New ES6 terminology

Allen Wirfs-Brock

May 2012

New Terminology: Objects

- *object* - An runtime entity that has unique identity and exposes properties (via implementations of the required "internal methods" specified in chapter 8)
- *Ordinary object* - An object that that uses only default behaviors for the required internal methods (as specified in chapter 8).
- *exotic object* - An object that provides non-default behavior for at least one of the required internal methods.

Exotic objects encompasses Proxies and most of what are currently called "host objects". It also includes some chapter 15 objects such as array instances that have non-default internal method behaviors.

New Terminology: Object Providers

- *standard object* - An object whose application level semantics are defined by aECMAScript specification.
- *built-in object* - an object that is provided by the ECMAScript implementation.
- *platform object* - An object that is provided by the environment that hosts the ECMAScript implementation.

Each of the above three categorizations can include both mundane and exotic objects. The distinction between built-in object and platform object is probably of minor importance.

New Terminology: Functions

- *function* - An object that exposes the [[Call]] internal method.
- *ECMAScript function* - A function whose invocation result and side-effects is proved by evaluating ECMAScript code.
- *alien function* - A function whose invocation result and side-effects is provided in some manner other than by evaluating ECMAScript code.
- *standard function* - a function whose invocation result and side-effects are defined by the ECMAScript specification (mostly chapter 15)

An ECMAScript function might be either a mundane or an exotic object. An alien function is always an exotic object because the default [[Call]] internal method produces the invocation result and side-effects by evaluating ECMAScript code. A standard function can potentially be implemented either as an ECMAScript function or an alien function.

New Terminology: Island, Home

- A “top level” ECMAScript environment with its own global environment, intrinsic objects, global ambient state, etc.

May 4 Draft: Feature additions

- Added syntax and semantics for Binary and Octal integers
- Added syntax/semantics for super in *MemberExpressions* and *CallExpressions*
- Added arrow functions (13.2) and concise methods (13.3)
- added Object.isObject
- added Array.of and Array.from
- added String.prototype repeat, startsWith, endsWith, contains, toArray
- added Number.EPSILON,MAX_INTEGER,parseInt, parseFloat,isNaN,isFinite, isInteger, toInt
- added Number.prototype.clz
- added Math.log10, log2, log1p, expm1, cosh, sinh, tanh, acosh, asinh, atanh, hypot, trunc, sign, cbrt

May 4 Draft: Editorial/Technical 1

- Clarified that *IdentifierNames* can include escape sequences
- Extended Reference to support super references
- Added abstract operations for Object and Array creations
- Added arrow functions (13.2) and concise methods (13.3)
- Preliminary introduction of code "Realms" (contexts with their own globals, intrinsics, etc.) (incomplete)
- Added Method Environment Records as part of super support
- Extensions to execution contexts needed to support generators, super, and code realms
- Eliminated "enter execution context" algorithms by merging them with [[Call]], eval, *Program* etc.

May 4 Draft: Editorial/Technical 2

- General migration of most material related to functions and their execution into Chapter 13
- Added additional explicit checks of completion values.
- made yield illegal outside of generators
- additional work/cleanup on for-in/for-of (prep for array comprehensions)
- Started tracking Annex D and E additions
- refactored “Creating Function Objects” into separate function and constructor creation abstract operations.
- Cleaned-up Array constructor specification
- clarification of Number.MIN_Value for Arm processors (that don’t support denormalized numbers)

Arrow Functions

Syntax

ArrowFunction **:**
    ArrowParameters **=>** *ConciseBody*

ArrowParameters **:**
    BindingIdentifer
    (*ArrowFormalParameterList* **)**

ArrowFormalParameterList **:**
    [empty]
    FunctionRestParameter
    CoverFormalsList
    CoverFormalsList **,** *FunctionRestParameter*

ConciseBody **:**
    [lookahead ∉ { **{** }] *AssignmentExpression*
    { *FunctionBody* **}**

CoverFormalsList **:**
    Expression

Supplemental Syntax

When processing the production *CoverFormalsList* **:** *Expression* the *FormalsList* production is used to further restrict the source code that matches *Expression*.

ArrowFormalParameterList **:**
    FormalParameterList

Concise Methods

11.1.5 **Syntax**

ObjectLiteral **:**
- **{ }**
- **{** *PropertyDefinitionList* **}**
- **{** *PropertyDefinitionList* **, }**

PropertyDefinitionList **:**
- *PropertyDefinition*
- *PropertyDefinitionList* **,** *PropertyDefinition*

PropertyDefinition **:**
- *IdentifierName*
- *PropertyName* **:** *AssignmentExpression*
- *MethodDefinition*

NOTE *MethodDefinition* is defined in 13.3.

13.3 **Syntax**

MethodDefinition **:**
- *PropertyName* **(** *FormalParameterList* **)** *ConciseBody*
- ***** *PropertyName* **(** *FormalParameterList* **)** *ConciseBody*
- **get** *PropertyName* **( )** *ConciseBody*
- **set** *PropertyName* **(** *PropertySetParameterList* **)** *ConciseBody*

PropertySetParameterList **:**
- *BindingIdentifier*
- *BindingPattern*

BTW…

Syntax

SealedObjectLiteral :
ObjectLiteral
ObjectLiteral

SealedArrayInitialiser :
ArrayInitialiser
ArrayInitialiser

?