ECMA

ECMA/TC39-TG1/2001/12

Standardizing Information and Communication Systems

Minutes of the 9th ECMA TG1 Meeting held in Mountain View on June 19, 2001

Present:

Bill Gibbons, Jeff Dyer, Herman, Waldemar, Chris Dollin, Andrew Clinick, Eric Lippert (Microsoft)

Agenda:

1. Hellos

2. 20 miniutes on progress/plans

3. Review structure of document

4. Semantic Issues

  - size > 32 bit arrays
  - new vs fresh
  - String : ref or value?
  - Instanceof object
  - Overriding
  - hoisting algorithm
  + const E
  + dynamic arrays

5. July Meeting planning

Lunch @ 12

Coffee @ 3pm

Finish <= 5:30

Progress / Planning

Waldemar does bulk of the writing. The overview section may need to be written by somebody else in parallel. Need others writing and in support roles.

Jeff will work on sections 1-4. Ongoing ownership.

Andrew proposes we don't tackle the Internationalization so we can hope to finish.

Units library, Chris will contribute to this. 23.3, which will cover the optional library part.

Andrew's concerns, reviewing. We haven't been good at doing this in e-mail. We need perhaps to do phone meetings in the interim.

Waldemar, we'll have to meet for additional days.

Next meeting at Netscape in July. Hosting the other working groups as well.

We don't have decimal support. Herman says we agreed to have a language mode, in which literals are interpreted in decimal mode. Alternatively, units can be used to handle decimals.

Perhaps we can enlist Mike Colishaw to write a proposal for a class library that implements decimal.

There is a type called decimal in JScript, which is the CLI version. Needed for backwards compatibility for OLE automation.

Chris will coordinate with Mike to do this work.

July -

Plans, attempt to solve outstanding technical issues. Less document review. Can we commit to resolving all outstanding technical issues? IF we resolve enough of them today, then that might be possible.

More concrete goals for July? Chris proposes that last action of today be a review of accomplishments from today, and see what that implies about July's meeting.

Document structure has been reviewed.

Semantic Issues Discussions

Realities - Microsoft's Design constraints. Semantic constraints.

H: JScript is shipping, backward compatibility, a de facto standard. Can add, cannot materially change it. Pertains to meaning of expressions, type signatures.

Don't want conflicting de facto and de dure standards.

Waldemar's response, cannot satisfy this goal completely. Issues such as instanceof.

H: Some issues can be declared as corner cases. Some cannot.

Ch: We need a mechanism to solve these disputes. Some kind of arbitration mechanism.

H: Instanceof is perhaps a corner case. Arrays aren't flexible.

1. Sizes of arrays, > 32 bits.

W: E3 has weird interpretation of range of literals, > MAXLONG.

Non-negative integral value, >= 2^32. We agree in principal. Proposal needs to be part of the document.

2. New vs. "fresh" objects. Freshly allocated objects. Chris owes us a write up of this issue. Allowing memoization.

H: effectively introduces operator new overriding. Actual syntax issue. 2 issues.

This is a matter of style. Whether it warrants a special syntax for it is debatable.

H: Don't go there. First reaction. Complexity. Verifiability.

Provisionally in, modulo further objections. Chris writes a rationale.

H: Doesn't agree, but won't block it.

Strings: ref or values types? Should null be a member of the type String?

var x : String;

x = null;

Is this allowed? In JScript it is allowed.

Strings are reference types.

W: preferred value types.

Instance Of Object

x instanceof Object

JScript says false unless x is a prototype based object.

This boils down to the definition of the class hierarchy.

```
      Object                              Any
    /      \               *OR*          /   \      \
  RegExp     String                       Object  String  Number
```

JScript uses syntax driven type contexts. If used in a type context, means.

What is the way out of this? Define an operator "is" which codifies the appropriate behavior. Relegate instanceof to "is it a prototype-based object". A semantics for is will be written by Waldemar for July.

We will deprecate the use of instanceof, as it is confusing.

Remove reference containers Boolean/String, etc. Not required.

instanceof and value wrapper types will end up in special appendix B.

H: Does the definition of toObject() get impacted by removing the wrapper types?

+ const E

CLI doesn't support constant element arrays. JScript can't therefore provide them without wrapping them, losing object identity.

```
function f(x : const T[])
{
        x[42] ...
        x[42] ...
}
```

But we can't guarantee that the two references to x[42] will produce the same result. If the underlying array is mutable, side-effects of calls (threads?) could modify it.

Might as well remove immutable arrays.

+ Dynamic Arrays

ES3 arrays - dynamic, extensible, typeless.

JScript arrays, var x : T[]; -- fixed sized arrays.

ES4 extensible arrays? var x : dynamic T[];

How about var x : ArrayOf(T)

new Array[T](42)

So, the proposal now reads:

var x : T[] denotes optional, fixed length arrays (of type T)

var x : Array[T], denotes variable length arrays (of type T)

no type relationship implies.

Parametrized types may end up looking like this proposal. (new Pair[int, String]).

var x : Array[Telephone];   y = new Array(t1, t2);

x = y is illegal.

Topic for next meeting. Pick holes in added value. Please discuss on mailing list.

