# Programming several Binary Files into Flash with EWARM

*IDE version: EWARM v5.xx*

## Background

IAR Embedded Workbench for ARM is able to download an application into flash memory by its integrated flash loaders. The flash loader requires a simple-code (*.sim) file to work which is generated by the debugger during the build process. If the application is to be built from several binary (*.bin) files without source code (and debug information), EWARM is not able to download it directly. If so, it is normally necessary to use a third-party flash programming tool.

**However**, EWARM can be set up to link several binary files of any kind (e.g. bin, bmp, wav, etc.) in your project. It is also possible to download the linked file into flash memory within EWARM.

## Example

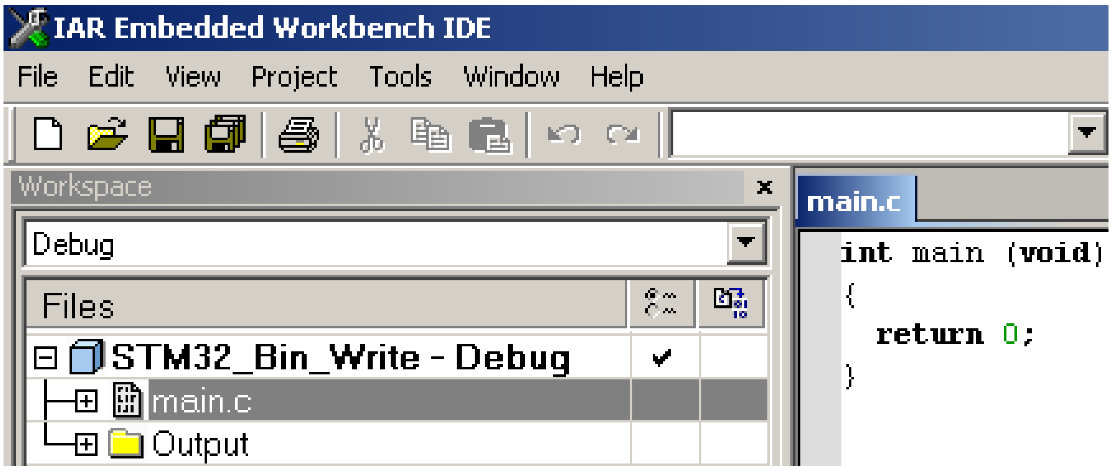
*MCU used in this example: STM32*

*Assumption: The files Prog0.bin and Prog1.bin are to be linked*

The required steps are listed below with one picture for each step, so that your can see the configurations. In case of other devices, the steps can be easily customized.

Step 1:

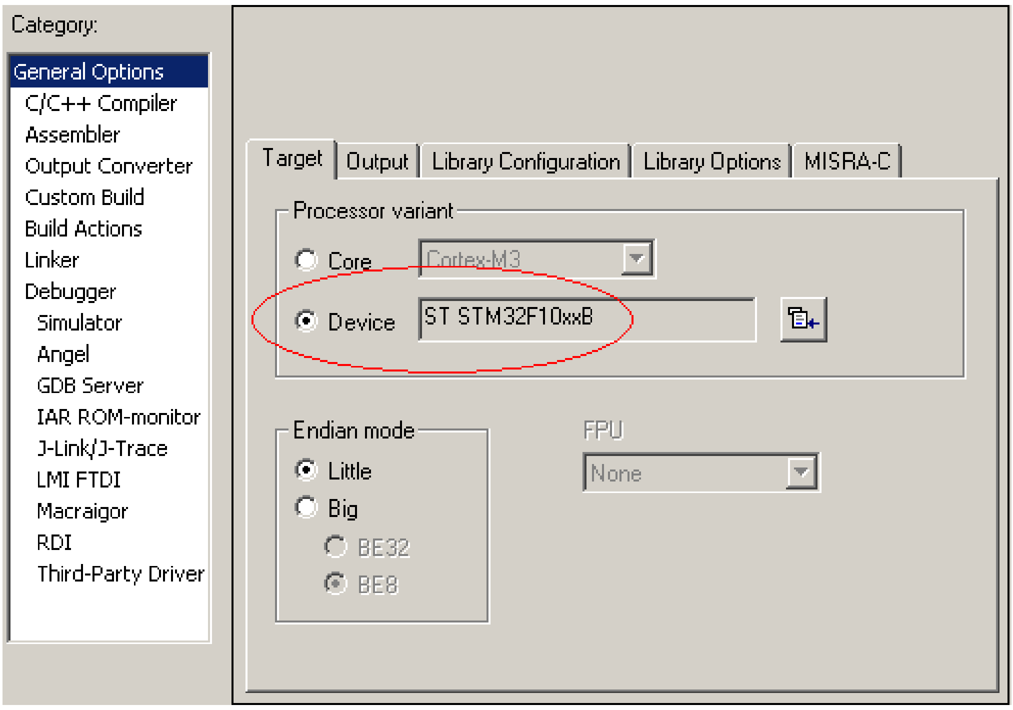
Create a new empty project in EWARM. Create a source file which should only contain an empty main() function. Add this source file (e.g. main.c) into the project, as the picture below:


IAR Embedded Workbench IDE
File Edit View Project Tools Window Help
Workspace
Debug
Files
STM32_Bin_Write - Debug
main.c
Output
main.c

```
int main (void)
{
  return 0;
}
```

Step 2:

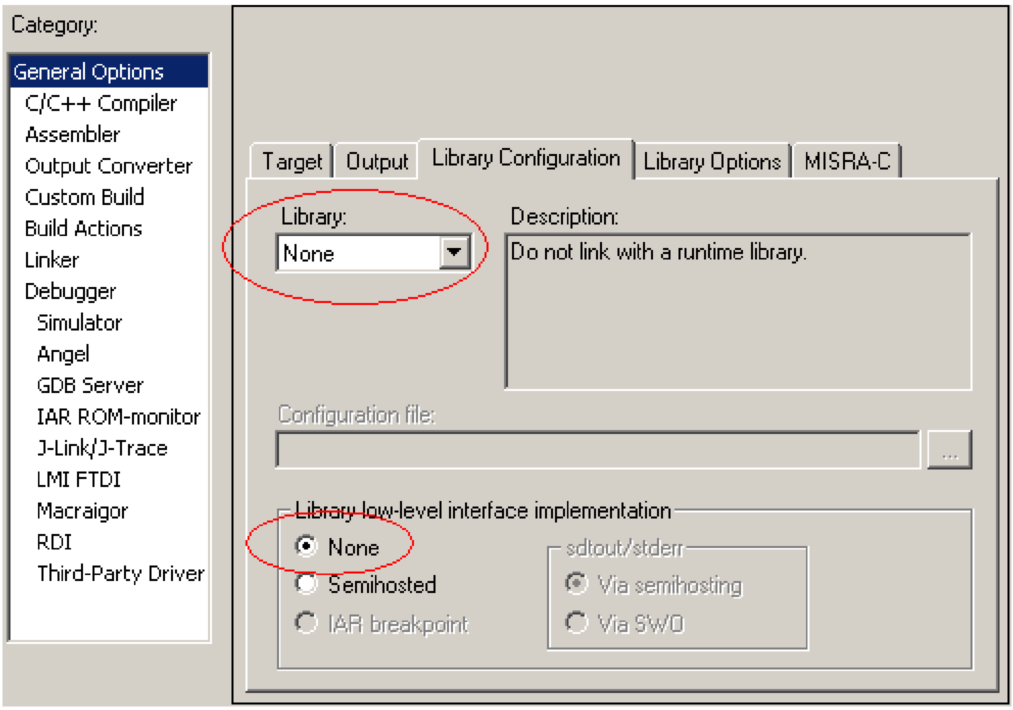
Open the Options dialog from the menu Project -> Options. In the General Options page, go to the Target tab and choose the type of MCU, e.g. STM32.



