



## APP TUTORIAL CREATE A GDB / VISUAL GDB INIT FILE

**By: Mr. James Calvin**

Developers can automate debugger commands normally typed on the command line by creating an Init file that is read into GDB/GDBTK (a.k.a. Visual GDB and Insight). The prerequisites are you need a GNU X-Tools toolchain installed and knowledge of what commands you want to automate with GDB / GDBTK. The Init file, also called the `'ini'` file, needs to be in a directory that the cross-executable is co-located. If the source is not located in the same directory, you will need to use a `'dir'` command as in Example 3 to show all paths to GDB / Visual GDB debug files and target executable.

A command file for GDB / Visual GDB is a file of lines that are GDB commands. Comments (lines starting with `#`) may also be included. An empty line in a command file does nothing; it does not mean to repeat the last command, as it would from the terminal.

When you start GDB / Visual GDB, it automatically executes commands from its *Init files*. These are files named ``.gdbinit'` on Unix and ``gdb.ini'` on Windows (Cygwin). During startup, the debugger does the following:

1. Reads the init file (if any) in your current working directory. (Execute `'pwd'` to find your current working directory before startup. Execute a `'cd <path>'` to change to the directory of the target executable and Init file.)
2. Processes command line options and operands in default Init file.
3. Optionally, reads command files specified by the `'-x'` option in lieu of the default Init file.

The Init file in your working directory can set options (such as `'set complaints'`) that affect subsequent processing of command line options and operands. Init files are not executed if you use the `'-nx'` option.

### Recap: Default `'ini'` File Names

Windows (Cygwin)  
`gdb.ini`

Linux  
`.gdbini`

Note: you can name the GDB/GDBTK Init (`'ini'`) file any name you want by using a `'-x'` and filename or disable the default `'ini'` file with `'-nx'`.



## Example 1. Using GDB / Visual GDB Simulator

In this example, we are assuming the target alias is 'arm-elf' (can be any GNU X-Tools toolsuite target alias that has a simulator). We start a GNU X-Tools Shell and compile a test file. Using an editor to create an Init file, we enter the simple commands presented below. We then start either GDB / Visual GDB to execute the Init file, starting the program in the simulator and stopping at the indicated function call, 'pascal\_triangle'.

Step 1. In GNU X-Tools Shell, type:

```
$ xtools arm-elf
arm-elf$ cd /home/test
arm-elf$ gcc -gstabs -o pascal.x pascal.c
```

Step 2. From a text editor, create a file named 'gdb-sim.ini' and enter the following information, and save the Ini file. Go back to the GNU X-Tools Shell (assuming you did not stop the last shell session – i.e., still in arm-elf mode and are still in the '/home/test' directory).

```
target sim
load
b pascal_triangle
run
```

Step 3. Execute the following command in the GNU X-Tools Shell:

```
arm-elf$ gdbtk -x gdb-sim.ini pascal.x
```

The GDB / Visual GDB program will start, run, and break at the 'pascal\_triangle' function.

## Example 2. A simple Init File Using an BDI2000 Debug Agent across Ethernet

```
target remote bdi:2001
load
b main
run
```

Note: To use a TCP connection, use an argument of the form 'host:port'. For example, the 'bdi' is the TCP host address (xxx.xxx.xxx.xxx) and the '2001' is the TCP port. Once this command is issued, you can use all the usual commands to examine and change data and to step and continue the remote program. In addition to this simple Init file, the developer is required to have knowledge on how the Abatron BDI2000 works with GDB. Consult Abatron user documentation.



### Example 3. A complex Init File Using an Abatron BDI2000 with GDB / Visual GDB

Create a file and name it 'gdb-bdi.ini' – this name is not the default, so you will need to use the '-x' switch in starting up the GDB / Visual GDB. We strongly encourage developers to use a naming convention that allows for multiple debug configurations (i.e., 'gdb-sim.ini' for simulator debugging; 'gdb-bdi.ini' for Abatron on-chip debugging; and so on). The following Init file example shows commands that would normally be input at the command line in succession. To understand what is happening, you need to understand GDB / Visual GDB and what commands are available and how each interacts with one another. For example, this code defines the directories for search paths for executable and source; the target commands starts a session of communications with the Abatron BDI2000; the 'load' commands a download of the executable; and the 'reset' starts the board and starts the monitor boot-up in memory. The final steps in code copies the monitor application to flash.

```
dir /home/projects/bsp/sbc332
dir ../lib332
target remote bdi:2001
load
define reset
    main packet D
end
define boot
    call boot()
end
define pflash
    set *(int*)0x1f2244=
    call copy_to_flash()
end
```

Note: The 'bdi' is the TCP host connection terminal server name with an address (xxx.xxx.xxx.xxx) and the '2001' is the TCP port number.

