EMAC OE SDK New Project

The EMAC OE SDK is a complete development kit for creating C/C++ applications for EMAC products. The SDK can be used to compile code anywhere in the development system. This procedure explains the process of creating and building a new project within the SDK.

Table 1. Conventions Used

`/path/to/sdk/`	Placeholder indicating the directory where the EMAC OE SDK is located.
`EMAC-OE-arm-linux-gnueabi-SDK_`**`XX.YY`**`.r`**`ZZ`**`.tar.bz2` `EMAC-OE-arm-linux-gnueabi-SDK_`**`XX.YY`**`/`	**XX** is the major version. **YY** is the minor version. **ZZ** is the current revision. The major and minor version numbers will match the version of OE for which the SDK was created. The current version is "4.0".

New Project Procedure

This guide assumes that the EMAC OE SDK has been installed and configured. It also assumes a basic familiarity with the C programming language and that a basic text editor is available. The example code calculates the perimeter of a right triangle given the length of its longest edge and the angle between that edge and one of the other edges.

Set Up the Project

The first step in creating an EMAC OE project is creating a project directory in `/path/to/sdk/EMAC-OE-arm-linux-gnueabi-SDK_XX.YY/projects`.

```
$ mkdir /path/to/sdk/EMAC-OE-arm-linux-gnueabi-SDK_XX.YY/projects/example
```

Creating the project in this directory is necessary in order for the `global.properties` include and the path variables in `global.properties` to be correct. If it is created in any other directory, then the make targets will fail without modifications on the Makefile that go beyond the scope of this guide.

Write the C Code

This guide uses the C code from Listing 1 as an example. It is simple enough that the basic familiarity with C expected of those viewing this guide precludes the need for explanation. As a general step in application development using the EMAC OE SDK, the C files in this step should be saved in the project's top-level directory. In this case, `example.c` should be saved as `/path/to/sdk/EMAC-OE-arm-linux-gnueabi-SDK_XX.YY/projects/example/example.c`.

Listing 1. example.c

example.c

```
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define HYP 10.0
#define DEG 30.0


int main(void)
{
        float opp, adj;

        opp = HYP * sin(DEG);
        adj = HYP * cos(DEG);

        printf("Hypotenuse: \t%3.2f\n", HYP);
        printf("Angle: \t\t%3.2f\n", DEG);
        printf("Opposite Side: \t%3.2f\n", opp);
        printf("Adjacent Side: \t%3.2f\n", adj);
        printf("%3.2f + %3.2f + %3.2f = %3.2f \n", HYP, opp, adj, (HYP + opp + adj));

        exit(EXIT_SUCCESS);
}
```

Modify the Makefile

Now that the C file has been written, it is time to copy a suitable Makefile from one of the example projects. The following command will accomplish this:

```
$ cd /path/to/sdk/EMAC-OE-arm-linux-gnueabi-SDK_XX.YY/projects/example/
$ cp ../hello/Makefile ./
```

The project is almost ready to be compiled. However, there are a few lines in the Makefile which must be modified before this can happen. First, the `CFILES` and `TARGET` variables in the Makefile must be modified to suit this project:

- `CFILES=example.c` instead of `CFILES=hello.c`
- `TARGET=example` instead of `TARGET=hello`

This tells the Makefile that the source file used is `example.c`. This variable can be assigned a space-delimited list of C files. In the example there is only one file so this is not a concern.

The last change that must be made to this Makefile is that we need to modify `LIBFLAGS` to reflect the use of the math library for the trigonometric functions in the C file. Add the following line to the Makefile:

- `LIBFLAGS+=-lm`

Where in the Makefile this line is added does not matter, though it makes sense to group it with the other variable declarations.

Listing 2 shows the complete modified makefile for the `example.c` project.

