Private Names and At-names


Allen Wirfs-Brock
Mozilla
September 18, 2012
Unique/Private Names

• Unique and private names (aka symbols) are ES6’s solution for objects that need to expose properties that have limited or controlled accessibility.

• Currently no syntactic support for definition or use.

```javascript
const secret = Name();

let obj = {};

obj[secret] = 42;
```
Basic Problem

• The current imperative code patterns for using names don’t mesh well with declarative object/class definitional forms.

```javascript
const secret = Name();
let obj = {secret: 42};  // does not define an
                          // unique name properties

Class MyClass extends Another {
  secret() {this.mine(); super[secret]();
}  // secret is a regular property string key
   // when used as method name but evaluated
   // to a name when used within [ ]
```
Some ES6 features are only available using declarative forms.

• Current no way to define a method that has a unique/private property name and references super.

```javascript
const secret = Name();
//class MyClass extends Another {
//    secret() {this.mine(); super[secret](); }
//}
// secret is a regular property string key

//The following is currently illegal:
class MyClass extends Another {...};
MyClass.prototype[secret] =
    function () {this.mine(); super[secret](); }
```
Computed Property Names for Object Literals Were Abandoned

```javascript
const secret = Name();
let obj = {[secret]: 42};  // does not define an
// unique name properties

Class MyClass extends Another {
  [secret]() {this.mine(); super[secret]();
}
```

**Issues**

– Allowed arbitrary expr. in prop name def. position
– Allowed aliasing of string valued prop keys
– Permitted same key to occur more than once
– Future hostile: ties property def. to indexing syntax
Also, some minor convenience issues

• Forced to use [ ] rather than . for name-based property accesses.
  – Hostile to future adoption of an extension model for [ ].

• Forced to explicitly call the Name constructor to create a new unique or private Name object.
At-Names

• An At-name is an IdentifierName that is lexically prefixed with @
• At-names are const bound to Name values by new declaration forms:

        name @x, @y;
    private @secret;
• Such declarations implicitly create new Name objects.
• Normal lexical scoping rules.
• No cross-talk between an At-name and a non-At-name binding of the same base IdentifierName
• “name” is a contextually reserved word: first token of statement and followed by an At-name
• name/private declarations make it unnecessary to expose the Name constructor objet.
At-Name References

• At-names can appear in any context where an IdentifierName would be interpreted as a literal property name.
  – As a property name in object literals/class definitions
  – After . in MemberExpressions

• Lexical scoping rules resolve such At-name references to a Name object value.

```javascript
private secret;
let obj = {@secret: 42};

Class MyClass extends Another {
    @secret() {mine(); super.@secret()}
}
```
At-names in Primary Expressions

- When used as a Primary expression, an At-name simply lexically resolves to the visible const name binding of the At-name.

  ```javascript
  obj.@secret;
  obj[@secret]
  ;   //mean the same thing
  ```

  ```javascript
  let f = (o,k) => o[k];
  ```

  //At-name values assignable to normal bindings
  ```javascript
  f(@secret);   //o[k] === o[@secret] === o.@secret
  ```
Name declarations with initializers

• In a name/private declaration, each At-name may have an initialization expression.

  private @secret = NameBroker.provideName(secretCode);

• The initializer must evaluate to a name object.
• The primary use case is initializing an At-name to a name provided via a function call or other computed value
Optional Feature – Class-scoped name declarations

• Allow name/private declarations to occur as a class body element.
• Any such declared At-names are scoped to the class body.

```javascript
class Point {
    private @x, @y; // <=== scoped to class body
    get x() { return this.@x }
    get y() { return this.@y }
    constructor (x,y) {
        this.@x = x;
        this.@y = y;
    }
}
```
Optional Future Feature – Class-scoped protected name declarations

- Allow “protected” declarations to occur as a class body element.
- Any such declared At-names may be explicitly imported into subclass body scopes.

```javascript
class Point {
  protected @x, @y;
  constructor (x,y) {
    this.@x = x;
    this.@y = y;
  }
}
class Point3D extends Point {
  protected @x, @y, @z;
  constructor (x, y, z) {
    this.@x = x;
    this.@y = y;
    this.@z = z;
  }
}
```

- Inherited protected at-name values are dynamically determined at class initialization time.
- Explicit declaration of inherited at-names avoids open with-like scoping issues.
Proposal for ES6

• Add At-names and `name/private` declarations
• At-names as properties keys supported in object literals and class bodies.
• Allow aliasing of unique/private names via initializers in `name/private` declarations
• Also allow `name/private` declarations to occur in class bodies.
• Added at meeting: name/private declarations in obj literals. Also private/name as prefix.
• Continue to explore future support for `protected`. 