IAR-VSC Documentation

Release 1.2

Philip Luyckx

Jul 01, 2020

Contents

CHAPTER 1

User Guide

Welcome to the IAR-vsc user guide!

This guide will take you through all the features the IAR-vsc plugin provides to enhance your workflow with IAR using the power of Visual Studio Code. If you just want to get going, take a look at the Quickstart below.

1.1 Quickstart

- Configure the following items:
  - IAR: Select Workbench
  - IAR: Select Compiler
  - IAR: Select Project
  - IAR: Select Configuration
- Configure an IAR - build task using Tasks: Configure Task
- Build your project

1.2 Installation

IAR-vsc can be installed simply through the extension marketplace from Visual Studio Code.

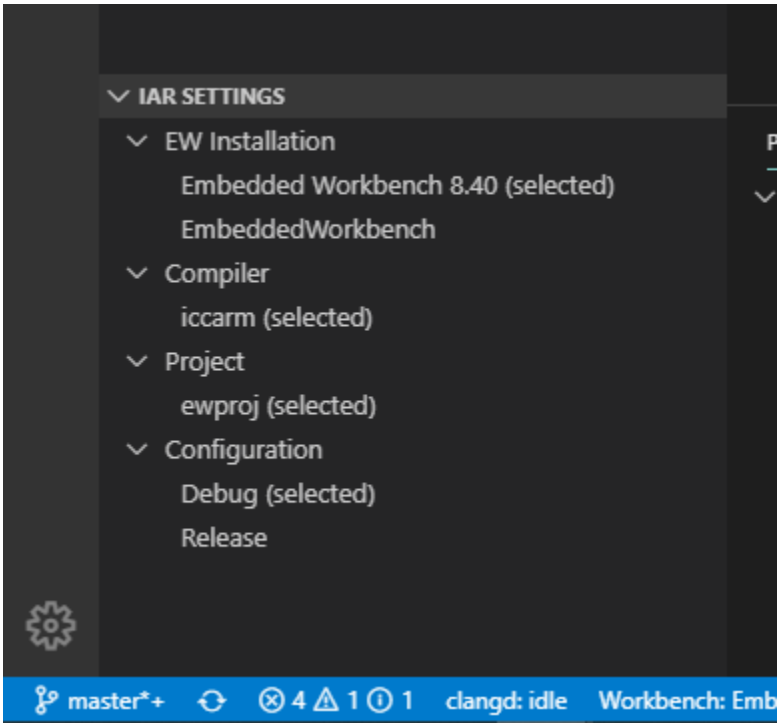
1.2.1 Requirements

This extensions presumes that you installed the cpptools plugin by *Microsoft*. Also a working IAR installation is needed.

1.3 Configuring

The extension will automatically detect Embedded Workbench installations on your systems, as well as IAR projects in your VS Code workspace. You may select which workbench to use, which configuration to build, and other settings using buttons in the statusbar.

You can also call the commands behind those buttons, see the `contribution` tab in the extension section of VS Code. There is also a view in the explorer tab for changing these settings. The explorer view and status bar view are functionally equivalent, so if you want you can hide one of them by right clicking on it.



To configure your project, configure the following options:

- IAR workbench installation
- Compiler
- Project (ewp file)
- Configuration

Note that if your Embedded Workbench is installed in a non-standard location, the plugin might not autodetect it. In that case you will need to set *iarvsc.iarInstallDirectories* in your *settings.json* file, see *Extension Settings*.

1.4 Building

When you execute the VSCode command `Tasks: Configure Task` two items are added which are generated by this extension:

- `iar: IAR Build - template using selected workbench, project and config`
- `iar: IAR Rebuild - template using selected workbench, project and config`

When selecting one of the two, a default task is generated which uses the workbench, project and configuration selected using the UI. When you select a different configuration, project or workbench, this script will use the newly selected items.

1.5 C-STAT

You can run C-STAT on your project with the `iar-cstat: Run C-STAT Analysis` task, and clear the warnings with `iar-cstat: Clear C-STAT Diagnostics`. When running these tasks, VS Code might prompt you about scanning the task output, and it is recommended to select `Never scan the task output for iar-cstat tasks`, since these tasks do not use a regular problem matcher. C-STAT will run the checks selected in your project settings in Embedded Workbench, but you can filter the messages in VS Code by setting `iarvsc.cstatFilterLevel` in your user settings.

1.6 Debugging

In v1.1.0 settings are added to configure a gdbserver and a gdb executable. The following data for the *launch.json* file will use this configuration to start debugging. Currently this is only for testing and is work in progress. The settings are nog yet automatically updated when selecting different projects or confiugrations (even though the description of the settings mention this).

