Put macrologger.h in the main directory, i.e. the one that contains the DLL. Add

#include "macrologger.h"

at the beginning of the file you want to debug. Whenever you want to generate a message add something like

LOG_DEBUG("My text… and a variable to boot. Player = %d", player);

There are various debug levels like debug, info, etc. that get written in case you need different types of information or want to switch between various debug levels with varying degrees of verbosity. The output looks like this:

2020-07-11 17:57:06 | DEBUG | scars_of_mirrodin.c | card_mimic_vat:1259 | Previous iid = -1

2020-07-11 17:57:18 | DEBUG | scars_of_mirrodin.c | card_mimic_vat:1332 | EVENT_ACTIVATE IS_AI(player) = 0, instance->targets[9].card = 3701

2020-07-11 17:57:18 | DEBUG | scars_of_mirrodin.c | card_mimic_vat:1333 | EVENT_ACTIVATE can_reveal = 1, can_activate = 1

2020-07-11 17:57:22 | DEBUG | scars_of_mirrodin.c | card_mimic_vat:1259 | Previous iid = 3701

2020-07-11 17:57:22 | DEBUG | scars_of_mirrodin.c | card_mimic_vat:1299 | Imprinted id = 3397, name = Figure of Destiny

The format is

Time stamp | Debug Level | File name | Function:Line number | Your text

To generate a log file run the program (Magic.exe in our case) from the command line like this

./Magic.exe 2> error.log

to redirect stderr (the “2>” part) to the file error.log in the same directory or

./Magic.exe 2>&1

to redirect stderr to stdout, i.e. your console window. This works for both Windows cmd and bash-like shells. Windows Powershell is different, and I am no Powershell geek, but if you got that far and desperately want to use it you surely will figure it out :).

Since the log files quickly become quite long it might be easier to import them afterwards into some kind of spreadsheet program like Excel, OpenOffice Calc or LibreOffice Calc. Just open the file, the csv-import assistant will show up and specify “|” (the pipe) as separator. Then you can easily scroll, filter or delete columns you don’t need.

For more options and details have a look at macrologger.h. What I outlined here should get you up and running, though.