Minutes of the 1st meeting of Ecma TC39 Special group on Secure ECMAScript held in Sunnyvale, CA, USA on 28 August 2008

Attending*: Mr. Tyler Close (HP), Mr. Douglas Crockford (Yahoo), Mr. David-Sarah Hopwood (Network Security), Mr. Scott Isaacs (Microsoft), Mr. Collin Jackson (Stanford), Mr. Marcel Laverdet (Facebook), Mr. Mark Miller (Google), Mr. Chip Morningstar (Yahoo), Mr. John Neumann (Ecma International) and Mr. Allen Wirfs-Brock (Microsoft).

By phone: Mr. Adam Peller (IBM), Mr. David Simmons (Microsoft) and Mr. Kris Zyp (Sitepen).

1 Opening, welcome and roll call

All conference papers are at SES.JSON.ORG.and attached to this report by reference.

1.1 Opening of the meeting (Mr. Crockford)

1.2 Introduction of the attendees

1.3 Host facilities, local logistics

2 Adoption of the agenda (2008/070)

Agenda adopted

3 Terms of Reference and goals/objectives

Doug Crockford outlined in his introductory remarks the purpose and goals of the meeting (see Introduction below).

4 Technical presentations

Douglas Crockford: Introduction, Problem, and Opportunity

Allen Wirfs-Brock: ES3.1 Object Methods

Marcel Laverdet: FBJS

Mark Miller: Caja, Cajita, and E

David-Sarah Hopwood: Jacaranda

Kris Zyp: dojox.secure (via phone)

Douglas Crockford: JSON, ADsafe, and Misty

5 Discussion and draft plan

The biggest problem to adoption and deployment of a new language is perception by browser manufacturers that there is no problem that needs fixing. This will likely change in the future when their current hot buttons are solved (i.e.: performance). The goal is to get rid of programs that can cause mischief on the web.

There are advantages to taking multiple approaches as outlined in the introduction as it will increase the likelihood that one or all will be accepted.

The problem needs to be clearly articulated and agreed. Then statements of possible solutions can be judged.

Mr. Neumann stated that the first step is to clearly define the problem. To that end, participants are invited to submit a contribution on this topic for the next meeting, particularly those who gave presentations on proposed solutions. The intent will be to delineate the issues surrounding security at the next meeting.

6 Any other business

None

7 Date and place of the next meeting(s)

The next meeting will be held in conjunction with the TC39 November meeting (November 18 – Hawaii)

8 Closure

The meeting ended at 5:45 PM

Secure

ECMAScript

Name Subject To Change

Douglas Crockford

Yahoo!

The purpose of this workshop is to consider the feasibility and necessity of a secure replacement for ECMAScript.

Security is our Number One Problem

All websites are under attack.

Progress is being frustrated.

Mash Up We Must!

Three Possible Solutions

- Safe JavaScript Subset.

  Timeframe: Immediate

- Communicating Vats.

  Timeframe: Intermediate

- Secure Programming Language.

  Timeframe: Distant

- All of the Above.

Safe JavaScript Subset

- Constrain the existing language by code rewriting and runtime repression or by static validation.
- The constrained language limits the capabilities that are given by default to a program.
- Good examples may inform the design of Ses.
- It may be good to derive a standard, but that is not the goal of this meeting.

Vats

- Secure containers for computation.
- Constrained intervat communication.
- First steps: Google's Gears Workerpools; durable **`<iframe>`**, XDM.
- Ultimately, transmission of capabilities, futures, distributed garbage collection.
- This is out of scope for today's meeting.

A New Language

- Similar, but not compatible.
- Retain the goodness of ECMAScript.
- Replace, repair, or remove the bad parts.

- JavaScript got a lot right.
- Minimize retraining.
- Capture programmers, not programs.

Goals

- A computation model that allows for cooperation under mutual suspicion.
- As simple as possible. Simple systems are easier to reason about.
- Approachable. The language must be usable by ordinary web developers.
- Unsurprising. Not freaky.
- Avoid confusion, difficulty, unmanagability.

Confusion of Interest

System Mode

Computer

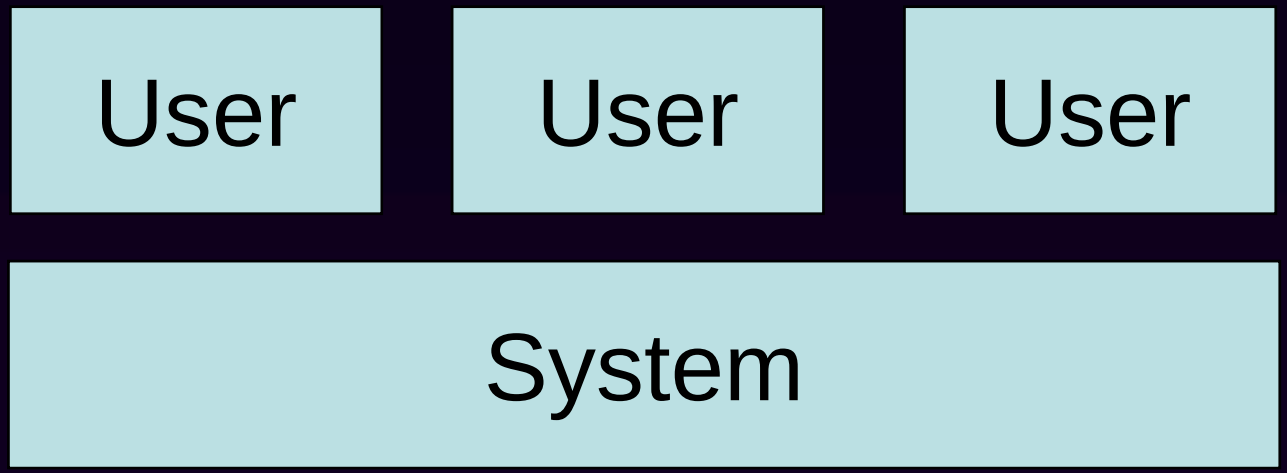
Confusion of Interest

System Mode

User

System

Confusion of Interest



Confusion of Interest

System Mode

CP/M MS-DOS MacOS Windows

Confusion of Interest

The System cannot distinguish the interest of the user from the interest of any program. This enables floppy-borne viruses.

