Notes - 3/26

Monday, March 26, 2012 9:00 PM

66

Closed modules

From: Luke Hoban
Sent: Monday, March 26, 2012 9:57 AM
To: Luke Hoban
Subject: Modules Scenarios

Modules Scenarios

#1: Web

```
// jquery.str
interface IJQuery
declare var $: IJQuery

// game.str
///<reference path='jquery.str'/>
class Game() {}

// main.str
///<reference path='jquery.str'/>
///<reference path='game.str'/>
var game = new Game()

//index.html
<script src='jquery.js'></script>
<script src='game.js'></script>
<script src='main.js'></script>
<script> game.ontick = function() {} </script>
```

#2: Node

```
// node.str
interface IReadableStream {}
interface IProcess { stdin: IReadableStream; }
declare var process: IProcess;
   declare module "http" { }

// um.str
///<reference path='node.str' />
module 'um';
export function start(in: IReadableStream) {}

// umconsole.str
///<reference path='node.str' />
///<reference path='um.str' />
module um from 'um';
um.start(process.stdin);

   //socketio.str
   declare module "socket.io";
   export function listen() { }

// umserver.str
///<reference path='node.str' />
///<reference path='um.str' />
///<reference path='socketio.str' />

import http from 'http'
module socketio from 'socket.io'
module um from 'um'

var app = http.createServer()
var io = socketio.listen(app);
io.sockets.on('connection', function(socket) {
       var fakestdin: IReadableStream = { }
       um.start(fakestdin);
};
```

Questions:
What does module http from 'http' mean.

1. Substitute 'require("http")' everywhere http is referenced
2. 'module http from "http"' => 'var http = require("http")' (DO THIS FIRST)
3. If you ever mention http in variable position – put a binding to

Node: Files which generate any code need to either generate entirely within a module, or entirely out.

Import "Foo.Bar"
Import Win.Base
Import "Foo.Bar".Utils.Bar
Import "Foo.Bar" as X

3/26

```
// defining a module
Module Id.Id… { .. }
Module "…" { .. };

// to avoid indentation
Module …;
```

Two kinds of modules: (1) internal (2) loadable

Module foo at "io/foo";

```
// foo.str

Module "foo";
…

//bar.str
///<reference path='foo.str'/>
Module "bar"
Module foo at "foo";
```

[In 41/4725 at 1:00 today.]

Agenda:

- Overloads and contextual typing
  - Keep the rule for now, but may come back later
- Update on modules
  Modules likely need a way to say they are modules vs. not modules
  Modules need to have a nested lexical scope, non-modules need to not have a nested lexical scope
  Can we add the set of /// options to hook compiler options: nested scope, -module, default references
  Only the ones that affect LS (scoping, bindings) need to be in the file, rest can be on the command line
  #module ?

- Details of typing of literals
- String enums, overload on constants

number, bool, string: Have all the members of Number, Boolean, String, (including members of Object), but not the brands but are not convertible to. Assignemnt compatibility needs to handle passing "goo" to IStringLike.

Object types: Has the members of Object unless it re-defines them

Object types that have at least one call or construct signature: Has the members of Function unless it re-defines them (including prototype:any).

(foobar: FooBar) => void ⇔ { (foobar: FooBar): void}

FooBar[] ⇔ { [i: number]: FooBar } (NO)

FooBar[] => Array<FooBar> (the latter is fake compiler generated instatiation of IArray)

Hypothetical (not possible today):

```
interface IMyArray extends string[] {
  myProperty: number;
}
```

Known limitation for now: No (easy) to represent members added to array objets

```
Import Win.Base  as Y
```

RegExp: /foo/ is a RegExp

#3: AMD

```
// TODO: Similar to half web, half Node
```

#4: Strada compiler (library code rehostable in various environments)

- Compiler mode to require :any
- Progress on generics
- ES6 features: string interpolation, destructuring
- Review all remaining “Open” issues on the design items spreadsheet
- Others?