H: JScript will have parametrized types. This syntax seems to be acceptable.

Bone of contention has now been removed.

Overriding

If a derived class overrides a method in base class, it's name overriding, not name + signature.

H: Peter would like to not have to repeate type annotations, so in principal, a limited form with same argument arity would work.

Example:

```
class A
{
        function foo ( x : Object) { .... }
}

class B extends A
{
        override function foo ( x : Int ) { .... }
}
```

This works with runtime dispatching on the argument types.

JScript supports overloading:

```
class A
{
        function foo ( x : Object) { .... }
        function foo ( x : Spoo) { .... }
}

class B extends A
{
        override function foo ( x : Int ) { .... }
        function foo ( x : String ) { .... }
}
```

(new B).foo(new Spoo); <-- var a : A = new B;

a.foo(new Object); <-- could work, by following superclass chain.

Runtime resolution, plus walking the chain can work.

Don't want runtime vs. compile time to affect the dispatching solution.

But,

a.foo(new String) will differ in which method gets dispatched to in JScript.

The only way out of this quandry, is to only define the non-overloaded case.

Chris: The standard does not define the behavior of functions which are overloaded.

Contravariance allows:

```
class A
{
        function foo ( x : String ) { .... }
}

class B extends A
{
        override function foo ( x ) { .... }
}
```

We could say that an override must have the exact same type signatures or none.

What about optional arguments? Should be matchable.

Bill: Optional arguments seems to require overloading semantics?

Chris: Does JScript have optional arguments now?

H & E: Currently, it doesn't let you define classes with methods with optional arguments.

P: Optional arguments shouldn't be passed positionally, but in a rest args association list. Then it doesn't require overloading logic to introduce them in a overriding method.

What if the override attribute is missing?

H: If method signatures match, JScript assumes an override. If signatures not exact match.

W: The override/mayoverride keyword is required, it is an error if signatures differ (each parameter either not same, or unspecified).

There are narrowing changes of signatures that could break sub-classes. Or clients of the class. Waldemar will write this up.

Enough frippery...

Hoisting algorithm. Action item in email.

Left over items...

- user-defined coerctions?
- pragmas
- default coercions for assignment, parameter passing
- side effects in initializers in a class
- toPrimitive / valueOf
- Numerics, longs, which way coercions go
- Decimals trailing zeroes
- Package details

Give 10 minutes to these topics...

Pragmas

No objections. Minor action, waldemar needs to discuss multiple uses of conflicting pragmas.

Default Coercions

```
var x : T;
x = 42;                    --> x = (implicit coercion T) 42
x = null;
```

x = new Object;

W: Questions about the default coercions. Have issues with it:

```
function f(a : Boolean, b : String) { ... }
f("Hello", true)
```

W: What is the set of valid implicit coercions?

H: An implicit coercion should have no loss of information. Implicit widening of numeric types.

What should E4 define for implicit coercions? Somebody should write down set of implicit coercions.

Herman will write down what JScript currently does.

Side Effects in initializers in a class

```
var y = 0;
class C {
	var x = y++;
}

new C;
```

Waldemar proposes evaluate once semantics for initializer expressions.

```
var y = 0;
class C {
	var x = y++;
	var y = 17;
	static var z = y++;
}
```

Can x see other instance variables? Initializers shouldn't be able to refer to unitialized members.

We are rejecting initialize once. Initialize every time, after superclass constructors are called.

Decimal trailing zeroes

0.000 has no scale. All decimal zeroes are all equally precise?

H: The proposal has problems.

Action item, roll this into proposal on decimal arithmetic.

Package details

H: Recollection, we will in future allow arbitrary code inside a package. Code executes when declaration is reached. But will also have code in classes, static initializers. These will execute lazily. Top level expressions will run before all of the class' static initializers. ASP.NET needs this behavior.

W: Want arbitrary statements to be able to run in the body of classes. Run in declaration order.

To reconcile eagerness, static {...} are given permission to run lazily (not required to).

More issues

+ are instance vars visible in static methods?
+ directive vs. statement
+ hoisting of const, functions

+ generic members
+ float/double comparisons

Are instance variables visible but inaccessible to static methods?

C++ does this. We'll do it that way to in the standard.

directives vs. statements, hoisting of const, functions

E3 says that functions can only be declared at first level of a function. JScript allows them at multiple levels (e.g. in with).

Bill: Let's apply hoisting algorithm, if we have to hoist a const we get an error.

H: Give an error message, avoid silent differing behavior.

W: We can be as strict as we like.

H: Don't want same code with subtle differences (e.g. type annotations) to act differently.

Concerned about user impact. Need to allow declarations in non-overlapping blocks to be allowed.

Float/double comparisons - Float vars with literals...

No float type in the standard.

When we have more context, we'll revisit these items:

- user-defined coerctions?

- toPrimitive / valueOf

July Meeting

Friday, 27th of July

Jeff: Why not write a delta to E3 that is the E4 standard?

Chris: We have the working document, hopefully we can distill a standard from it.

W: Small pile of issues for July. Writing up object model.

H: Please let's get documents ready at least two working days before the next meeting so we have adequate time to prepare.

Meeting closed at 5:02PM.