CP/M MS-DOS MacOS Windows

Confusion of Interest

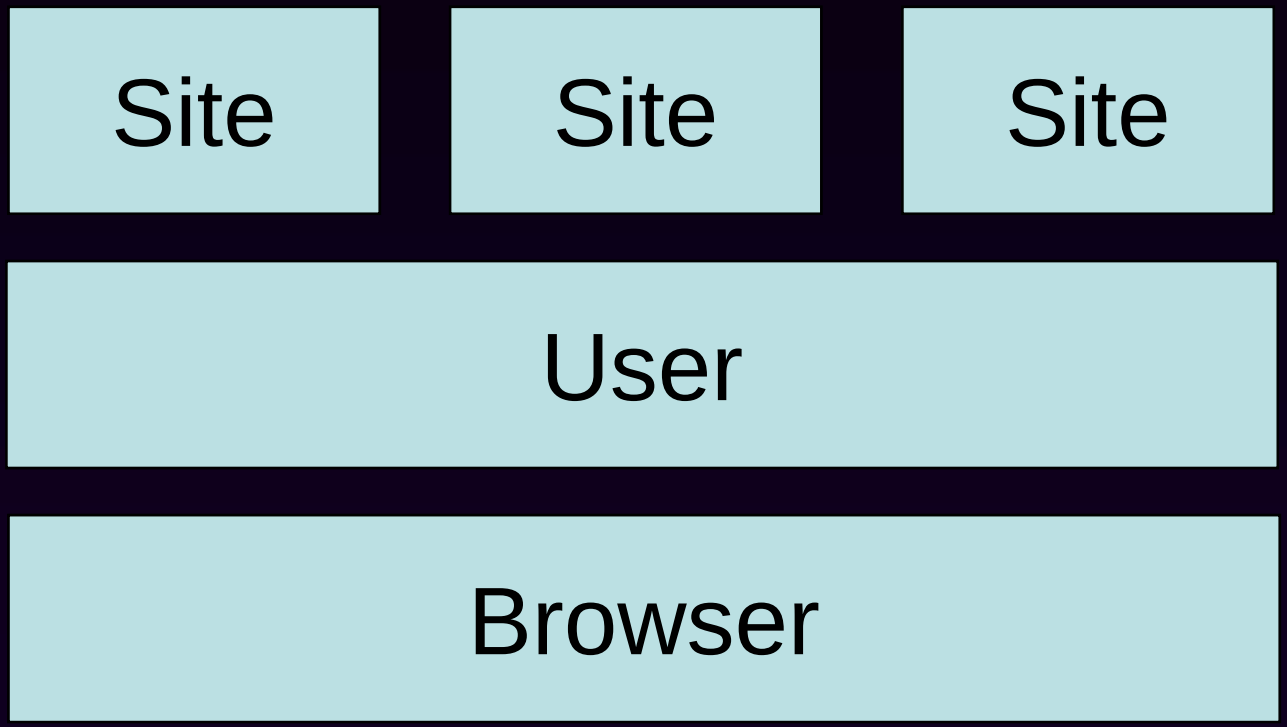
System Mode

When networking is introduced, network-borne viruses are enabled.

CP/M MS-DOS MacOS Windows

Confusion of Interest

The browser is a significant improvement, able to distinguish the interests of users and sites in some cases.



But within a page, interests are confused.

An ad or a widget or an Ajax library gets the same rights as the site's own scripts.

JavaScript got close
to getting it right.

Except for the Global Object.

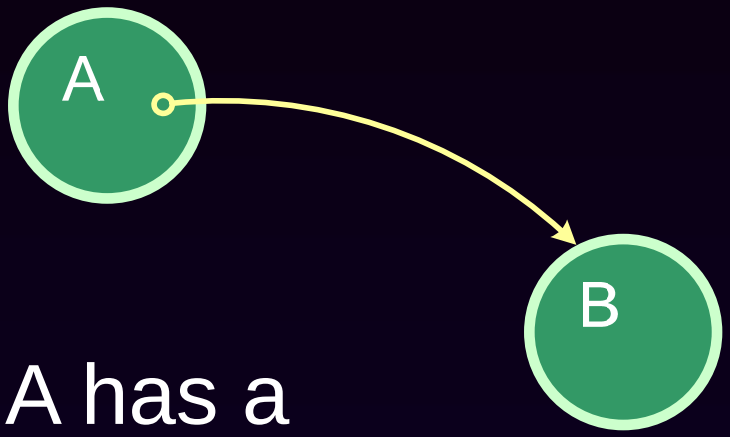
It can be repaired, becoming an
object capability language.

An Introduction to Object Capabilities

A is an Object.

Object A has state and behavior.

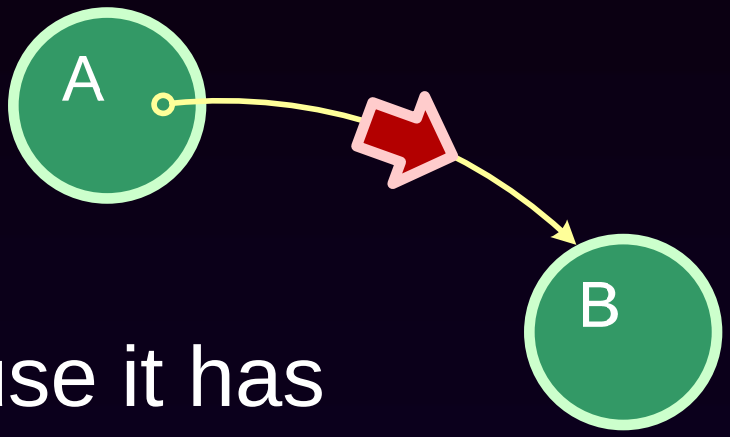
has-a



Object A has a reference to Object B.

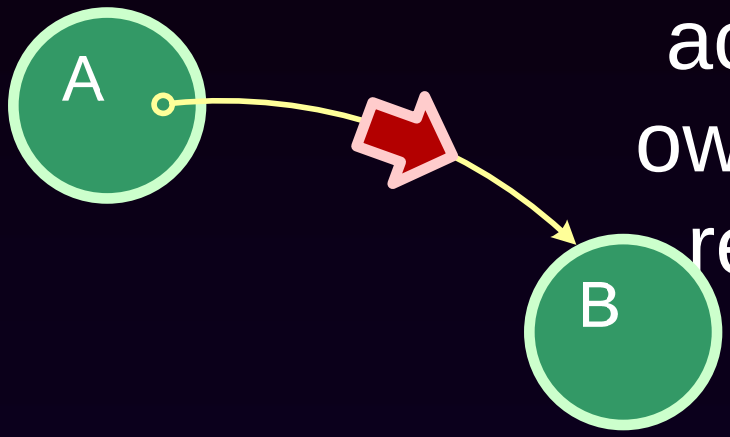
An object can have references to other objects.

Object A can communicate with Object B...



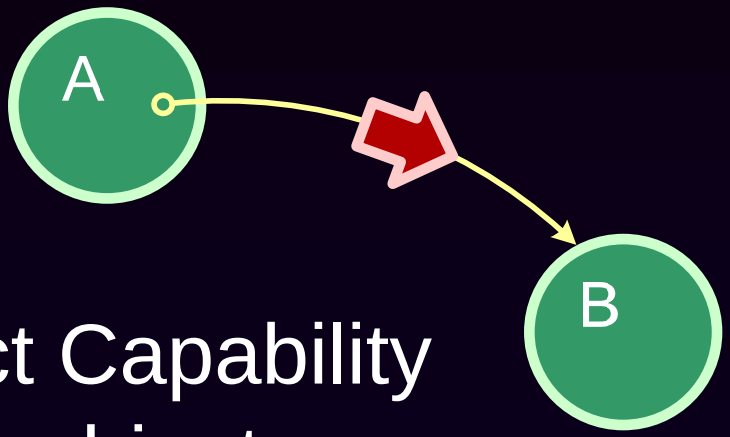
...because it has a reference to Object B.

