Arduino IDE 1.5: Library specification

KLsz edited this page on Aug 17 · 45 revisions

This is the specification for 3rd party library format to be used with Arduino IDE 1.5.x onwards.

- rev.1 has been implemented starting with version 1.5.3 (now superseded by rev.2)
- rev.2 will be implemented starting from version 1.5.6.
- rev.2.1 will be implemented starting from version 1.6.10

This new library format is intended to be used in tandem with an automatic **Library Manager**, available since version 1.6.2. The Library Manager allows users to automatically download and install libraries needed in their projects, with an easy to use graphic interface. It doesn't yet take care of dependencies between libraries. More information about how Library Manager works are available here.

Arduino IDE 1.5.x+ supports multiple microcontroller architectures (e.g. AVR, SAM, etc), meaning that libraries may need to work on multiple architectures. The new 1.5 library format doesn't contain special support for cross-architecture libraries, but it does provide a preprocessor based mechanism for libraries to target sections of code to specific architectures.

See also

The Arduino library style guide is here : http://arduino.cc/en/Reference/APIStyleGuide

The style guide for examples is here : http://arduino.cc/en/Reference/StyleGuide

1.5 library format (rev. 2.1)

Library metadata

The most significant addition to the format is the ability to add information about the library itself through a properties file called **library.properties**.

This file will allow the future *Library Manager* to search and install a library and its dependencies in an easy and automated way.

library.properties file format

The library.properties file is a key=value properties list. Every field in this file is UTF-8 encoded. The available fields are:

- **name** - the name of the library. Library names must contain only basic letters (`A` - `Z` or `a` - `z`) and numbers (`0` - `9`), spaces ( ), underscores (`_`), dots (`.`) and dashes (`-`). It cannot start or end with a space, and also it cannot start with a number.
- **version** - version of the library. Version should be semver compliant. 1.2.0 is correct; 1.2 is accepted; r5, 003, 1.1c are invalid
- **author** - name/nickname of the authors and their email addresses (not mandatory) separated by comma ","
- **maintainer** - name and email of the maintainer
- **sentence** - a sentence explaining the purpose of the library
- **paragraph** - a longer description of the library. The value of **sentence** always will be prepended, so you should start by writing the second sentence here
- **category** - if present, one of these:
  - Display
  - Communication
  - Signal Input/Output

Clone this wiki locally

https://github.com/ardui

    - Sensors
    - Device Control
    - Timing
    - Data Storage
    - Data Processing
    - Other
- **url** - the URL of the library project, for a person to visit. Can be a github or similar page as well
- **architectures** - a comma separated list of architectures supported by the library. If the library doesn't contain architecture specific code use "*" to match all architectures
- **dot_a_linkage** - **(available from IDE 1.6.0 / arduino-builder 1.0.0-beta13)** when set to `true` , the library will be compiled using a .a (archive) file. First, all source files are compiled into .o files as normal. Then instead of including all .o files in the linker command directly, all .o files are saved into a .a file, which is then included in the linker command.
- **includes** - **(available from IDE 1.6.10)** a comma separated list of files to be added to the sketch as `#include <...>` lines. This property is used with the "Include library" command in the IDE. If the property is undefined all the headers files (.h) on the root source folder are included.

Example:

```
name=WebServer
version=1.0
author=Cristian Maglie <c.maglie@example.com>, Pippo Pluto <pippo@example.com>
maintainer=Cristian Maglie <c.maglie@example.com>
sentence=A library that makes coding a Webserver a breeze.
paragraph=Supports HTTP1.1 and you can do GET and POST.
category=Communication
url=http://example.com/
architectures=avr
includes=WebServer.h
```

Layout of folders and files

Arduino libraries will be composed of a number of folders. Each folder has a specific purpose (sources, examples, documentation, etc). Folders not covered in this specification may be added as needed to future revisions.

Source code

For 1.5.x+-only libraries, the source code resides in the **src** folder. For example:

```
Servo/src/Servo.h
Servo/src/Servo.cpp
```

The source code found in **src** folder and *all its subfolders* is compiled and linked in the user's sketch. Only the *src* folder is added to the include search path (both when compiling the sketch and the library). When the user imports a library into their sketch (from the "Tools > Import Library" menu), an #include statement will be added for all header (.h) files in the src/ directory (but not its subfolders). As a result, these header files form something of a de facto interface to your library; in general, the only header files in the root src/ folder should be those that you want to expose to the user's sketch and plan to maintain compatibility with in future versions of the library. Place internal header files in a subfolder of the src/ folder.