Device: ST STM32F10xxB

Step 3:

Go to the Library Configuration Tab. You do not need any runtime library here since there is no real application to be built in this project.



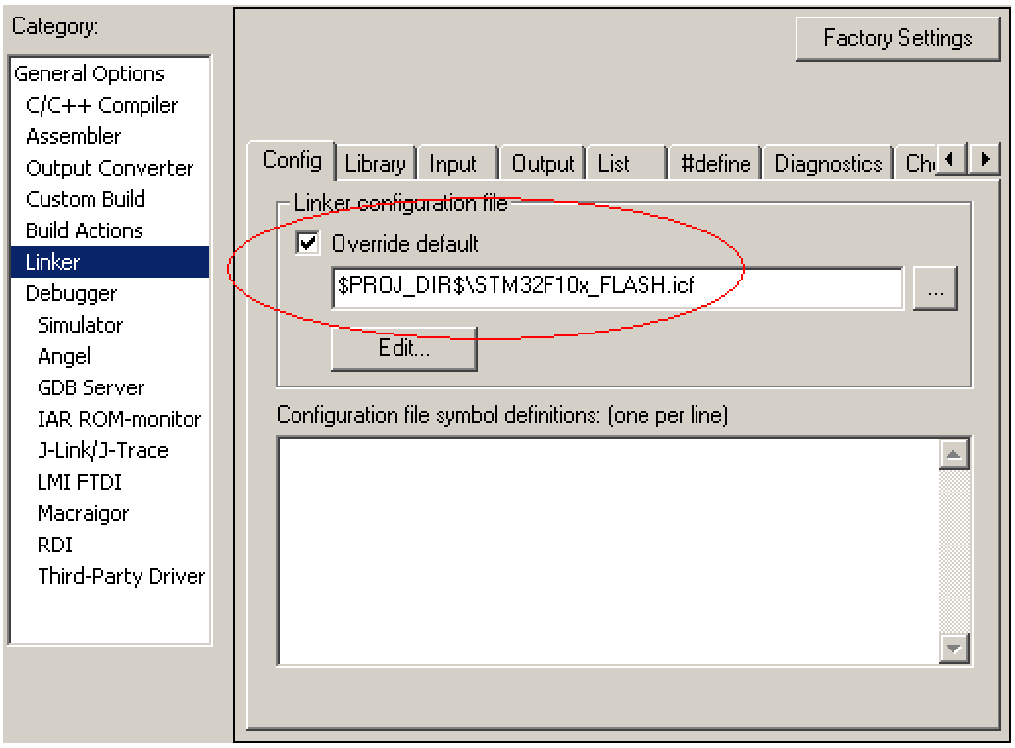
Library: None

Description: Do not link with a runtime library.

Library low-level interface implementation: [✓ None]

Step 4:

Create an ICF (ILINK Configuration File) file and add it into the Config tab of the Linker Page. ICF file is used to define the address on which the binary file will be programmed. An example of an ICF file is given at the end of this document and it should be customized according to the memory map of different hardware.



Linker configuration file

☑ Override default

$PROJ_DIR$\STM32F10x_FLASH.icf

Configuration file symbol definitions: (one per line)

Step 5:

Go to the Extra Options tab and add 2 lines for each binary file (e.g. Prog0.bin and Prog1.bin) On the first line assign a section name and a symbol name for the binary file.
On the second line force the linker to keep the symbol associated to the binary file.