Object B provides an interface that constrains access to its own state and references.



Object A does not get access to Object B's innards.

Object A does not have a reference to Object C, so Object A cannot communicate with Object C.



In an Object Capability System, an object can only communicate with objects that it has references to.

An Object Capability System is produced by constraining the ways that references are obtained.

A reference cannot be obtained simply by knowing the name of a global variable or a public class.

There are exactly three ways to obtain a reference.

1. By Creation.
2. By Construction.
3. By Introduction.

1. By Creation

If a function creates an object, it gets a reference to that object.

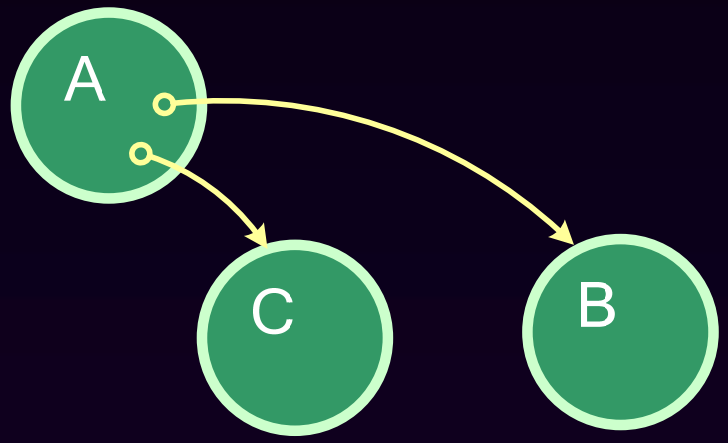
2. By Construction

An object may be endowed by its constructor with references.

This can include references in the constructor's context and inherited references.

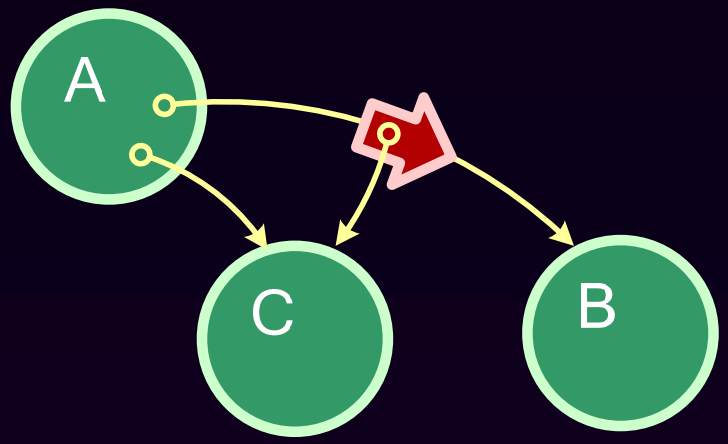
3. By Introduction

A has a references to B and C.
B has no references, so it cannot communicate with A or C.
C has no references, so it cannot communicate with A or B.



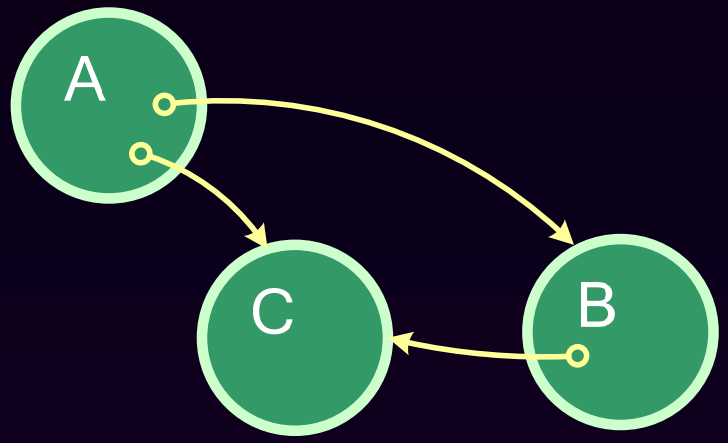
3. By Introduction

A calls B, passing a reference to C.



3. By Introduction

B is now able to communicate with C.



It has the capability.

If references can only be obtained by Creation, Construction, or Introduction, then you may have a safe system.

If references can be obtained in any other way, you do not have a safe system.

Potential weaknesses include

1. Arrogation.
2. Corruption.
3. Confusion.
4. Collusion.

1. Arrogation

- To take or claim for oneself without right.
- Global variables.
- public static variables.
- Standard libraries that grant powerful capabilities like access to the file system or the network or the operating system to all programs.
- Address generation.

2. Corruption

It should not be possible to tamper with or circumvent the system or other objects.

3. Confusion

It should be possible to create objects that are not subject to confusion. A confused object can be tricked into misusing its capabilities.

4. Collusion

- It must not be possible for two objects to communicate until they are introduced.
- If two independent objects can collude, they might be able to pool their capabilities to cause harm.
- For example, I can give gasoline to one object, and matches to another. I need to be confident that they cannot collude.

Rights Attenuation

- Some capabilities are too dangerous to give to guest code.
- We can instead give those capabilities to intermediate objects that will constrain the power.
- For example, an intermediate object for a file system might limit access to a particular device or directory, or limit the size of files, or the number of files, or the longevity of files, or the types of files.

Ultimately, every object should be given exactly the capabilities it needs to do its work.

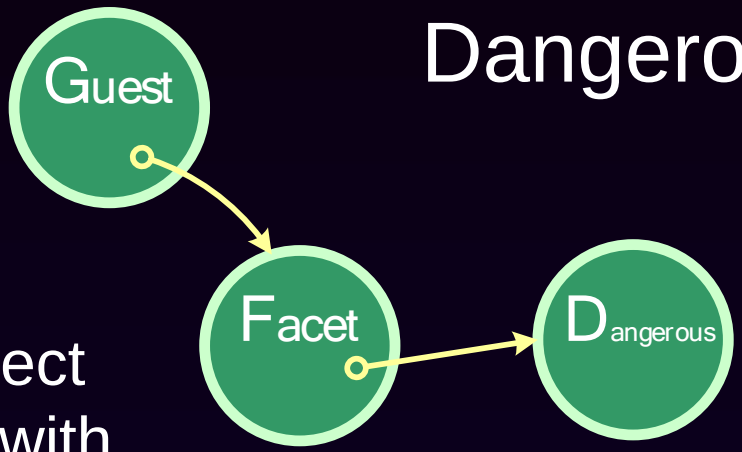
Capabilities should be granted on a need-to-do basis.

Information Hiding - Capability Hiding.

Intermediate objects, or facets, can be very light weight.

Class-free languages can be especially effective.

The Facet object limits the Guest object's access to the Dangerous object.