Listing 2. Modified Example Makefile

Makefile

```
include ../global.properties

TARGET=example
CFILES=example.c

LIBFLAGS+=-lm

OBJS=$(CFILES:.c=.o)
DEPS=$(OBJS:.o=.d)

all: $(TARGET)

$(TARGET): $(OBJS) Makefile
        $(CC) $(VERBOSE) $(OBJS) $(OFLAGS) $(LIBFLAGS) $(SLIBS) -o $@

%.o: %.c
        $(CC) $(VERBOSE) $(CFLAGS) -o $@ -c $<

clean:
        $(RM) *.o *.gdb $(TARGET) $(DEPS)

upload: all
        $(WPUT) $(TARGET) ftp://$(LOGIN):$(PASSWORD)@$(TARGET_IP)/../../tmp/$(TARGET)


-include $(DEPS)
```

In `global.properties` there are also variables which must be modified in order for all the `make` targets to accomplish their purpose. For instructions on how to do this, please refer to the Remote Upload Set Up Guide.

Cross-Compile with the EMAC OE SDK

Now it is time to use GNU make to compile the example source code into an application that can be run on the target machine. Makefile-based development with the EMAC OE SDK is a powerful tool which enables the customer to compile code for the target EMAC product on the Linux development machine. Once a project has been set up as described above, development can begin using the GNU make targets as described below.

First, set the current directory to the one containing the Makefile, then execute the make targets as shown in Listing 3. For a description of each make target, read the bullet items below.

Listing 3. Example Make Target Invocation

```
$ cd /path/to/sdk/EMAC-OE-arm-linux-gnueabi-SDK_XX.YY/projects/example/

# Target 1
$ make clean
rm -f *.o *.gdb example example.d

# Target 2
$ make all
../../gcc-4.2.4-arm-linux-gnueabi/bin/arm-linux-gnueabi-gcc  -Wall -MMD -g -march=armv5te -mtune=arm9
../../gcc-4.2.4-arm-linux-gnueabi/bin/arm-linux-gnueabi-gcc  example.o  -lm  -o example

# Target 3
$ make upload
wput --reupload --dont-continue sentence ftp://root:emac_inc@10.0.2.41/../../tmp/example
--16:52:48-- `app_name'
    => ftp://root:xxxxx@10.0.2.41:21/../../tmp/example
Connecting to 10.0.2.41:21... connected!
Logging in as root ... Logged in!
Length: 13,774

16:52:48 (example) - `350.6K/s' [13774]

FINISHED --16:52:48--
Transfered 13,774 bytes in 1 file at 107.3K/s
```

- `make clean` removes all object, dependency, and executable files generated by previous invocations of make. It literally "cleans" the directory as shown in Listing 1, Target 1.
- `make all` resolves dependencies for the source code necessary to cross-compile the target file. The `gcc` and `g++` binaries used are specified in `global.properties` as relative paths to the `/path/to/sdk/EMAC-OE-arm-linux-gnueabi-SDK_XX.YY/projects/example/` directory. This is why the source files must be kept in their own directory within `/path/to/sdk/EMAC-OE-arm-linux-gnueabi-SDK_XX.YY/projects`. Listing 1, Target 2 demonstrates a successful `make all` invocation.
- `make upload` uses `wput` to send `example` to the remote machine. This requires the remote EMAC product have network connection whose IP is available to the development machine. This IP address must be set as the value of the `TARGET_IP` variable in `global.properties`. Listing 1, Target 3 demonstrates a successful `make upload` invocation with `TARGET_IP` set to 10.0.2.41.

Optional global.properties Modifications

In order to debug applications effectively, the `-g` debug flag must be specified when running the compiler. This is set up to occur automatically by its inclusion in the `global.properties` `CFLAGS` variable. However, this can actually be harmful to the application's performance since keeping the debug symbols in the executable bloats its size. In a typical general-purpose computer this may not be noticeable, but for embedded systems there are typically memory and disk limitations. For production or release versions of an application EMAC recommends modifying the `global.properties` `CFLAGS` variable to replace `-g` with `-g0` which tells the compiler not to include debug information in the executable.

Next Steps

Once the target binary has been compiled, the project is ready to be debugged.

See Also

- EMAC Software Development Kit
  - Install EMAC OE SDK
  - Configure EMAC OE SDK
  - Example Projects
  - New Project

- Debugging With gdbserver

» boot_process » emac_oe_gadget » time » emac_oe_development » esdk » install » configure » example » linux_start » new

---