TC
39

Builtin Modules

Formerly JavaScript Standard Library

Seeking stage 2

https://github.com/tc39/proposal-built-in-modules

Champions: Michael Saboff, Mattijs Hoitink & Mark S. Miller

Currently Stage 1

Acknowledgement

- I want to thank various people who have provided feedback and suggestions to work through current issues:
  Mattijs Hoitink, Keith Miller, Mark S. Miller, Jordan Harband,
  Shu-Yu Guo, Devin Rousso, Kris Kowal & Chip Morningstar

Agenda

- Goals
- BuiltInModule object
- High Level Operation
- Register ModuleSpecifier prefix `js:` with IANA?
- Stage 2?

Review

- This proposal was originally called JavaScript Standard Library.
- Provides the mechanism to deploy a standard library of modules provided in the implementation. The proposal does not define any modules itself only the mechanism.

Review

- This proposal was originally called JavaScript Standard Library.
- Provides the mechanism to deploy a standard library of modules provided in the implementation. The proposal does not define any modules itself only the mechanism.
- Stage 2 blockers

Review

- This proposal was originally called JavaScript Standard Library.
- Provides the mechanism to deploy a standard library of modules provided in the implementation. The proposal does not define any modules itself only the mechanism.
- Stage 2 blockers
    - ➡ Allow scripts to import and shim

Review

- This proposal was originally called JavaScript Standard Library.
- Provides the mechanism to deploy a standard library of modules provided in the implementation. The proposal does not define any modules itself only the mechanism.
- Stage 2 blockers
    - Allow scripts to import and shim - Proposal changed to address concerns. Presented at June 2020 meeting.

Review

- This proposal was originally called JavaScript Standard Library.
- Provides the mechanism to deploy a standard library of modules provided in the implementation. The proposal does not define any modules itself only the mechanism.
- Stage 2 blockers
  - ➡ Allow scripts to import and shim - Proposal changed to address concerns. Presented at June 2020 meeting.
  - ➡ Originally specified a coordinated namespace

Review

- This proposal was originally called JavaScript Standard Library.
- Provides the mechanism to deploy a standard library of modules provided in the implementation. The proposal does not define any modules itself only the mechanism.
- Stage 2 blockers
  - Allow scripts to import and shim - Proposal changed to address concerns. Presented at June 2020 meeting.
  - ~~Originally specified a coordinated namespace~~ *Eliminated.*

Goals

- To provide the mechanism to deploy a standard library of modules provided in the implementation.
  Note, this proposal does not define any modules itself only the mechanism.

- Advantages of module based library over adding to Global Object.
  - Reduces namespace pressure and collisions of top level names.
  - Hosts can implement modules as loadable components reducing memory footprint by only loading the modules needed by app / webpage.
  - Reduce page load time by providing common components locally.
  - Give JavaScript a library model similar to most every other language.
  - Hopefully accelerate process to add new library components.

New `BuiltInModule` Object

Add a new `BuiltInModule` object with the following prototype methods:

- `hasModule(moduleSpecifier)`- Returns boolean based on presence of a module in the the built in module map with `moduleSpecifier` **key**.
- `import(moduleSpecifier)` - Returns the exports for module with `moduleSpecifier` **key** from the built in module map.
- `export(moduleSpecifier, exports)` - Adds or replaces the exports for module with `moduleSpecifier` **key** in the built in module map.
- `freezeModule(moduleSpecifier)` - Disallow modification of module exports for module with `moduleSpecifier` **key**.
- `freezeAllModules()` - Freezes all modules in the built in module map.