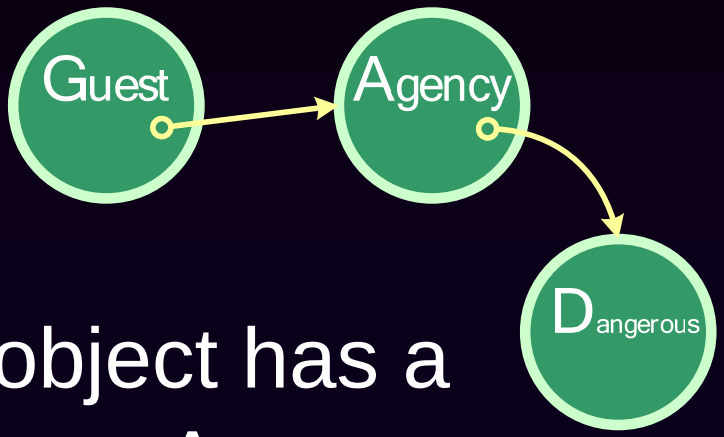


The Guest object cannot tamper with the Facet to get a direct reference to the Dangerous object.

References are not revocable.

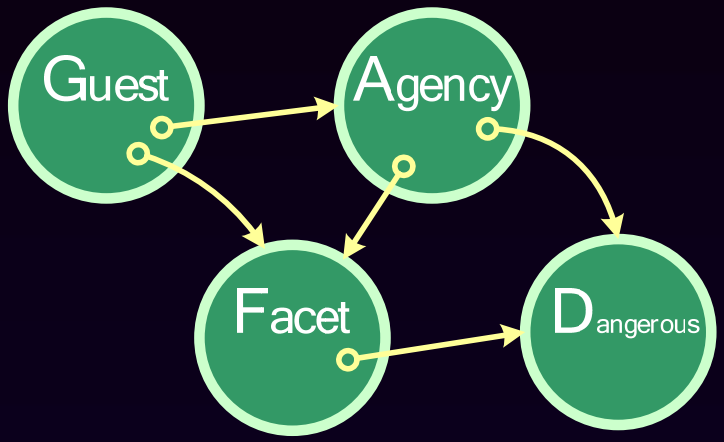
Once you introduce an object, you can't ask it to forget it.

You can ask, but you should not depend on your request being honored.



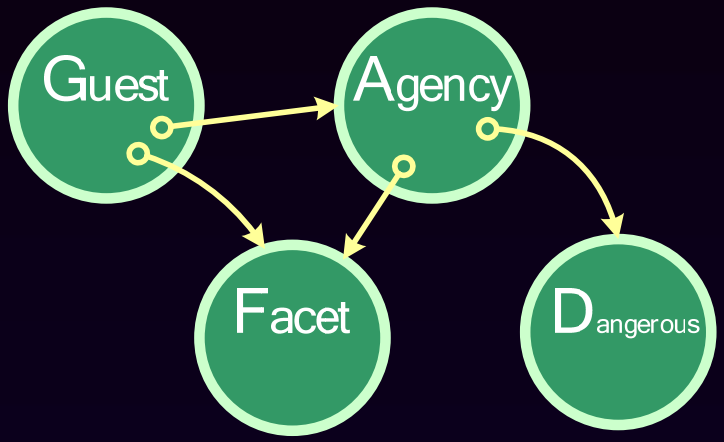
The Guest object has a reference to an Agency object. The Guest asks for an introduction to the Dangerous object.

The Agency object makes a Facet, and gives it to the Guest.



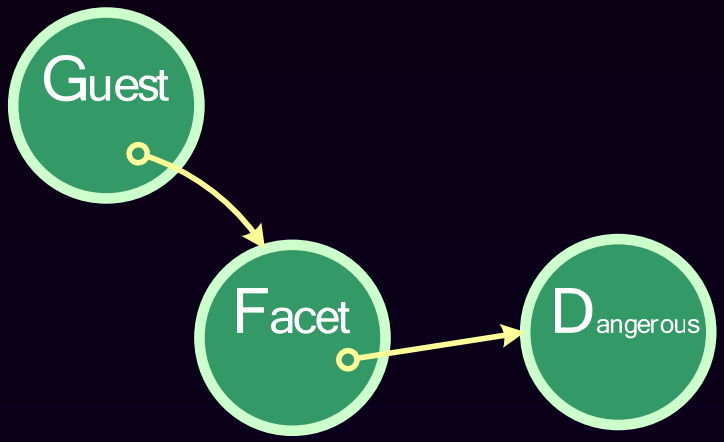
The Facet might be a simple pass through.

When the Agency wants to revoke the capability, it tells the Facet to forget its capability.



The Facet is now useless to the Guest.

A Facet can mark requests so that the Dangerous object can know where the request came from.



Facets

- Very expressive.
- Easy to construct.
- Lightweight.
- Power Reduction.
- Revocation.
- Notification.
- Delegation.
- The best OO patterns are also capability patterns

Good Object Capability Design
is
Good Object Oriented Design

Secure ECMAScript Must Be Incompatible With ES3

- If it were compatible, it would share the weaknesses of ES3.
- Incompatibility gives us license to correct many of the problems that ES3.1 must preserve.
- Lacking compatibility in the design process could lead to a lack of feature discipline.

Minimal

- An elegant, minimal language is easier to reason about than an over-featured, maximal language.
- Committees are generally unable to produce minimal designs.
- We should avoid a Design-by-committee.

Competition

- We invite members to submit designs.
- We drafts rules for the competition, and select a winner based on the criteria of security, expressiveness, and minimalism.
- Over several rounds of evaluation and influence, we may find either a clear winner or convergence on an ideal approach.