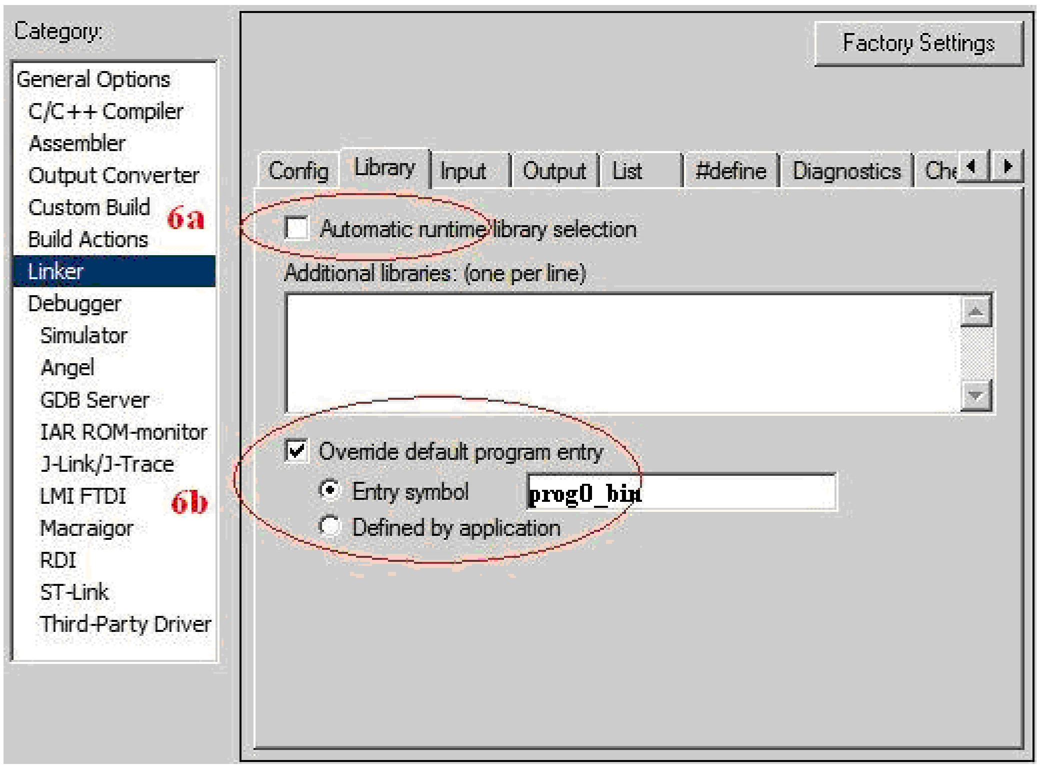
The 2 + 2 lines can look like:

```
--image_input $PROJ_DIR$\Debug\Exe\Prog0.bin,prog0_bin,prog0_section,8
--keep prog0_bin
--image_input $PROJ_DIR$\Debug\Exe\Prog1.bin,prog1_bin,prog1_section,8
--keep prog1_bin
```

Step 6: Go to the Library tab.

Step a: Disable "Automatic runtime library selection" (There should be no runtime library linked in this project.), so disable

Step b: Override the default program entry and set it to one of the symbols defined in the previous step (e.g. "prog0_bin").



