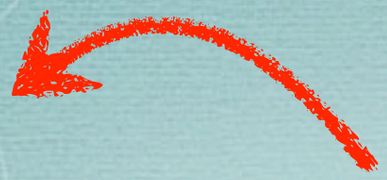
Ecma/TC39/2010/017



(Keynote "Harmony" theme)

SIMPLE MODULES

Sam & Dave

Goals

- **Simplicity:**
  - Smooth migration from small- to large-scale
  - Seamless loading over the web
  - Lightweight linking
- True lexical scope
- Controlled reflection
- Isolation
- Cross-file, cyclic dependencies
- Compatibility with existing code

Emaker-style goals

- Isolation
- Security
- No global namespace
- No constraints on simultaneous “versions”
- First class objects
- Non-blocking without boilerplate
- Zero-admin provisioning
- Uniform location and retrieval

Shared goals

- Isolation
- Security
- No global namespace
- No constraints on simu[illegible]
- First class objects
- Non-blocking without boilerplate
- Zero-admin provisioning
- Uniform location and retrieval

compatible

partial

host

TOUR

A simple module

```
<script type="harmony">
module Math {
    export function sum(x, y) {
        return x + y;
    }
    export var pi = 3.141593;
}
</script>
```

A simple client

```
<script type="harmony">
// works in script code!
import Math.{sum, pi};
import DOM.{alert};

alert("2π=" + sum(pi, pi));
</script>
```

Basics

* Modules are statically defined (“second-class”)
* Nothing in scope at outset
* Only exported top-level definitions are public
* All imported/exported bindings are immutable

Convenience modes

```
<script type="harmony-script">
// works in script code!
import Math.{sum, pi};

alert("2π=" + sum(pi, pi));
</script>
```

Convenience form

```
<script type="harmony-script">
// import and bind everything
import Math.*;

alert("2π=" + sum(pi, pi));
</script>
```

Lightweight instance objects

```
<script type="harmony-script">
// M is an immutable "module instance object"
import Math as M;

alert("2π=" + M.sum(M.pi, M.pi));
</script>
```

Local renaming

```
<script type="harmony-script">
import Geometry.{ drawShape: draw };
import Cowboy.{ drawGun: draw };

...
</script>
```

Summary of importing

* Import specific names (possibly renamed)
* Import all names (still lexically scoped!)
* Import modules as values

Cyclic dependencies

```
module Even {
    import Odd.*;
    export function even(n) {
        return n == 0 ||
                    odd(n - 1);
    }
}
```

Cyclic dependencies (*cont'd*)

```
module Odd {
    import Even.*;
    export function odd(n) {
        return n == 0 ? false
                      : even(n - 1);
    }
}
```

Parameterization

```
module DomMunger {
    // parameterized by a DOM implementation
    export function make(domAPI) {
        return {
            munge: function(doc) {
                ...
                domAPI.alert(“hi!”);
                ...
            }
    }; } }
```

Parameterization (*cont'd*)

```
module SafeDom {
    import Dom;
    export var document = {
        write: function(txt) {
            Dom.alert("sorry, Dave...")
        },
        ...
}; }
```

Parameterization (*cont'd*)

```
<script type="harmony">
import DomMunger as DM;
import SafeDom as Dom;

var instance = DM.make(Dom);

instance.munge(...);
</script>
```

Remote modules

```
<script type="harmony"
        src="http://json.org/m.js" />
```

```
<script type="harmony-script">
import JSON;

alert(JSON.stringify({"hi": "world"}));
</script>
```

The common case

* This covers much of what ordinary programs do.
* Goal: incorporate rest while keeping simplicity of what we just described.

MANAGING MODULES

Modules are powerful

* They get to have shared global state.
* They get to name themselves.
* They get to refer to other modules.

Tools to manage modules

Two primary tools:

1. Namespace isolation
2. State isolation

Namespace isolation

- Separable registries of module names
- Ability to instantiate code in a fresh registry
- Protect against accidental name conflicts

State isolation

- Custom module loaders with separate module instances
- May restrict what modules you can name, load
- May specify mapping between module names, instances
- Selectively share module instances

Reflective API

- `new ModuleLoader(...)`
- `ml.loadModule(str, callback)`
- `ml.attachModule(str, minst)`
- `ml.evalScript(src)`