Tutorial

Intro

TRXMAKE is a tool for speeding up embedded projects builds.

Its purpose is to be able of build a project with just a very restricted commands. Some of them are introduced through the command line, other through the trxmake.json file.

It allows you to auto generate source templates when creating a new module from scratch.

TRXMAKE works with a very specific directory layout for each module / application:

```
/module--|

          |--/build

          |--/src

          |--/inc

          |--/test
```

In build it can be found:

- The trxmake.json file.
- The autogenerated Makefile.
- The auto generated loader script on no HOST apps.
- All the output of a build.

In src/ it can be found all the source files for the module.

In inc/ it can be found all the header files for the module.

In test/ it can be found an app with the same structure as the module so you can test it.

Generating a Module

In order to speed up the process of working on a new process from scratch, TRXMAKE helps you by auto generating the necessary templates for:

- *.h and *.c file templates.
- A test/ template
- An empty trxmake.json file
- In no HOST generations, a loader script for the test template

All of it laid out with a TRXMAKE directory structure. Then you just need to start having fun.

```
trxmake.py -c genmod -m testModule --platform MSP432
```

Building a Project

Once you have a project with an application using different modules, and those modules using other modules, to build it you do not need hours configuring the Makefiles, or using a platform specific editor. You just need to have a single, pretty easy to create, trxmake.json file in the /build folder of each module.

When your source is ready, and you have linked all the modules between them through the trxmake.json file, you just need to run the followint command:

```
trxmake.py -c build -f testModule/testModule_test/build/trxmake.json --
platform MSP432
```

-f indicates where is the original trxmake.json file. It will search for the needed modules, compile then, link it all together and give you back a nice *.elf file in build/bin/

The original file must be the one of an application, that means the source file must have a main function.

--platform tells TRXMAKE the platform to be used, if no platform is indicated, TRXMAKE will build for the host machine (no cross compiling).

The trxmake.json File

The trxmake.json file is the one that tells TRXMAKE what to do in the same way that a Makeifle commands Make.

Every project must have a trxmake.json file. It might be empty, but it must exists.

```
{

}
```

There are up to three fields that a trxmake.json file might have:

- **modules:** list of modules this module is using, comma separated.
- **name:** name of the module in case you one to assign one, otherwise a default one will be given.
- **defines:** list of global defines (-D), comma separated.

```
{
        "modules"         : [ "/absolute/path/to/mod1", "/absolute/path
/to/mod2" ],
        "name"                : "moduleName",
        "defines"        : [ "BLUE", "RED" ]
}
```

TRXMAKE Help

```
usage: trxmake.py [-h] [-c {genmod,build}] [-m MODULE] [-p PARENT] [-f
FILE] [--platform {HOST,MSP432}] [-M {True,False}]

trxmake 1.0 :: TRX build app.

optional arguments:
  -h, --help            show this help message and exit
  -c {genmod,build}, --command {genmod,build}
                        What trxmake must do.
  -m MODULE, --module MODULE
                        Module name.
  -p PARENT, --parent PARENT
                        Parent folder of the module.
  -f FILE, --file FILE  Build file.
  --platform {HOST,MSP432}
                        Platform we are building for.
  -M {True,False}, --Make {True,False}
                        Build (Make) the thing.
```