☐ Automatic runtime library selection

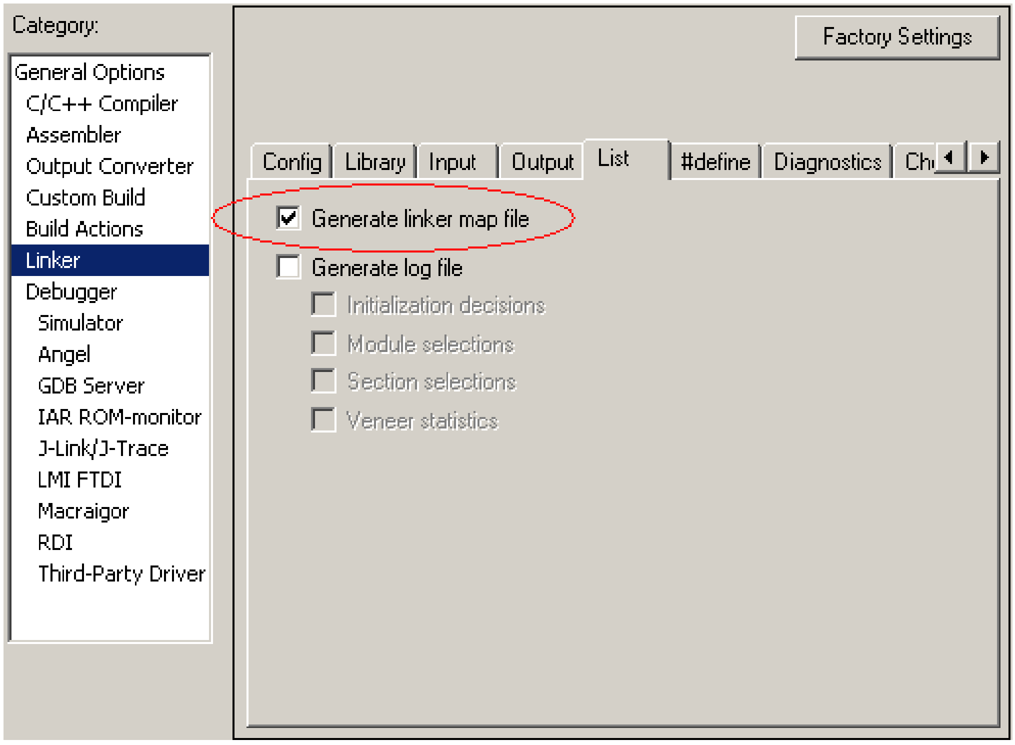
☑ Override default program entry

● Entry symbol prog0_bin

○ Defined by application

Step 7:

Go to the List tab. Generate a linker map file here in order to check whether the binary file has been located at the desired address.



☑ Generate linker map file

☐ Generate log file

☐ Initialization decisions

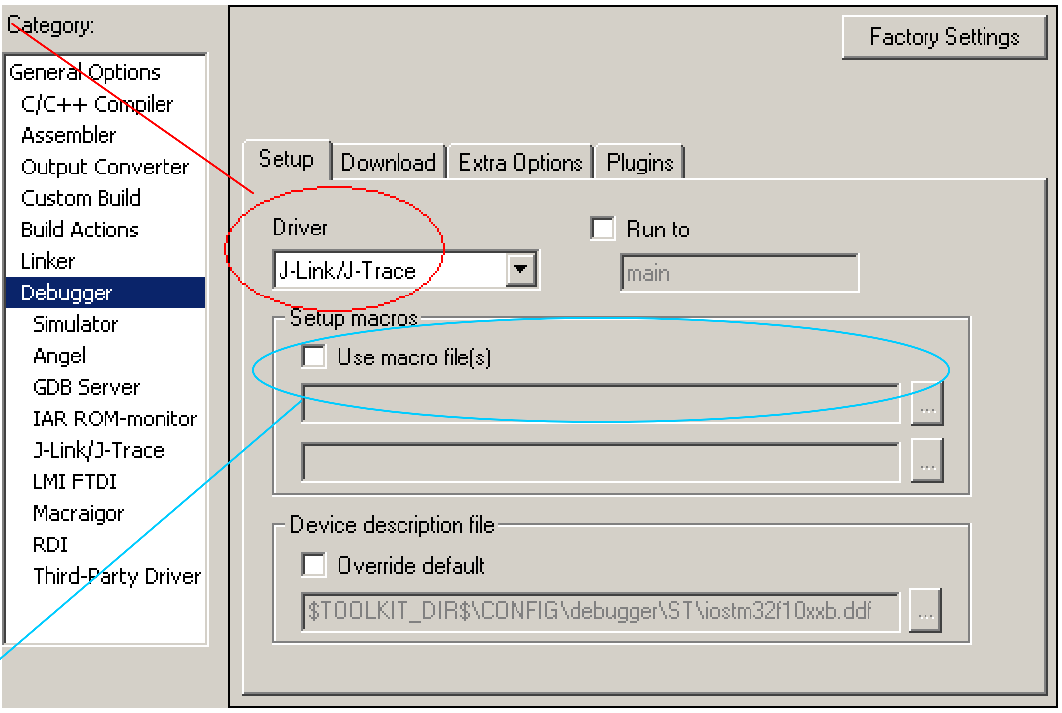
☐ Module selections

☐ Section selections

☐ Veneer statistics

Step 8:

Go to the Setup tab of the Debugger page. Select the driver of debug probe which is in use, e.g. J-Link/J-Trace. If required, configure the probe-specific options in its related options page.



