Lexical Rules

Slice's lexical rules are very similar to those of C++ and Java, except for some differences for identifiers.

On this page:

Comments

Slice definitions permit both the C and the C++ style of writing comments:

Slice

```
/*
 * C-style comment.
 */

// C++-style comment extending to the end of this line.
```

Keywords

Slice uses a number of keywords, which must be spelled in lowercase. For example, `class` and `dictionary` are keywords and must be spelled as shown. There are two exceptions to this lowercase rule: `Object` and `LocalObject` are keywords and must be capitalized as shown.

Identifiers

Identifiers begin with an alphabetic character followed by any number of alphabetic characters or digits. Underscores are also permitted in identifiers with the following limitations:

- an identifier cannot begin or end with an underscore
- an identifier cannot contain multiple consecutive underscores

Given these rules, the identifier `get_account_name` is legal but not `_account`, `account_`, or `get__account`.

Slice identifiers are restricted to the ASCII range of alphabetic characters and cannot contain non-English letters, such as Å. (Supporting non-ASCII identifiers would make it very difficult to map Slice to target languages that lack support for this feature.)

Case Sensitivity

Identifiers are case-insensitive but must be capitalized consistently. For example, `TimeOfDay` and `TIMEOFDAY` are considered the same identifier within a naming scope. However, Slice enforces consistent capitalization. After you have introduced an identifier, you must capitalize it consistently throughout; otherwise, the compiler will reject it as illegal. This rule exists to permit mappings of Slice to languages that ignore case in identifiers as well as to languages that treat differently capitalized identifiers as distinct.

Identifiers That Are Keywords

You can define Slice identifiers that are keywords in one or more implementation languages. For example, `switch` is a perfectly good Slice identifier but is a C++ and Java keyword. Each language mapping defines rules for dealing with such identifiers. The solution typically involves using a prefix to map away from the keyword. For example, the Slice identifier `switch` is mapped to `_cpp_switch` in C++ and `_switch` in Java.

The rules for dealing with keywords can result in hard-to-read source code. Identifiers such as `native`, `throw`, or `export` will clash with C++ or Java keywords (or both). To make life easier for yourself and others, try to avoid Slice identifiers that are implementation language keywords. Keep in mind that mappings for new languages may be added to Ice in the future. While it is not reasonable to expect you to compile a list of all keywords in all popular programming languages, you should make an attempt to avoid at least common keywords. Slice identifiers such as `self`, `import`, and `wh ile` are definitely not a good idea.

Escaped Identifiers

It is possible to use a Slice keyword as an identifier by prefixing the keyword with a backslash, for example:

Slice

```
struct dictionary {     // Error!
    // ...
};

struct \dictionary {    // OK
    // ...
};

struct \foo {           // Legal, same as "struct foo"
    // ...
};
```

he backslash escapes the usual meaning of a keyword; in the preceding example, `\dictionary` is treated as the identifier `dictionary`. The escape mechanism exists to permit keywords to be added to the Slice language over time with minimal disruption to existing specifications: if a pre-existing specification happens to use a newly-introduced keyword, that specification can be fixed by simply prepending a backslash to the new keyword. Note that, as a matter of style, you should avoid using Slice keywords as identifiers (even though the backslash escapes allow you to do this).

It is legal (though redundant) to precede an identifier that is not a keyword with a backslash — the backslash is ignored in that case.

Reserved Identifiers

Slice reserves the identifier `Ice` and all identifiers beginning with `Ice` (in any capitalization) for the Ice implementation. For example, if you try to define a type named `Icecream`, the Slice compiler will issue an error message.

You can suppress this behavior by using the `--ice` compiler option, which enables definition of identifiers beginning with `Ice`. However, do not use this option unless you are compiling the Slice definitions for the Ice run time itself.

Slice identifiers ending in any of the suffixes `Helper`, `Holder`, `Prx`, and `Ptr` are also reserved. These endings are used by the various language mappings and are reserved to prevent name clashes in the generated code.

See Also

- Slice Keywords