Notes - 4/20

Friday, April 20, 2012 10:18 AM

70

[In 41/4725 at 10:15 today.]

Agenda:

- Review Monaco feedback on modules (attached)
- Class expressions / nested classes (attached)
- Continue review of design items spreadsheet
- Others?

Decisions:

a. Only emit static dependency if module referenced as a value
b. Import syntax below
c. Only dotted identifiers in type position

Questioning decision on merging modules

Separating namespaces vs. modules?

Thinking on 'object':

```
object Foo.Bar.Baz {
        Public x = 1;
        Public f(…) {…}
        class X {…}
        Interface Z  // PROBLEM: default visibiltiy?  Steve wants private (which means internal).  Anders wants true encapsulation.



}
```

```
// Winner!
import a = module("./a");
import b = X.Y.Z
import c = 'vs/ide/code'.X.Y
import X.Y.Z
import module("./a");
```

In type [po

Thanks for the notes Dirk. Adding the Strada design team as well. We'll discuss this in our Strada design meeting today in advance of meeting with you on Monday.

Thanks,
Luke

From: Dirk Baeumer
Sent: Friday, April 20, 2012 7:48 AM
To: Luke Hoban; Joe Pamer
Cc: Monaco Tools Team
Subject: Feedback on the Module Loading proposal

Hi Luke, Hi Joe

the Zurich team looked over the proposal you sent me and it is very close to what we do right now. So converting our code to it should be easy and straight forward and can very likely be automated. But we have some concerns with the new proposal as well:

Module names in the generated JS code:

In the included example the JS code for the module foo looks like this:

```
    // foo.js

    define("foo", [], function(exports) {

        exports.bar = function() { return 3; }
})
```

AMD strongly recommends to use anonymous modules to allow consumers to name it. In addition anonymous modules are required to use the “r” tool (see http://requirejs.org/docs/optimization.html) to optimize a system consisting out of n module for production.

The nice benefit of using anonymous modules and r is that the development structure is decouple from the production structure. When r is executed it has a reverse mapping from folder structure to module ids and traverse the modules and packages the code into one big file. To make sure the modules are correctly registered in the AMD module loader during runtime r patches in the module ids into the code. But only if the modules are unnamed.

Module names in the Strada code:

It was our assumptions so far that we will not name modules on the provider side especially not in a way that the name must match the name on the consumer side. In Strada modules should be anonymous like in AMD unless the programmer explicitly decides to name them. If he does name them the generated JS code should list a module name as well in the define call.

For example named modules will not allow to include two different version of a module into my code unless I name them differently (which is quite unlilkely for public modules). Consider a scenario like this

```
/base/myLib

        1.0

                mylib.str

        1.1

                mylib.str
```

With unnamed modules I can write something like this

```
// consumer.str

module lib1_0 from “base/mylib/1.0/mylib”;

module lib1_1 from “base/mylib/1.1/mylib”;
```

which is not possible with named modules unless I change the name of my library with every version (which then makes consuming the latest hard). JQuery for example uses this style of versioning together with a module loader.

Additionally relative module names in the from part should be supported on the consumer side. Consider a system like this

```
/base/core

    helper.str

    model.str

/base/ui

    ui.str
```

In model.str I would like to reference helper.str relatively to model.str. In ui.str I would like to reference model.str absolute. That means:

```
// model.str

module helper from “./helper”;
```
← **this is relative to model.str**

```
// ui.str

module model from “base/core/model”
```

Reference statement

The reference statement basically duplicates the information that is already present in the from part of the module statements. I have attached the strada source file of the scmViewlet as an example.

It was our understanding that the /// reference statement goes away and that we will have a first class statement to reference used code. The module statement would be a perfect candidate for this especially when combined with an import statement. And the compiler could even analyse the code and decide whether a module reference is there to satisfy compilation or is a real runtime reference. And only for real runtime references the compiler would then generate a dependency onto that module. This would remove the need of a separate reference statement as we proposed some time ago. Considering module references at compile time to load the referenced code will in addition remove the requirement of naming modules on the provider side.

However we see that this might be a bigger change on the compiler side but I think it is the right direction. To split this into steps we could do the following:

- for now the caller of the compiler is still required to pass in all strada files required at compile time.
- if the strada file is anonymous we will give it a “temporary” name when feeding it into the compiler. Since require supports path rewrites the name must the reverse transformed from the physical location (like the r tool does). Or we plug in some little tool into the compiler to reverse map a file location to a module name.
- in a next step the compiler supports pluging-in a file resolver which is passed the requested module id and will return the file content.

Additional items we would like to discuss on Monday are:

- is self really going away. We saw a checkin notice and it disappeared from the spec.
- Being able to use the lexer without the rest of the compiler. For syntax coloring we use the lexer right now which currently loads the whole compiler into the main worker.
- how to continue with the language service. For a compiler adoption we spent the most time on adopting the language service. We should come up with a plan to have one unified language service.

Dirk