Open or create the `launch.json` file and place your cursor at the beginning of the configurations array. Now press `Ctrl + Space` to activate autocompletion. You should see an item like `IAR: Debug using gdb server`. If you select this, the configuration below is automatically generated.

Some information about the used config parameters:

- `iarvsc.debugger`: The path to the debugger to use. In case your debugger is on your `PATH` environment you can just enter the debugger executable like `arm-none-eabi-gdb.exe`, otherwise, use the absolute path to the debugger.
- `iarvsc.gdbServer`: The path to the gdb server. If you are using a J-Link Segger, you will probably have to enter the full path like: `C:\GNU Tools ARM Embedded\2018-q4-major\bin\arm-none-eabi-gdb.exe`. Keep in mind you have to escape the *backslashes* `\` in *json*.
- `iarvsc.device`: The device you are want to *flash* and *debug*. Check your *debug server* documentation which values you can use here.
- `iarvsc.outFile`: This field is not yet used in the current release, but it is mentioned here for completeness. You can use it, but keep in mind the value you enter here can conflict in future releases.

```
{
    "version": "0.2.0",
    "configurations": [
        {
            "name": "Debug GDBServer",
            "type": "cppdbg",
            "request": "launch",
            "program": "Path to the out file",
            "stopAtEntry": true,
            "cwd": "${workspaceFolder}",
            "externalConsole": true,
            "MIMode": "gdb",
            "miDebuggerPath": "${config:iarvsc.debugger}",
```

(continues on next page)

(continued from previous page)

```
            "debugServerPath": "${config:iarvsc.gdbServer}",
            "debugServerArgs": "-if swd -singlerun -strict -endian little -speed auto␣
↪-port 3333 -device ${config:iarvsc.device} -vd -strict -halt",
            "serverStarted": "Connected\\ to\\ target",
            "serverLaunchTimeout": 5000,
            "filterStderr": false,
            "filterStdout": true,
            "setupCommands": [
                {
                    "text": "target remote localhost:3333"
                },
                {
                    "text": "monitor flash breakpoints = 1"
                },
                {
                    "text": "monitor flash download = 1"
                },
                {
                    "text": "monitor reset"
                },
                {
                    "text": "load \\\"Path to the out file\\\""
                },
                {
                    "text": "monitor reset"
                }
            ]
        }
    ]
}
```

1.7 Extension Settings

To change extension settings, go to `Ctrl+Shift+P->Preferences: Open Settings (JSON)` to open your `settings.json` file and add the appropriate json entries. This extension contributes the following settings:

- `iarvsc.iarInstallDirectories`: The rootfolder where all IAR workbenches are installed. By default this is `C:\Program Files (x86)\Iar Systems`. The default settings contain also the non-x86 folder in case IAR will move to 64-bit installations. For example, if your Embedded Workbench installation is at `D:\Iar Systems\Embedded Workbench 8.40`, add the following to your `settings.json` file:

```
"iarvsc.iarInstallDirectories": ["D:\\Iar Systems"],
```

This will also let the extension find any other workbench installations in that folder (e.g. `D:\Iar Systems\My Second Workbench 7.20`).

- `iarvsc.defines`: Some custom defines you can add to the define list. They follow the `identifier=value` structure. This list will contain all intrinsic compiler functions that are known by the author of this extension. If some are missing, create a GitHub issue.
- `iarvsc.c-StatFilterLevel`: Sets the lowest severity of C-STAT warnings to display.
- `iarvsc.c-StatDisplayLowSeverityWarningsAsHints`: When the filter level is set to low, this option will display low severity warnings as 'hints' instead of warnings. This is helpful if you have lots of low severity warnings and want to hide them from the problems list (but still see them in the editor).

1.7.1 Advanced usage

Using the settings it is possible to automate other IAR tasks. You can for instance automate flashing the device or running tests in the simulator using the generated cspy scripts. These scripts are available in the `settings` folder present in the same folder as you `.ewp` file.

CHAPTER 2

Contributing

This extension is developed during the spare time of the author, so every contribution is welcome. Please take a look at the issue list on github for open issues. Maybe there is an item you can help with. For discussions there is a gitter room.

2.1 Developing

The `IAR-vsc` plugin is written in *TypeScript*. It's not necessary to be an expert in this language to contribute (the author is also not an expert). Again, the gitter room is the place to be to ask questions and discuss things.

Before starting implementing things, please create first an `issue` or `feature request` on the github issue page. This way I have an overview what is going on and we can start a discussion when necessary.