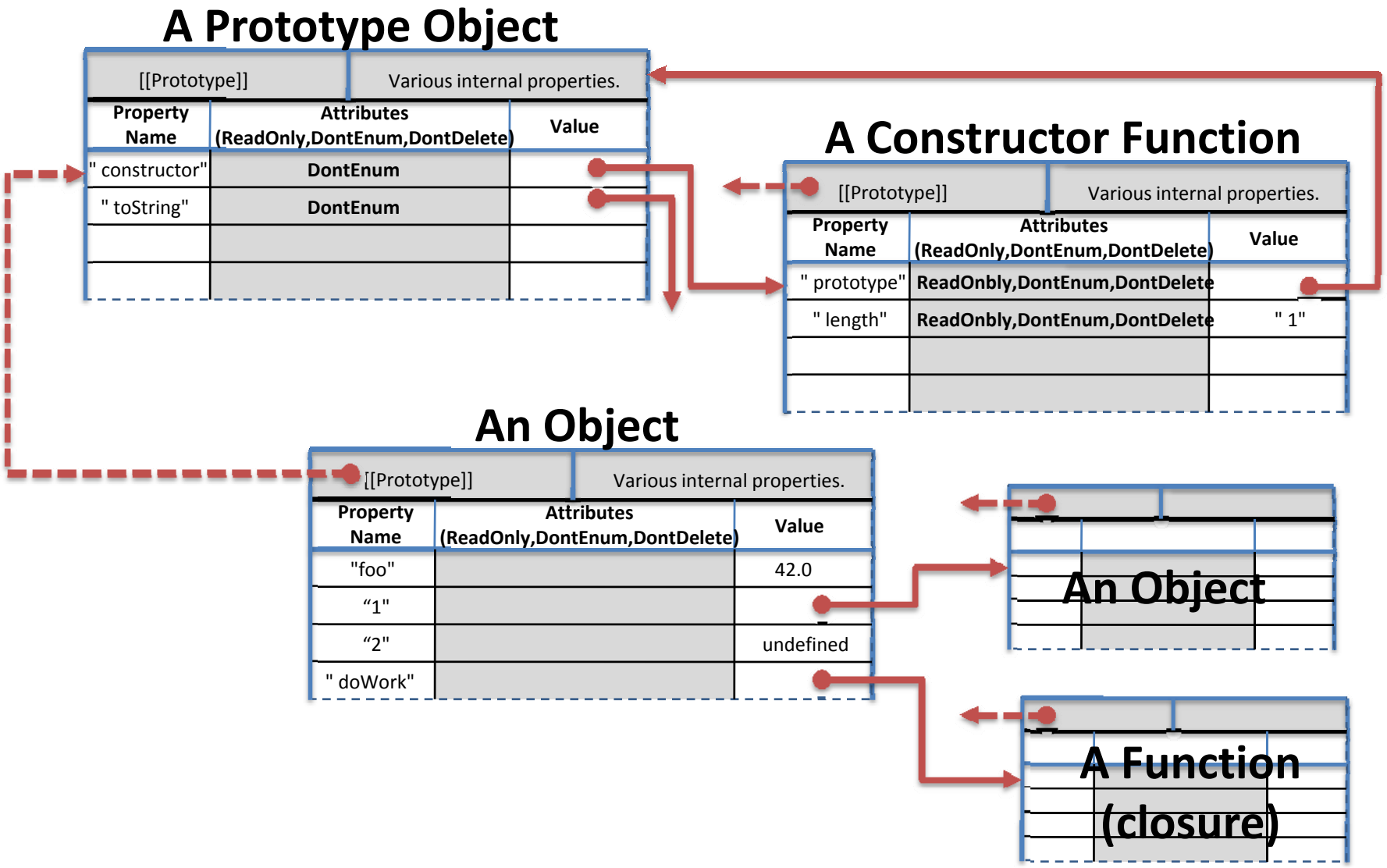
Review of Current Work

- ES3.1
- FBJS
- Caja, Cajita (and E)
- Jacaranda
- dojox.secure
- JSON, ADsafe (and Misty)

ECMAScript 3.1 Object Model

Allen Wirfs-Brock
Microsoft

ECMAScript 3 Object Model



ES3.1 Object Model Changes

- Rename/repurpose attributes
  - ReadOnly → Writable
  - DontEnum -> Enumerable
  - DontDelete -> Configurable
- Attribute values are programmatically settable and testable
- An object may be programmatically marked as “nonExtensible” (properties may not be added)
- Accessor Properties (getter/setter)

Configurable Attribute

- The [[configurable]] attribute of a property controls whether the definition of an attribute can be programmatically changed:
  - Delete the attribute
  - Change the state of a property attribute: writable, enumerable, configurable
  - Change/delete the getter and/or setter function of an accessor property.
  - Convert a data property to an accessor property or visa versa.
- If Configurable attribute is false for a property
  - None of the above can occur
  - Writable can be change from true to false

Manipulating Properties and Attributes

- "Static" functions accessed via Object constructor
  - Object.defineProperty(obj, propName, propDescriptor)
  - Define a property:

```
Object.defineProperty(o, "length", {
        getter: function() {return this.computeLength()},
        setter: function(value){this.changeLength(value)}  });
Object.defineProperty(o, "1", {value: 1,
                                enumerable: true, configurable: true});
```

  - Modify property attributes

```
Object.defineProperty(Array.prototype, "forEach",
                      {enumerable: false, writable:false, configurable: false});
```

Retrieving a Property Definition

- Object.getOwnPropertyDescriptor (obj, propName)

  var desc = Object. getOwnPropertyDescriptor(o, “length));

- Return value isdescriptor with data properties
  - value, writeable, enumerable, configurable
    or
  - getter, setter, enumerable, configurable

- Return value is usable as 3rd argument to Object.defineProperty

Object Lock-down

- Prevent adding properties to an object
  - Object.preventExtensions(obj)
- Prevent adding or reconfiguring properties
  - Object.seal(obj)
- Prevent adding, reconfiguring, modify the value of properties
  - Object.freeze(obj)

Other Object Meta Methods

- Object.defineProperties(obj, propName, descriptorSet)
- Object.create(protoObj, descriptorSet);
- Object.getOwnPropertyNames(obj)
- Object.getPrototypeOf(obj)
- Object.isExtensible(obj)
- Object.isSealed(obj)
- Object.isFrozen(obj)

Example

```
// a Point “class”.
function Point(x, y) {
   const self = Object.create(Point.prototype, {
                toString: {value: Object.freeze(function() {
                                    return '<' + self.getX() + ',' + self.getY() + '>')}},
                          enumerable: true},
                getX: {value: Object.freeze(function() {return x}),
                          enumerable: true},
                 getY: {value: Object.freeze(function() {return y}),
                          enumerable: true}
                 });
   return self;
}
```

Example

```
//Point as a non-final non-abstract root/mixin class where toString is a final method:
function PointMixin(self, x, y) {
   Object.defineProperties(self, {
                toString: {value: Object.freeze(function() { return'<' + self.getX() + ',' + self.getY() + '>')}},
                          enumerable: true},
                 getX: {value: Object.freeze(function() {return x}),
                          enumerable: true, flexible: true},
                 getY: {value: Object.freeze(function() {return y}),
                          enumerable: true, flexible: true}
                 });
}
function Point(x, y) {
   const self = Object.create(Point.prototype); // only for instanceof
   PointMixin(self, x, y);
   return Object.freeze(self);
}
Object.freeze(PointMixin);
Object.freeze(Point.prototype);
Object.freeze(Point);
```

FBJS

A brief overview

marcel laverdet

facebook

Goals

- Allow application developers to create rich applications on Facebook.
- Don’t endanger our user’s privacy.

facebook

Features

- Large subset of ECMA-262… except
  - eval() and with(){}
  - Prototypes of native objects [Objects, Array, Number, etc]
  - Several undocumented quirks
- DOM manipulation
  - Non-standard getters: getChildNodes, getFirstChild, setValue, etc. Most DOM attributes are supported in some fashion.
  - innerFBML, innerXHTML

facebook

Features

- Facebook-specific host objects
  - Ajax (XMLHttpRequest wrapper)
    - Capable of returning blessed FBML strings
    - Breaks same origin policies
  - Animation
  - Dialog
    - Capable of extending an application's DOM authority

DOM Authority

- Applications are granted authority to entire DOM branches
  - There are exceptions -- opaque nodes
- Host objects may grant authority to more branches.

facebook

facebook Marcel Georgés Laverdet II Friends Applications Inbox Home Settings Logout Search

Add to Profile

Free Gifts

Virtual Gifts Should Be Free

Settings | Invite Friends

Free Gifts | Game Gifts | Community | Received | Sent

You are viewing the **Storefront** collection. Show more gift collections

Page: 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | Page 1 of Storefront

Selected Gift

no gift selected

Search Gifts

Privacy Setting

Public

Latest Gift

Skunk by Heather Corey

Birthdays Today: Pedram Keyani, Venkat Venkataramani, Jeff Egbe

Advertise

Radiohead – "In Rainbows"

Radiohead's hit album "In Rainbows" is available now. Watch the video for "House Of Cards," the hit single here!