For backward compatibility with Arduino 1.0.x, the library author may opt to place source code into the root folder, instead of the folder called **src**. In this case the 1.0 library format is applied and the source code is searched from the **library root folder** and the **utility** folder, for example:

```
Servo/Servo.h
Servo/Servo.cpp
Servo/utility/ServoTimers.h
Servo/utility/ServoTimers.cpp
```

This will allow existing 1.0.x libraries to compile under 1.5.x+ as well and vice-versa. If a library only needs to run on 1.5.x+, we recommend placing all source code in the src/ folder. If a library requires recursive compilation of nested source folders, its code must be in the src/ folder (since 1.0.x doesn't support recursive compilation, backwards compatibility wouldn't be possible anyway).

Library Examples

Library examples must be placed in the **examples** folder. Note that the **examples** folder must be written exactly like that (with lower case letters).

```
Servo/examples/...
```

Sketches contained inside the examples folder will be shown in the Examples menu of the IDE.

Extra documentation

An **extras** folder can be used by the developer to put documentation or other items to be bundled with the library. Remember that files placed inside this folder will increase the size of the library, so putting a 20MB PDF in a library that weights a few kilobytes may not be such a good idea.

The content of the *extras* folder is totally ignored by the IDE; you are free to put anything inside such as supporting documentation, etc.

Keywords

The list of highlighted keywords must be placed in a file called *keywords.txt*. The format of this file is the same as the 1.0 libraries

```
Servo/keywords.txt
```

The keywords file is used to define what keywords the Arduino IDE should highlight when the library is included in a project file.

An example keywords file:

```
#######################################
# Syntax Coloring Map For ExampleLibrary
#######################################

#######################################
# Datatypes (KEYWORD1)
#######################################

Test    KEYWORD1

#######################################
# Methods and Functions (KEYWORD2)
#######################################

doSomething     KEYWORD2

#######################################
# Instances (KEYWORD2)
#######################################

#######################################
# Constants (LITERAL1)
#######################################
```

This keywords file would cause the Arduino IDE to highlight `Test` as a DataType, and `doSomething` as a method / function. Multiple entries can be added to each section. A tab should be used to separate each name from the `KEYWORD1` / `Keyword2` / `Literal1` identifier.

A complete example

A hypothetical library named "Servo" that adheres to the specification follows:

```
Servo/
Servo/library.properties
Servo/keywords.txt
Servo/src/
Servo/src/Servo.h
Servo/src/Servo.cpp
Servo/src/ServoTimers.h
Servo/examples/
Servo/examples/Sweep/Sweep.ino
Servo/examples/Pot/Pot.ino
Servo/extras/
Servo/extras/Servo_Connectors.pdf
```

Working with multiple architectures

In 1.5.x+, libraries placed in the user's sketchbook folder (in the libraries/ subfolder) will be made available for all boards, which may include multiple different processor architectures. To provide architecture-specific code or optimizations, library authors can use the ARDUINO_ARCH_XXX preprocessor macro (#define), where XXX is the name of the architecture (as determined by the name of the folder containing it), e.g. ARDUINO_ARCH_AVR will be defined when compiling for AVR-based boards. For example,

```
#if defined(ARDUINO_ARCH_AVR)
  // AVR-specific code
#elif defined(ARDUINO_ARCH_SAM)
  // SAM-specific code
#else
  // generic, non-platform specific code
#endif
```

Alternatively, if a library only works on certain architectures, you can provide an explicit error message (instead of allowing the compilation to fail in an difficult to understand way):

```
#if defined(ARDUINO_ARCH_AVR)
  // AVR-specific code
#elif defined(ARDUINO_ARCH_SAM)
  // SAM-specific code
#else
  #error "This library only supports boards with an AVR or SAM processor."
#endif
```

Old library format (pre-1.5)

In order to support old libraries (from Arduino 1.0.x), Arduino 1.5.x+ will also compile libraries missing a library.properties metadata file. As a result, these libraries should behave as they did in Arduino 1.0.x, although they will be available for all boards, including non-AVR ones (which wouldn't have been present in 1.0.x).