DRAFT: The Hierarchical Module Namespace Extension, version 0.0
An Addendum to the Haskell 98 Report

Simon Marlow, Microsoft Research, Cambridge
Malcolm Wallace, University of York

The master version of Hierarchical Module Namespace Extension is at `haskell.org`. Any corrections or changes in the report are found there.

Preface

Haskell 98 defines a module system with a *flat* module namespace. That is, module names are just identifiers, and all module names occupy the same namespace. This creates two problems for the Haskell programmer:

- There is no consistent naming scheme for modules, so it is hard to locate functionality within a library.
- Module names are likely to clash with each other. It is not possible to choose a module name for a library that is guaranteed not to clash with other library modules.

The purpose of this document is to define a modest extension to Haskell 98 that extends the module namespace and gives it a hierarchical structure.

On the face of it, this extension solves neither of the above two problems! However, it is an important first step: in having a way to arrange modules into a hierarchy, we have a *mechanism* to avoid name clashes, and a way to organise libraries into a tree by functionality. The policy by which the hierarchy itself is organised is not in the scope of this specification, but we expect it to be the subject of future specification(s).

This document is an *addendum* to the Haskell 98 definition, which means that it has undergone extensive peer-review by the Haskell community prior to publication[1].

The hierarchical module namespace extension depends on no other Haskell 98 extensions.

Acknowledgements

We would like to thank everyone on the mailing list `libraries@haskell.org` who has contributed in some way to this proposal, and also those Haskell system implementors who have added this extension to their compilers and interpreters.

[1] or at least it will have done, by the time version 1.0 is published

1 The Language Extension

The key concept is to map the module namespace into a hierarchical directory-like structure. We propose using the dot as a separator, analogous to Java's usage for namespaces.

This is a surface change to the module naming convention. It does not introduce nested definition of modules. One change is required to the lexical syntax of Haskell, namely that *modid* is redefined from:

$$modid \rightarrow conid$$

to:

$$modid \rightarrow qconid$$

for reference, the definition of *qconid* is:

$$qconid \rightarrow [modid\texttt{.}]conid$$

Note that the new syntax is recursive, a *modid* may contain multiple components separated by dots, where the final component is a *conid*.

A consequence of using the dot as the module namespace separator is that it steals one extremely rare construction from Haskell 98:

```
A.B.C.D
```

in Haskell'98 means the composition of constructor D from module C, with constructor B from module A:

```
(.)  A.B  C.D
```

With the hierarchical module namespace extension, `A.B.C.D` would instead be interpreted as the identifier `D` from module `A.B.C`. If the original Haskell 98 interpretation is intended, then it must be written with extra spaces, as `A.B . C.D`.

2 Modules and the filesystem

This section describes possible implementation techniques, and how we expect the hierarchical namespace to be exposed to the programmer by Haskell implementations. What follows is *not* part of the specification of the hierarchical module namespace extension; Haskell implementations are still free to implement any mapping between modules and filenames they choose (as in plain Haskell 98).

For most compilers and interpreters, we expect that the hierarchical module namespace will map directly to a directory/file structure in which the modules are stored. For example, a module `Foo.Bar.Baz` might be stored in the file `Foo/Bar/Baz.hs` if `/` is the directory separator.

Note that the hierarchical module namespace is far simpler than a file directory structure, in that hierarchical module names do not provide any operations such as relative paths or parent directory.

In the hierarchical module namespace, a particular module name may also be a node in the hierarchy. For example, we might have modules names both `Data.Array` and `Data.Array.IO`. It is therefore important that directories are distinguished form Haskell source modules, for example by using the `.hs` suffix: `Data.Array` would be stored in `Data/Array.hs` and `Data.Array.IO` in `Data/Array/IO.hs`.