Want a MacBook for free?

Complete 2 reward offers from the top, prime and premium pages and receive a free MacBook! See if program is available in your area!

facebook

facebook Marcel Georgés Laverdet II Friends Applications Inbox Home Settings Logout Search

Profile settings Updates, FAQ and help

Last.fm Music

+ Invite your friends

Friends' Music | Music you might like 1 2 ... 5 Next

Compare music taste with a friend

Start typing a friend's name Compare

Sort by musical compatibility

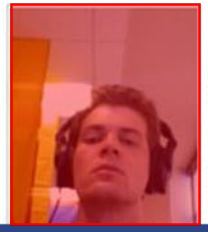
Chris Hughes + Add on Last.fm

Play Chris's favorite music

Last track played: The Ting Tings

Total tracks played: 7,508 | Last.fm profile

Charlie Cheever + Add on Last.fm



Play Charlie's favorite music

Now playing: Duffy – Rockferry

Total tracks played: 2,136 | Last.fm profile

Advertise

Jackie Chang is a fan of The Visa Business Network.

Grow your business.

THE VISA BUSINESS NETWORK BETA on Facebook

There's potential to reach millions of people on Facebook. Click to start connecting and receive $100 of free Facebook ads.

[?]

More Ads

facebook

facebook Marcel Georgés Laverdet II Friends Applications Inbox Home Settings Logout Search

Marcel Georgés Laverdet II

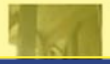
Wall | Info | Photos | Boxes | +

Graffiti

See all | Draw!

Julie Zhuo drew
at March 13 4:12am
Replay

Melissa Holtz (Google) drew
at November 18, 2007 5:49pm
Replay



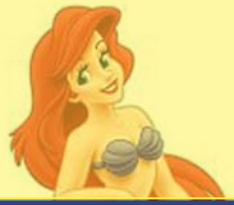
Valerie Hajdik drew
at November 18, 2007 12:01pm
Replay

Last.fm Music

Compare music

Edit settings

Which Disney Princess Are You?



Advertise

Looking for a IT job

Apply to the largest database of Software companies by state. We apply to hr of 1500 software companies in days for you.

Liza Heider Photography

Are you an actor? A model? A musician? Then you need a headshot! Don't wait. Contact Liza Heider today for that

facebook

facebook | Marcel Georgés Laverdet II | Friends | Applications | Inbox | Home | Settings | Logout | Search

Nat Brown

Wall | Info | Photos | Boxes | Music

iLike Sidestage 2

Nat has added 16 songs and likes 154 artists. Edit Settings

Write a comment about my music...

Post

Songs iLike — See All (16)

- I Like by Men Without Hats — iLike
  Buy – Dedicate
- Kick Out the Jams by The Presidents Of The United States Of America (on tour) — iLike
  Buy – Dedicate
- Thunderstruck by AC/DC — iLike
  Buy – Dedicate – Play video
- Ring Of Fire by Social Distortion (on tour) — iLike
  Buy – Dedicate

Add music to your profile:

Artists iLike — See All (154)

Photos

2 of 3 albums — See All

Mobile Uploads
Updated August 7

Doing the Bauhaus
Created August 4

facebook

Nat Brown

Wall | Info | Photos | Boxes | Music

iLike Sidestage 2

Nat has added 16 songs and likes 154 artis

Write a comment about my music...

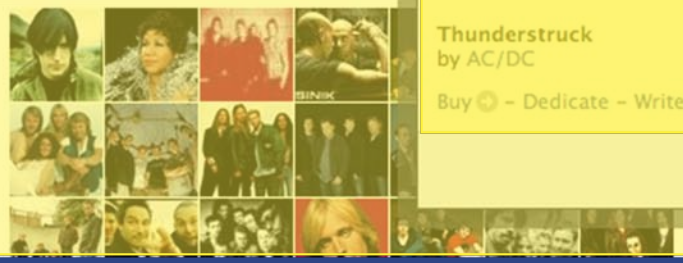
Post

Songs iLike

- I Like by Men Without Hats
  Buy – Dedicate
- Kick Out the Jams by The Presidents O
  States Of America (on tour)
  Buy – Dedicate
- Thunderstruck by AC/DC
  Buy – Dedicate – Play video
- Ring Of Fire by Social Distortion (on to
  Buy – Dedicate

Add music

Artists iLike



Thunderstruck by AC/DC

Thunderstruck
by AC/DC

Buy – Dedicate – Write a comment (see all)

Is this the right video?

Close

facebook

Entry Points

- Canvas pages may execute Javascript onDOMReady
- FBJS embedded on a profile must receive an “active” DOM event before being activated

facebook

Implementation

- Lexer [not a parser :( ]
  - Namespaces identifiers by using a unique application id
- Statically removes known evils:
  - constructor, caller, __proto__, __parent__, etc
- Run-time which also imposes the static limitations

facebook

What We Don’t Address

- Communication between applications
- Information leakage: img.src, Ajax (XMLHttpRequest)
- Method stealing, and mutability

facebook

The Future

- Secure and implicit getters and setters -- getParentNode is cumbersome
- Browser eccentricities be gone!
- Allow simulated mutability of native objects
- `eval(str)` and `new Function(str)` under controlled circumstances
- Selected features from future iterations of Javascript

facebook

Thanks

facebook

When Two Languages Are Simpler Than One

Lessons for SES from Cajita, Original-Caja, and Valija

Mark S. Miller

Simultaneous Problems

D = Defensive code problem
O = Offensive code problem
T = Legacy tools problem
C = Legacy code problem

Simultaneous Solution?

D = Defensive code problem
O = Offensive code problem
T = Legacy tools problem
C = Legacy code problem

Original-Caja	dOTc	Secure Linux/Windows
Cajita	DOT	Secure microkernel OS

Don’t try this at home (or at all)

D = Defensive code problem
O = Offensive code problem
T = Legacy tools problem
C = Legacy code problem

~~Original Caja~~	~~dOTc~~	~~Secure Linux/Windows~~
Cajita	DOT	Secure microkernel OS

Separate Solutions

D = Defensive code problem
O = Offensive code problem
T = Legacy tools problem
C = Legacy code problem

Cajita	DOT	Secure microkernel OS
Valija	OTC	Virtual Machine

Layered Solutions

D = Defensive code problem
O = Offensive code problem
T = Legacy tools problem
C = Legacy code problem
V = Virtualizability problem

	Cajita*	DOT V	Secure microkernel OS
	Valija	OTC	Virtual Machine
+	Valija on Cajita	DOTCV	VMM + policy glue logic

Lessons for SES

D = Defensive code problem
O = Offensive code problem
T = Legacy tools problem
C = Legacy code problem
V = Virtualizability problem

	SES	DOT V	Secure microkernel OS
	~Harmony-strict	OTC	Virtual Machine
+	Safer scripting	DOTCV	VMM + policy glue logic

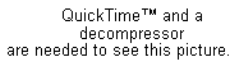
Proposed SES Goals

- SES is smallest secure subset of ~Harmony-strict without loss of functionality.
- SES is a good target for a multiply instantiable embedding of ~Harmony-strict.