2.2 Branches

There are *two* important branches when contributing:

1. `master`
2. `dev`

You should start from one of those two branches. The difference is that the `master` branch is more *stable*, but I also expect when receiving a pull request on the `master` branch that everything is done by the contributor. The following list of items is a start of requirements your pull request should satisfy:

- Feature fully tested
- Style is ok
- Documentation is up-to-date (including this sphinx documentation)

If one of these items are not satisfied, then I will try to merge on the `dev` branch. If this is not possible, I will ask to either fix the pull request to meet the above items or rebase to the `dev` branch.

On the `dev` branch, I tolerate that some of the above items are missing. It is a dev branch, so it is still under development. However, this branch is less stable than the master branch. Not only the features on this branch, but also the history. It is possible that from time to time I will cleanup the history. I will try to keep this to a minimum, but keep in mind this could happen.

There is also a third branch, `pluyckx`. This is my personal `dev` branch. I advise to not contribute on this branch. The history will change a lot, and it will contain experiments. At home I am not really using `IAR`, so I have to test things at the office. This branch is used to share changes.

2.3 Testing

In order to make sure the plugin keeps working, there is some testing to be done. To be honest, there are not a lot of tests at the moment, but I would like to change that. So I advise to write tests. There are different frameworks which help to write tests. Currently two modules are added which will help with writing tests:

- Mocha
- Sinon (ts-sinon)

The first module helps writing test *suites* and the *tests* themselves. The second helps mocking functions and modules. When writing tests, please use those two modules. If you want to use different modules or add more modules to help with some tasks, please discuss this first with the author.

2.4 Documentation

The documentation is created using Sphynx and hosted on Read The Docs. When contributing, please update the documentation. Or at least give a good start so the author can finalize it. Update an already existing page, or start a new one if necessary.

2.5 License

All source files in this repository are released under the MPL 2.0 license. If you create new files, you have to add the license header to your source files. Probably you can just reuse a header from a file in the repository of the extension.

CHAPTER 3

Release Notes

3.1 1.2.1

- Remove 'Scan for task output' dialog when running C-STAT tasks
- Automatically rescan for workspaces when changing `iarInstallDirectories`
- Change name of C-STAT settings
- Update readme section on settings

3.2 1.2.0

- Change extension name and icon
- Set license to MPL 2.0
- Pull #61: Integrate C-STAT
- Pull #60: Support IAR extended keywords
- Pull #59: Autoselect options for new projects
- Pull #59: Add explorer view for project options
- #51: Use a separate config file to store IAR project related configs which can differ between developers
- Pull #63: Add clearer error and warning messages
- Pull #48: Add extra build argument setting
- #47: Speed up startup time by only calling the compiler when selecting it and not going through all compilers at startup
- Pull #58: Allow extension to run on non-windows OSs
- Update documentation and use readthedocs as host

- Add dependency on `ms-vscode.cpptools` so it is easier to install it
- Clean up readme

3.3 1.1.0

- Implement #28: Add support to generate build tasks using the VSCode built-in command `Tasks: Configure Task`.
- Fix #18: Save all files before build. (this is actually fixed because we are now using tasks)
- Add settings to test a generic launch script to start debugging.
- Implement #10: Create a launch task to start debugging using the Segger J-Link debugger using. settings configurations parameters.
- Fix #44: Perform a recursive search to find eww files in the workspace root folder.

3.4 1.0.0

- Add __root to default defines
- Add keywords so the extension is easier to find in the marketplace
- Add listeners to the define settings so the cpptools config file is generated when changed
- Fix issue when cpptools config file is empty or invalid: plugin would not load
- Add some more settings for default c and c++ standard configuration
- Correct relative include paths in cpptools config file
- Add = sign to default defines in settings
- Add a problem matcher
- Redisgned the extension
- Instead of completely command drive, status bar items are added to configure most things
- Automatically monitor the selected ewp file and auto generate the config file
- Only generate one config file `IAR` in the `c_cpp_properties.json` file. Changing between projects/configurations will automatically trigger an update of the `IAR` config section in the json file.
- Move all settings from iar-vsc.json to the configuration file of vscode. This way configurations are reusable.
- Make extension platform aware so you can choose from all compilers from the selected workbench.
- Add new setting to define your own `defines` through the IAR settings. This way it is possible to define the intrinsics of your compiler so you do not have to wait for extension updates.
- Add command to open IAR workbench. For now only the workbench opens without a workspace. See Issue #24 @ GitHub.
- Extension is now loaded when the workspace contains `.ewp` files.
- Added basic unit- and integrationtests.
- Probably a lot more...

3.5 0.0.3

- Fixes #1: Extension did not detect end of build command
- Fixes #3: Renamed iar-vsc.js to iar-vsc.json

3.6 0.0.2

Add system include paths

3.7 0.0.1

Initial release of iar-vsc

Welcome on the `Sphinx` generated documentation website of the *Visual Studio Code* extension `IAR-vsc`. Use the menu on the left to navigate through the documentation.