

DeepSee

DeepSee is a set of C++ macros + a C++ library that will help you to have a **deep see of your software**, and provides many features that will help you **debug/instrument/secure/optimize/check** it. It is a **complement to gdb/valgrind** kind of tools as it works a completely different way.

It is intrusive into your software code, but when disabled by your build system, it has **zero impact on your release soft**: for a release build, all macros are avoided and since then, you won't need the DeepSee library. A binary issued of disabled DeepSee build is strictly equal to a binary issued of the same soft without DeepSee at all. DeepSee features can be activated one per one, so you'll only compile what you need.

Some features:

- * **filtered logging** (dynamically redirect-able/formatted streams, filter logs on substrings, level, origin source code file, directory, ...)
- * memory (and class instances) **leak detections**
- * **backtrace support** (at any time, see the call stack)
- * **runtime console with evaluate interpreter** (completions, history, command edit, runtime formatted help, custom environment that can be edited/loaded/saved,...) used to allow dynamic introspection of your software.
- * **thread-able** (compilation switch to enable/disable mutex features and thread management)
- * use internally **UTF8**, auto conversion from/to unicode and 8b chars
- * plus lot of *other goodies*

The library only use libstdc++ (and pthread if in thread mode). Currently, it uses some STL, but this will be optionnal in the future to allow the user to take advantage of his own custom implementation, or fallback to a deepsee implementation for only really necessary methods.

The design was made with **optimal performance in non-debugging mode** (=> no deepsee code in the final binary) and with **maximal functionalities in debug mode** in mind.

So, depending of your compilation option (-DDEEPSEE_ENABLE + few others to allow finer grain level), you have parts of deepsee linked to your application binary, or not. If not, all deepsee macros called in your code are empty, so nothing from deepsee is really used, and static compiled binary will have no overhead compared if not using deepsee at all[[BR]] This is big point, because a 10MB static stripped binary compiled in release mode can grow easily to more than 100MB when compiled with debug flags (-g). With deepsee, you can still do software debugging (however not as gdb does) with minimal hoverhead (around 1 or 2MB using system STL).

A really nice feature is the **embedded console**, that gives you control of your deep internal code structures and objects: you can quickly provide functions that will be used by the shell to read/write/use your objects. Just write an interface to objects you want to interact with, and start the console from your application.

deepsee-Console is not a debugger (like gdb), it is really a part of the software, so use gdb for very low level debugging tasks, and use deepsee Console facilities to make high level internal reports, use your objects in scripts, for tests (detect leaks during the runtime), for optimisations (see how long a single method takes depending of its parameters or environment, dynamically without restarting the application or making batch testcases and parse long output logs after the end of execution), for data debugging (view/modify data during runtime), ...

Some cool features are planned like interfacing with gdb, with valgrind, ...

As an open source project, you can ask for features, and of course, all contributions are welcome!