	SES	DOT V	Secure microkernel OS
	~Harmony-strict	OTC	Virtual Machine
+	Safer scripting	DOTCV	VMM + policy glue logic

Questions?

Freeze Primordials



Hide Sharp Objects = Cajita

QuickTime™ and a
decompressor
are needed to see this picture.

Cajita + Implementation

QuickTime™ and a
decompressor
are needed to see this picture.

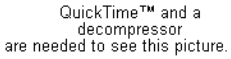
Replace with per-gadget toy knives

QuickTime™ and a
decompressor
are needed to see this picture.

Valija on Cajita Impl

QuickTime™ and a
decompressor
are needed to see this picture.

Valija Impl on Cajita Impl



Jacaranda – language properties

- Statically verifiable subset of bug-fixed ES3.
- Object-capability language with strongly encapsulated objects.
- Methods can use `this`, no need for closure-based encapsulation.
- Strict lexical scoping (including `this`).
- Goals are “DOT$_{CV}$” in MarkM’s terminology
- or “DOT^{CV}” with a refactoring tool.

Specification approach

- Attribute the existing ES3 grammar.
- Attributes are computed bottom-up (see spec introduction for advantages).
- If top-level ‘errors’ attribute is non-empty, reject, otherwise run as ES3 code.
- Attribute rules are an executable specification.
- This approach may be applicable to other Secure ECMAScript proposals, with some adaptation.

Disclaimer

- Can’t possibly cover all details in this presentation.
- There are ***many*** details.
- The spec has detailed rationales for most of them.

Unshadowable names

- Make implicitly imported names (Array, String etc.) and names starting with $ unshadowable.
- Now the spec can assume that specific $ functions are provided by the Jacaranda library.
- Simplifies other restrictions by providing somewhere to stand.

GetValue problem

- obj.foo yields a “Reference”[[obj, obj#foo]]
- [[obj, obj#foo]](x) = obj.foo(x)
- OK so far.
- But var x = [[obj, obj#foo]] gives x = obj#foo.
- Then obj#foo(x) = [[global, obj#foo]](x)

What happened?

- We implicitly threw away information.
- We broke substitutability.
- We broke preservation of authority (reference to global came from nowhere).
- We broke object encapsulation (even when global object is not accessed).

$get

- $get(obj, ‘foo’) = (obj#foo).bind(obj), when obj#foo is a function.
- No loss of expressiveness.
- Can do automated translation of ‘.’ and ‘[]’ to $get (works for programs that weren’t relying on the broken ‘this’ semantics).
- Closures are not memoized (see rationale in spec).
- But still ugly and inefficient (no inlining => function calls are expensive).

Short-term fix (oversimplified)

- Expressions have “classes”.
- Result of property access is 2nd-class.
- GetValue degrades 2nd-class to 3rd-class.
- Can’t call a 3rd-class expression.
- Variables can only hold values of 1st-class expressions.
- Some operations produce 1st-class (or stronger) results regardless of their argument.
- Can chain property accesses or calls without problems.

Words of wisdom from R5RS

"Programming languages should be designed not by piling feature on top of feature, but by removing the weaknesses and restrictions that make additional features appear necessary. Scheme demonstrates that a very small number of rules for forming expressions, with no restrictions on how they are composed, suffice to form a practical and efficient programming language that is flexible enough to support most of the major programming paradigms in use today."

Words of wisdom from R5RS

"Programming languages should be designed not by piling feature on top of feature, but by removing the weaknesses and restrictions that make additional features appear necessary. Scheme demonstrates that a very small number of rules for forming expressions, with no restrictions on how they are composed, suffice to form a practical and efficient programming language that is flexible enough to support most of the major programming paradigms in use today."

- Oops.

Long-term fix

- DependOnNewSemantics = true
- Make ‘.’ and ‘[]’ work like $get.
- Was already proposed for ES4.
- Must be opt-in to avoid breaking existing code.
- Result of property access is 1st-class.
- No other specification changes needed

Support both

- So might as well specify both fixes:
- Short-term: $module(…)
- Long-term: $newmodule(…)

Object encapsulation

- Now ‘this’ behaves sensibly in the subset, and we can use it to create protected objects as in Original-Caja, but without rewriting.
- Properties starting or ending with _ are protected.
- Protected properties can only be *read* via this.
- No properties can be *directly* written.
- No other restrictions on calling functions.

Expressiveness problem

- ‘`this`’ is not in scope in nested functions
- Allow ‘`const thisFoo = this;`’, then allow protected accesses via thisFoo.

Exposed properties

- Problem: can’t allow [] syntax because it might access a non-public property, or it might access a public property that is a function referring to this.
- But some expressions are guaranteed to be numbers.
- Allow foo[numeric_expression].
- Would like to allow ‘+’ expressions, but it also operates on strings.
- Exposed (0^{th}-class) properties are always public, and cannot hold functions that refer to this (can relax that for modules using NewSemantics).