Driver: J-Link/J-Trace

☐ Run to: main

Setup macros

☐ Use macro file(s)

Device description file

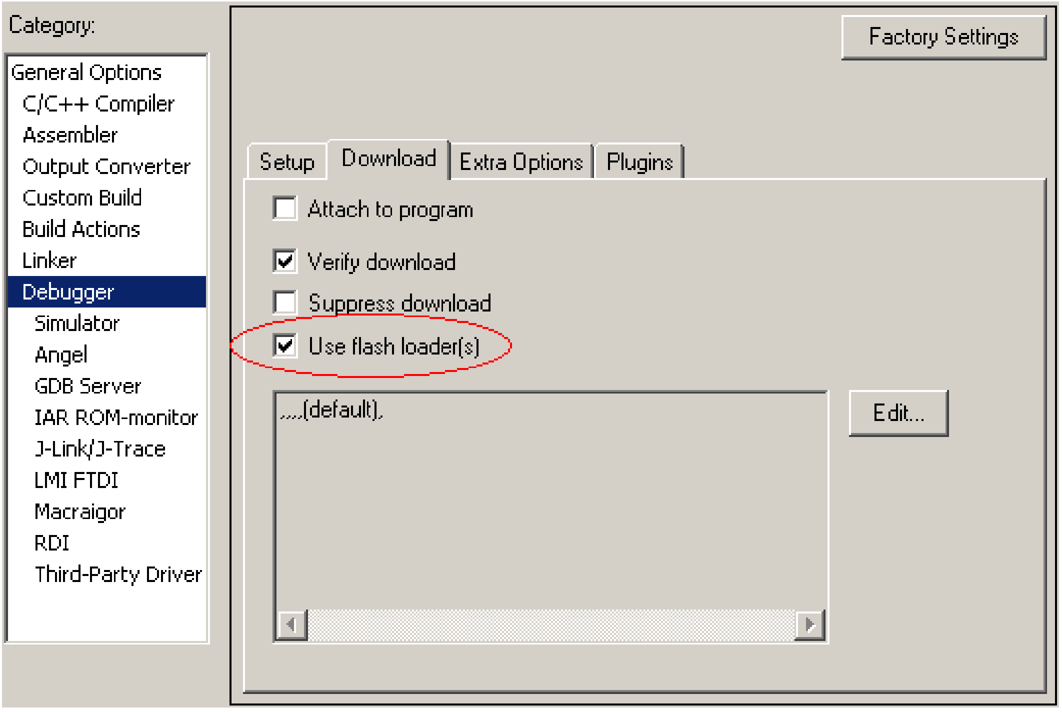
☐ Override default

$TOOLKIT_DIR$\CONFIG\debugger\ST\iostm32f10xxb.ddf

Note that some devices (for example LPC devices) need that a .mac (macro) file is supplied. To understand if such a file is needed, just compare with an example project for the device.

Step 9:

Go to the Download tab and enable the usage of flash loader. The string “default” means that a flash loader that is appropriate with the MCU type selected in step 2 will be used to program the binary file. In this example, the flash loader for STM32 internal flash memory will be used. If there is any extra option for the flash loader, click the Edit button to configure.



