

# Slice Source Files

Slice defines a number of rules for the naming and contents of Slice source files.

On this page:

- [File Naming](#)
- [File Format](#)
- [Preprocessing](#)
- [Definition Order](#)

## File Naming

Files containing Slice definitions must end in a `.ice` file extension, for example, `Clock.ice` is a valid file name. Other file extensions are rejected by the compilers.

For case-insensitive file systems (such as DOS), the file extension may be written as uppercase or lowercase, so `Clock.ICE` is legal. For case-sensitive file systems (such as Unix), `Clock.ICE` is illegal. (The extension must be in lowercase.)

## File Format

Slice is a free-form language so you can use spaces, horizontal and vertical tab stops, form feeds, and newline characters to lay out your code in any way you wish. (White space characters are token separators). Slice does not attach semantics to the layout of a definition. You may wish to follow the style we have used for the Slice examples throughout this book.

Slice files can be ASCII text files or use the UTF-8 character encoding with a byte order marker (BOM) at the beginning of each file. However, Slice identifiers are limited to ASCII letters and digits; non-ASCII letters can appear only in comments.

## Preprocessing

Slice is preprocessed by the C++ preprocessor, so you can use the usual preprocessor directives, such as `#include` and macro definitions. However, Slice permits `#include` directives only at the beginning of a file, before any Slice definitions.

If you use `#include` directives, it is a good idea to protect them with guards to prevent double inclusion of a file:

### Slice

```
// File Clock.ice
#ifndef _CLOCK_ICE
#define _CLOCK_ICE

// #include directives here...
// Definitions here...

#endif _CLOCK_ICE
```

`#include` directives permit a Slice definition to use types defined in a different source file. The Slice compilers parse all of the code in a source file, including the code in subordinate `#include` files. However, the compilers generate code only for the top-level file(s) nominated on the command line. You must separately compile subordinate `#include` files to obtain generated code for all the files that make up your Slice definition.

Note that you should avoid `#include` with double quotes:

### Slice

```
#include "Clock.ice" // Not recommended!
```

While double quotes will work, the directory in which the preprocessor tries to locate the file can vary depending on the operating system, so the included file may not always be found where you expect it. Instead, use angle brackets (`<>`); you can control which directories are searched for the file with the [-I option](#) of the Slice compiler.

Also note that, if you include a path separator in a `#include` directive, you must use a forward slash:

**Slice**

```
#include <SliceDefs/Clock.ice>  // OK
```

You cannot use a backslash in `#include` directives:

**Slice**

```
#include <SliceDefs\Clock.ice>  // Illegal
```

## Definition Order

Slice constructs, such as modules, interfaces, or type definitions, can appear in any order you prefer. However, identifiers must be declared before they can be used.

### See Also

- [Using the Slice Compilers](#)