Importing a BuiltIn Module

From a module:

```
    import “js:Complex”;

    import * as Comp from “js:Complex”;

    let complexPromise = import(“js:Complex”);
```

Importing a BuiltIn Module

From a module:

```
    import “js:Complex”;

    import * as Comp from “js:Complex”;

    let complexPromise = import(“js:Complex”);

    let complex = BuiltInModule.import(“js:Complex”);
```

Importing a BuiltIn Module

From a module:

```
import “js:Complex”;

import * as Comp from “js:Complex”;

let complexPromise = import(“js:Complex”);
```

Preferred

```
let complex = BuiltInModule.import(“js:Complex”);
```

Importing a BuiltIn Module

From a module:

```
    import “js:Complex”;

    import * as Comp from “js:Complex”;

    let complexPromise = import(“js:Complex”);

    let complex = BuiltInModule.import(“js:Complex”);
```

From a script:

```
    let complex = BuiltInModule.import(“js:Complex”);

    let complexPromise = import(“js:Complex”);
```

Importing a BuiltIn Module

From a module:

```
import “js:Complex”;

import * as Comp from “js:Complex”;
```

Synchronous

```
let complexPromise = import(“js:Complex”);
```

Async

```
let complex = BuiltInModule.import(“js:Complex”);
```

Sync

From a script:

```
let complex = BuiltInModule.import(“js:Complex”);
```

Sync

```
let complexPromise = import(“js:Complex”);
```

Async

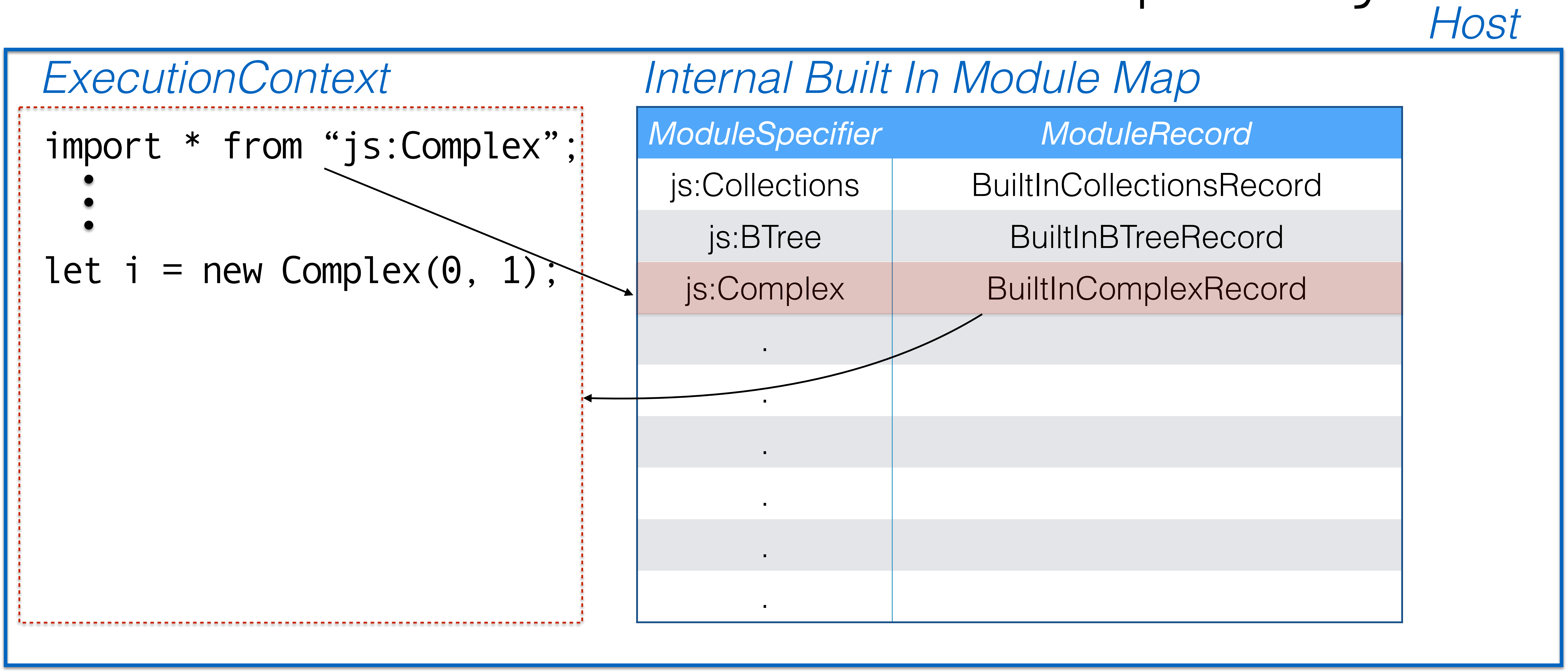
Built In Module Map

Part of the Host

Internal Built In Module Map

ModuleSpecifier (key)	*ModuleRecord*
ModuleSpecifier_1	$\text{BuiltInModuleRecord}_1$
ModuleSpecifier_2	$\text{BuiltInModuleRecord}_2$
.	.
.	.
.	.
ModuleSpecifier_N	$\text{BuiltInModuleRecord}_N$

Built In Modules Conceptually



```
import * from "js:Complex";
   •
   •
   •
let i = new Complex(0, 1);
```

ModuleSpecifier	ModuleRecord
js:Collections	BuiltInCollectionsRecord
js:BTree	BuiltInBTreeRecord
js:Complex	BuiltInComplexRecord
.	
.	
.	
.	
.	
.	

Shimming

- Built in modules need to be “shimmable”.
- If required, shimming needs to happen before the “main app” code runs.
- Shims can be applied to prior shims.
- Setup code shimming code needs the ability to lock down the resulting shimmed modules.

Shimming Example

Check for builtin, provide polyfill if module is missing:

```
if (!BuiltInModule.hasModule(“js:Complex”))
   BuiltInModule.export(“js:Complex”,
       { Complex: myComplexPoly });
```

Shimming Example

Check for builtin, provide polyfill if module is missing:

```
if (!BuiltInModule.hasModule(“js:Complex”))
   BuiltInModule.export(“js:Complex”,
       { Complex: myComplexPoly });
```

Shim part of a builtin:

```
let shimmedComplex = BuiltInModule.import(“js:Complex”);
shimmedComplex.toString = myComplexToString;
BuiltInModule.export(“js:Complex”,
    { Complex: shimmedComplex });
```

BuiltIn Module Names

- Modules added by TC-39 will begin with the prefix `js:`, e.g. `js:Complex`.
    - Other uses of the `js:` prefix are non-standard.
    - Organizations such as other standards bodies can use other prefixes. e.g. TC-53, OpenJS Foundation, implementors…
    - Formal coordination of prefixes was rejected by TC-39 (July 2019) as well as by the W3C (Sept 2019).
    - **Should TC-39 register “js:” with IANA?**

Questions?

Stage 2?

Thank you!

Automatice Module Loading

Builtin Modules as Globals

- Web developers are accustomed to accessing features via globals.
- Some concern Builtin Modules will provide too much friction for devs.
- We can provide an automatic means to import Builtin Modules as globals.

Automatic BuiltIn Module Importing

Provide a new internal Object for each module to automatically load on first access. Conceptually like:

```
function loadSelf(prefix, moduleName)
{
    globalThis[moduleName] = BuiltinModule.import(prefix + moduleName);
}

Object.defineProperty(globalThis, "Complex" {
    get: function() {
        loadSelf(“js:”, "Complex");
        return globalThis.Complex;
    },
    configurable: true,
    enumerable: true
});
```

This internal object should resolve as an Object for `typeof` and `instanceof` without loading the module. Likely this is implemented in native C++ code with a table listing each such module.

Advantages of Auto Load to a Global

- Provides module availability as global objects while preserving the intent of Builtin Modules.

- Modules could be shimmed and polyfilled as they are today.

- Promote an unforced community driven transition to Builtin Module.

- It would provide a standard means for lazy loading features.

- Legacy features could transition to being builtin modules without any compatibility issues.

Disadvantages of Auto Loading

- Isn't needed for other JS hosts, e.g. Node and IOT.
  - ➡ Would making this normative optional be acceptable?
  - ➡ Is this counter to a "one JavaScript" goal?
- Doesn't solve namespace pressure issues.
- Could confuse devs as to which way they access a feature.

Thank you!