☐ Attach to program

☑ Verify download

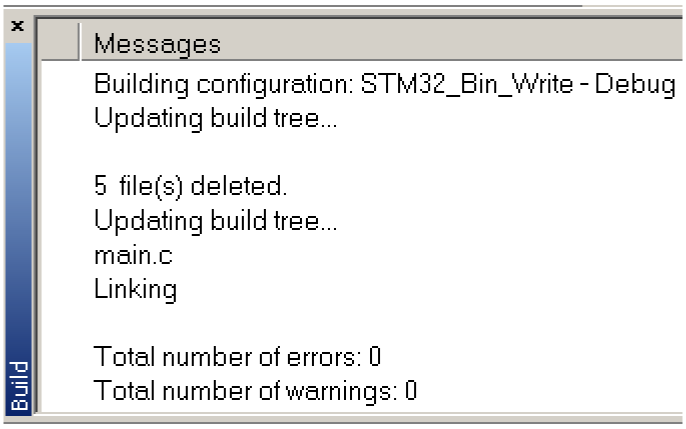
☐ Suppress download

☑ Use flash loader(s)

....(default).

Step 10:

Now the configuration is finished for all options. Close the Options dialog, save the current workspace from the menu File -> Save Workspace, and then rebuild the project from the menu Project -> Rebuild all. If all option settings are OK, there should be neither errors nor warnings:



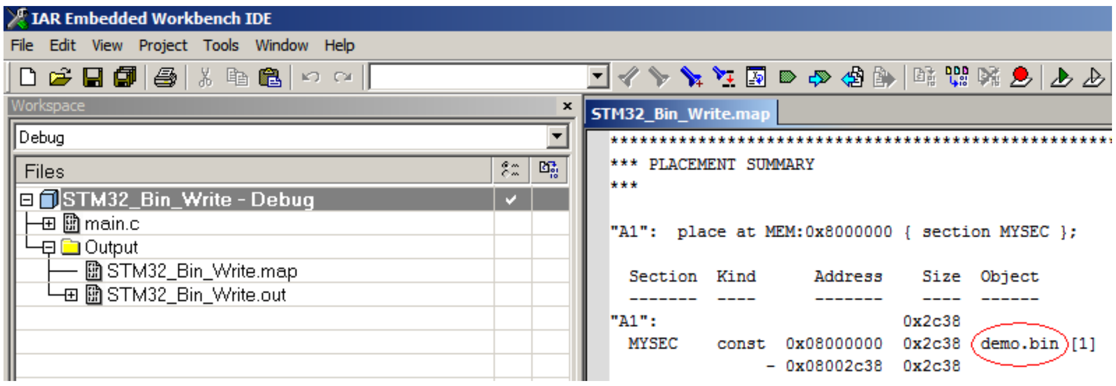
```
Building configuration: STM32_Bin_Write - Debug
Updating build tree...

5 file(s) deleted.
Updating build tree...
main.c
Linking

Total number of errors: 0
Total number of warnings: 0
```

Step 11:

Check the linker map file to verify that whether the binary file has been located at the desired address:



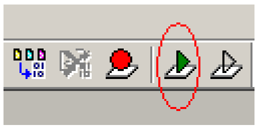
```
*****************************************************
*** PLACEMENT SUMMARY
***

"A1":  place at MEM:0x8000000 { section MYSEC };

  Section   Kind      Address    Size   Object
  -------   ----      -------    ----   ------
"A1":                           0x2c38
  MYSEC     const  0x08000000  0x2c38  demo.bin [1]
                 - 0x08002c38  0x2c38
```

Step 12:

Start the flash programming by the Download and Debug button on the toolbar!



## Appendix: content of the ICF file (STM32F10x_FLASH.icf)

```
/*************************************************************/
define symbol ROM_START      = 0x08000000;
define symbol ROM_END        = 0x0801FFFF;
define symbol RAM_START      = 0x20000000;
define symbol RAM_END        = 0x20004FFF;

define memory MEM with size = 4G;
define region ROM_region = MEM:[from ROM_START to ROM_END];
define region RAM_region = MEM:[from RAM_START to RAM_END];

place at address MEM:ROM_START { section MYSEC };
/*************************************************************/
```

The last “place” command will locate the MYSEC section (that is, content of demo.bin in this example) to the position start from ROM_START. This file can be customized according to the memory map of different hardware.