ECMAScript Proposal: First-Class Protocols

Stage 1 • Michael Ficarra • 65th Meeting of TC39

Plans For This Proposal

- Today
  - Present recent changes and current status
  - Receive feedback and further considerations from committee
- Soon
  - Update proposal (as necessary)
  - Update implementations (sweet.js, runtime polyfill, babel?) to match proposal
  - Send PRs to early adopters to update their usage
  - Evangelise and gather community feedback
  - Come back to committee for stage 2

PLEASE HOLD ALL SYNTAX BIKE-SHEDDING

(the current syntax is intentionally verbose)

Brief Overview: Components of this Proposal

Syntax For Declaring a Protocol

required fields

provided fields

```
protocol Foldable {
  foldr;

  toArray() {
    return this[Foldable.foldr](
      (m, a) => [a].concat(m), []);
  }

  get length() {
    return this[Foldable.foldr](
      m => m + 1, 0);
  }

  contains(eq, e) {
    return this[Foldable.foldr](
      (m, a) => m || eq(a, e),
      false);
  }
}
```

Inline Implementations For Existing Classes

```
protocol Foldable {

  // ...

  implemented by Array {
    foldr(f, memo) {
      // implementation elided
    }
  }
}
```

New ClassElement For Declaring Protocol Implementation

```
class NEList {
  constructor(head, tail) {
    this.head = head;
    this.tail = tail;
  }

  implements protocol Foldable {
    foldr(f, memo) {
      // implementation elided
    }
  }
}
```

Protocol Inheritance

```
protocol A { a; }
protocol B extends A { b; }

class C {
  implements protocol B {
    a() {}
    b() {}
  }
}

// or

class C {
  implements protocol A {
    a() {}
  }
  implements protocol B {
    b() {}
  }
}
```

Protocol Constructor

```
const Foldable = new Protocol({
  name: 'Foldable',
  extends: [ … ],
  requires: {
    foldr: Symbol('Foldable.foldr'),
  },
  staticRequires: { … },
  provides:
    Object.getOwnPropertyDescriptors({
      toArray() { … },
      get length() { … },
      contains(eq, e) { … },
    }),
  staticProvides: …,
});
```

Querying: implements

```
if (C implements P) {
  // reached iff C has all fields
  // required by P and all fields
  // provided by P
}
```

Dynamic Implementation

```
protocol Functor { map; }

class NEList {
  constructor(head, tail) {
    this.head = head;
    this.tail = tail;
  }
}

NEList.prototype[Functor.map] =
  function (f) {
    // implementation elided
  };

Protocol.implement(NEList, Functor);
```

Recapping: Components of this Proposal

Features Supporting Declarative Usage:

- protocol declarations
  - inline implementation declaration for pre-existing classes
  - protocol inheritance
  - `export` form
- new `ClassElement` for declaring protocol implementation (the required fields) inline

Features Supporting Dynamic Usage:

- `Protocol` constructor
- `implements` operator
- `Protocol.implement` API
- protocol expression form

Committee Feedback:
Globally-enforced Instance Coherence

Two Convenient Ways To Define Implementation

```
protocol P {
  // ...

  implementation for C {
    // ...
  }
}
```

```
class C {
  // ...

  implements protocol P {
    // ...
  }
}
```

encourages, but does not *enforce* coherence

Committee Feedback:
String-based Required Fields For Legacy Protocol Compatibility

String Required Fields

```
protocol P {
  "a";
  b(){ print('b'); }
}

class C {
  a() {}
  implements protocol P {}
}

C implements P; // true
(new C)[P.b](); // prints 'b'
```

Committee Feedback: Relationship to Mixins Proposal

mixins: incredibly simple syntax sugar

with mixins:

```
mixin M0 {
  f() { console.log('f'); }
}

mixin M1 {
  g() { console.log('g'); }
}

class C extends S with M0 with M1 {
  h() { this.f(); this.g(); }
}
```

without mixins:

```
const M0 = S => class M0 extends S {
  f(){ console.log('f'); }
}

const M1 = S => class M1 extends S {
  g(){ console.log('g'); }
}

class C extends M0(M1(S)) {
  h() { this.f(); this.g(); }
}
```

mixins: antithetical to composition

with mixins:

```
mixin M0 { f() { print('M0.f'); } }
mixin M1 { f() { print('M1.f'); } }

class C with M0 with M1 {
  g() {
    this.f(); // prints 'M0.f'?
    // oh dear, we've effectively
    // created a new shared global
    // namespace
  }
}
```

with protocols:

```
protocol M0 { f(){ print('M0.f'); } }
protocol M1 { f(){ print('M1.f'); } }

class C {
  implements protocol M0 {}
  implements protocol M1 {}
  f() {
    this[M0.f](); // prints 'M0.f'
    this[M1.f](); // prints 'M1.f'
  }
}
```

Committee Feedback:
Relationship to Decorators Proposal

Protocols & Decorators

- Decorators are very powerful; we could probably do this whole thing without new syntax
- Inline implementation blocks would be tricky
- At that point, might as well just use `new Protocol({ /* ... */ })`

Questions & Feedback