Modules

- ```
  $module(… blah … {
    name: ‘Foo’,
    imports: [‘YAHOO’],
    powerbox: function (powersource, m) {
      …
      m.start();
    },
    start: function() { YAHOO.xyzzy();}
  }…);
  ```
  ```

Modules (continued)

- $makeCaplet(‘Foo’, environment, powersource)
- Jacaranda itself doesn’t say anything about what environment and powersource should be granted.
- It’s sufficient that we can’t instantiate a module with authority that we didn’t have.
- Can build a more sophisticated module system on top of this in Jacaranda code.

Preventing access to globals

- Freely used identifiers must be listed in module imports.
- Then can use ‘with’ to run module code with a given set of imports (depends on ‘with’ being essentially lexically scoped).

Other rules – Lexical

- Limit valid code units to intersection of current implementations.
- Don’t allow Jscript and Venkman extensions using comments.
- Treat /*const*/ and /*fallthru*/ as tokens.
- No \v escape.
- <= 20 significant digits in decimal literals.
- No regexp literals.
- No semicolon insertion.

Other rules – Syntactic

- Const variables cannot be assigned to (can spell ‘`const`’ as ‘`/*const*/ var`’ for compatibility).
- Imports cannot be assigned to.
- Break/continue statements (labelled and unlabelled) must be used correctly.
- No named function expressions (too inconsistent between implementations).
- Some property names are “inaccessible” (similar to ADsafe blacklist) – cannot be accessed or overridden.
- Some identifiers are reserved (mainly for forward compatibility with ES-Harmony).
- Identifiers must be US-ASCII.
- This-variables must be initialised to ‘this’.
- Variables must not be multiply declared in a function body. (Would like to specify block lexical scoping, but not compatible with ES3.)
- Probably missed some – see spec.

ES3F (if time permits)

- Mozilla and ES3.1 errata
- Add ‘const’
- Add ‘useNewSemantics’ to opt-in to new GetValue.
- Unicode 5.1
- Limit implementation-defined behaviour of internal methods on reachable objects.
- [[DefaultValue]] bugfix
- Add Array.prototype and Date methods
- No undefined behaviour
- Make functions as opaque as possible
- Restrictions on extensions (don’t allow extra visible properties on reachable objects).

Is Jacaranda practical?

- Probably too complicated in its current form.
- But demonstrates that an object-capability language subset that includes `this` is possible using only static verification.
- ES3.1/Harmony changes could make it practical.

Dojo Secure

Kris Zyp

sitepen

Dojo Secure

- Full framework for loading, validating, and providing a safe set of library functions and safe access to the DOM.
  - Provides loading registry with different loading mechanisms
  - Uses ADsafe style language constraints
  - Provides |this| within class constructors

http://www.sitepen.com/blog/2008/08/01/secure-mashups-with-dojoxsecure/

Dojo Secure

- Provides access to the DOM (a facade), with the standard API, that is restricted
- Provides a library API (with no namespacing, no need in a global-less environment)
- All on the client side in JavaScript
- Full framework: loading, validation, and DOM sandboxing

ADsafe

- Disables features in JavaScript that prevent containment/sandboxing
  - Global variables
  - [index], this, ==, !=
  - Properties:
    - apply,call,callee,caller,constructor,eval, prototype,this,unwatch,valueOf,watch, and anything starting with _
  - with, eval

http://adsafe.org

Dojo Secure differences from ADsafe

- |this| is allowed in Class method bodies
  - Statically validated
  - Dynamically bound methods
- Names ending with __
  - Due to VBScript usage

Demo/Test Page

- Easy to test validation
- Load pages and scripts

http://www.sitepen.com/labs/code/secure/dojox/secure/tests/load.html

Dojo Secure

JSON, ADsafe, and Misty

Douglas Crockford

Yahoo!

JSON was the first safe subset

It starts with JavaScript array literals and object literals, and removes all behavior, yielding a convenient data format.

http://www.JSON.org/

ADsafe

A system for safe web advertising.

http://www.ADsafe.org/

Static validation only,
no code rewriting.

No impact on performance.

JSLint is an ADsafe validator.

ADsafe

- ADsafe is a JavaScript subset that adds capability discipline by deleting features that cause capability leakage.
- No global variables or functions may be defined.
- No global variables or functions can be accessed except the **`ADSAFE`** object.
- Use of the **`[]`** subscript operator is limited.
- These words cannot be used: **`apply arguments call callee caller constructor eval prototype unwatch valueOf watch`**
- Words starting with _ cannot be used.

Impact on the programming model

- Use of `[ ]` for subscripting is extremely limited. `ADSAFE.get(`name`)` and `ADSAFE.set(`name`,` value`)` must be used instead. This can be annoying.
- `this` cannot be used because it can be made to bind to the global object.
- JavaScript is still quite useable without `this`.

Constructor Recipe

1. Make an object.
    - Object literal
    - **`new`**
    - **`Object.create`**
    - call another constructor

Constructor Recipe

1. Make an object.
   - Object literal, **`new`**, **`Object.create`**, call another constructor
2. Define some variables and functions.
   - These become private members and private methods of the new object.

Constructor Recipe

1. Make an object.
   - Object literal, **`new`**, **`Object.create`**, call another constructor
2. Define some variables and functions.
   - These become private members.
3. Augment the object with privileged methods.

Constructor Recipe

1. Make an object.
   - Object literal, **`new`**, **`Object.create`**, call another constructor
2. Define some variables and functions.
   - These become private members.
3. Augment the object with privileged methods.
4. Return the object.

Step One

```
function myConstructor(x) {
    var that = otherMaker(x);
}
```

Step Two

```
function myConstructor(x) {
    var that = otherMaker(x);
    var secret = f(x);
}
```

Step Three

```
function myConstructor(x) {
    var that = otherMaker(x);
    var secret = f(x);
    that.priv = function () {
        ... secret x ...
    };
}
```

- The methods should use neither **`this`** nor **`that`**.

Step Four

```
function myConstructor(x) {
    var that = otherMaker(x);
    var secret = f(x);
    that.priv = function () {
        ... secret x ...
    };
    return that;
}
```

Objects made with this pattern do not need hardening.

Object tampering does not cause confusion.

ADsafe does not allow access to **`Date`** or **`random`**

This is to allow human evaluation of ad content with confidence that behavior will not change in the future. This is for ad quality and contractual compliance, not for security.

ADsafe DOM Interface

- Light weight.
- JQuery-like.
- Scope of queries is strictly limited to the contents of a the widget's **`<div>`**.
- Guest code cannot get direct access to any DOM node.

Widget Template

```
<div id="ADSAFEID_">
    HTML content goes here.
<script>
"use strict";
ADSAFE.id("ADSAFEID_");
</script>
<script src="approvedlibrary.js"></script>
<script>
"use strict";
ADSAFE.go("ADSAFEID_", function (dom, lib) {
    Application initialization goes here.
});
</script>
</div>
```

Library Template

```
"use strict";
ADSAFE.lib("libraryname", function () {
        Create that library object
    return that;
});
```

- The widget accesses the library object with **`lib.libraryname`**.

ADsafe validation is not destructive, so it can be performed at any and every point in the ad delivery pipeline.

It can even be done after consumer delivery to test compliance.

Multiple points of validation provide greater confidence that bad content will be blocked.

Dangers

- There may still be undiscovered weaknesses in ECMAScript and its many implementations.
- Those implementations are changing, introducing new weaknesses.
- The wrappers must be flawless.
- We are still subject to XSS attacks.

Misty

An experimental object capability language.

Goal: Correct every problem in JavaScript

Reasonable people will disagree
on what the problems actually are.

Misty Objectives

- Make the language easier for beginners.
- Make the language unastonishing and low cruft.
- Make the language an object capability language.

http://www.crockford.com/misty/

Syntax

- **:=** for assignment **=** for identity
- **+** for addition **&** for concatenation
- No semicolons. No blocks.

```
for i to length koda do
    if koda[i].id is null then
        raise 'misshapen'
    fi
od
```

No Global Object

- Each compilation unit is a function body, which gets the capability to return an object that exposes an interface that can be used by other compilation units.
- Compilation units share a vat, so communication is very fast. They can directly invoke methods. They can share object references.

Misty Object Hardening

- The **`fix`** operator produces an immutable reference. The original object is still mutable, but it cannot be changed with the fixed reference.

```
define frozen := fix my_object
```

- **`frozen`** and **`my_object`** are references to the same object, but the **`frozen`** reference is attenuated.

```
my_object.works := true
frozen.works := false # raise 'fix'
```

Fixed References

- A fixed reference cannot be used to modify an object.
- All references obtained with a fixed reference will be fixed.
- This avoids the ICE-9 problem.
- Function values cannot be obtained with a fixed reference. The functions can only be invoked.
- This prevents confusion.

Methods

- A method can obtain a reference to the object of interest with the **$** operator.
- A function that uses **$** can only be called as a method.
- The **$** operator can modify the object even if the object was fixed.

```
$.status := true     # succeeds
struct := $.struct   # struct is fixed
return $             # returns fixed
```

$ could be viewed as a rights amplification, but it is only available to functions that are added to the object before it is fixed.

Simplicity

- Very simple operation. Just **`fix`** references before handing objects to strange code.
- Your own code is not inconvenienced by fixing.
- This level of simplicity is required for successful adoption.