### Summary

A lot of reference documentation exists for using GDB / GDBTK. Unfortunately, not a lot of tutorial type of documentation exists. This short App Tutorial is not a comprehensive introduction to GDB or Visual GDB. Microcross GNU X-Tools' User Guide has a complete section dedicated to documenting the GDB and Visual GDB commands. In general, any command that can be entered on the shell command line, can be placed and executed in the init file. Appendix 1 below shows the most useful commands.



## APPENDIX 1.

### Choosing Modes

You can run GDB in various alternative modes--for example, in batch mode or quiet mode.

`-nx`

`-n`

Do not execute commands found in any initialization files (normally called ``.gdbinit'`, or ``gdb.ini'` on PCs). Normally, GDB executes the commands in these files after all the command options and arguments have been processed.

`-quiet`

`-silent`

`-q`

"Quiet". Do not print the introductory and copyright messages. These messages are also suppressed in batch mode.

`-batch`

Run in batch mode. Exit with status 0 after processing all the command files specified with ``-x'` (and all commands from initialization files, if not inhibited with ``-n'`). Exit with nonzero status if an error occurs in executing the GDB commands in the command files. Batch mode may be useful for running GDB as a filter, for example to download and run a program on another computer; in order to make this more useful, the message `Program exited normally.` (which is ordinarily issued whenever a program running under GDB control terminates) is not issued when running in batch mode.

`-nw`

"No windows". If used with GDBTK, then this option tells GDBTK to only use the command-line interface like GDB. This has no effect on Microcross' standard GDB.

`-cd directory`

Run GDB using *directory* as its working directory, instead of the current directory.

`-fullname`

`-f`

GNU Emacs sets this option when it runs GDB as a subprocess. It tells GDB to output the full file name and line number in a standard, recognizable fashion each time a stack frame is displayed (which includes each time your program stops). This recognizable format looks like two ``\032'` characters, followed by the file name, line number and character position separated by colons, and a newline. The Emacs-to-GDB interface program uses the two ``\032'` characters as a signal to display the source code for the frame.

`-epoch`

The Epoch Emacs-GDB interface sets this option when it runs GDB as a subprocess. It tells GDB to modify its print routines so as to allow Epoch to display values of expressions in a separate window.

`-annotate level`

This option sets the *annotation level* inside GDB. Its effect is identical to using ``set annotate level'`. Annotation level controls how much information does GDB print



together with its prompt, values of expressions, source lines, and other types of output. Level 0 is the normal, level 1 is for use when GDB is run as a subprocess of GNU Emacs, level 2 is the maximum annotation suitable for programs that control GDB.

`-async`

Use the asynchronous event loop for the command-line interface. GDB processes all events, such as user keyboard input, via a special event loop. This allows GDB to accept and process user commands in parallel with the debugged process being run, so you don't need to wait for control to return to GDB before you type the next command. (*Note:* as of version 5.1, the target side of the asynchronous operation is not yet in place, so ``-async'` does not work fully yet.) When the standard input is connected to a terminal device, GDB uses the asynchronous event loop by default, unless disabled by the ``-noasync'` option.

`-noasync`

Disable the asynchronous event loop for the command-line interface.

`-baud bps`

`-b bps`

Set the line speed (baud rate or bits per second) of any serial interface used by GDB for remote debugging.

`-tty device`

`-t device`

Run using *device* for your program's standard input and output.

`-tui`

Activate the Terminal User Interface when starting. The Terminal User Interface manages several text windows on the terminal, showing source, assembly, registers and GDB command outputs. Do not use this option if you run GDB from Emacs.

`-interpreter interp`

Use the interpreter *interp* for interface with the controlling program or device. This option is meant to be set by programs which communicate with GDB using it as a back end. ``--interpreter=mi'` (or ``--interpreter=mi1'`) causes GDB to use the *gdb/mi interface*. The older GDB/MI interface, included in GDB version 5.0 can be selected with ``--interpreter=mi0'`.

`-write`

Open the executable and core files for both reading and writing. This is equivalent to the ``set write on'` command inside GDB.

`-statistics`

This option causes GDB to print statistics about time and memory usage after it completes each command and returns to the prompt.

`-version`

This option causes GDB to print its version number and no-warranty blurb, and exit.

## Quitting GDB

`quit [expression]`

`q`



To exit GDB, use the `quit` command (abbreviated `q`), or type an end-of-file character (usually `C-D`). If you do not supply *expression*, GDB will terminate normally; otherwise it will terminate using the result of *expression* as the error code.

An interrupt (often `C-C`) does not exit from GDB, but rather terminates the action of any GDB command that is in progress and returns to GDB command level. It is safe to type the interrupt character at any time because GDB does not allow it to take effect until a time when it is safe.

If you have been using GDB to control an attached process or device, you can release it with the `detach` command.

**For more information, see Chapters 4 and 5 of the GNU X-Tools User Guide.**