ECMAScript TC39 TG1 meeting 4th December 2000, hosted by HP

Participants

Waldemar Horwat	Netscape	waldemar@netscape,com
Patrick Beard	Netscape	beard@netscape.com
Roger Lawrence	Netscape	rogerl@netscape.com
Chris Dollin	HP	kers@hplb.hpl.hp.com
Dave Raggett	Openwave	dave.raggett@openwave.com
Herman Venter	Microsoft	hermanv@microsoft.com
Peter Torr	Microsoft	ptorr@microsoft.com
Jeff Dyer	Compiler Company	jeff@compilercompany.com

Next Meeting

10am start - Tuesday, January 16th at Microsoft (Redmond)

Agenda suggestions

- Compact Profile
- Property Lookup
- Unit Syntax
- Operator Overriding
- Package Importing
- Package Definition
- Language Pragmas
- Top-level Predefined Identifiers
- Where we are [going]

Unit Syntax

Waldemar has revised his proposal following comments at the last meeting. The grammar now allows:

```
3in
3 " in "
3 "in * in"
3 "in in"
3 "in^3"
```

The power operator is still there. You can now put white spaces, and you can omit the explicit multiplication symbol.

Herman feels that the use of juxtaposition for multiplication just for units is a little jarring. He can live with the proposal, though.

Chris says the motivation is to follow accepted convention as used in text books.

Waldemar is agnostic.

Resolved: We agree to the new proposal.

Operator Overloading

At the last meeting, Waldemar was asked to looking syntax for operator overloading.

```
function operator "+"(a:C, b:C)
{
}
```

The quotation marks are needed to avoid problems with the grammar, says Waldemar. Herman wants to defer this issue until we have covered other aspects.

Herman is concerned with the number of operators that can be overloaded. In particular, he wants to prevent overloading of === (tripple equals).

Waldemar notes the peculiarities for the tripple equals operator, e.g. NaN === Nan is false. Strings represent another special case. You don't want to give people a chance to distringuish strings with the same length and sequence of characters, since this would prevent implementation optimizations.

Tripple equals is often used to comparison with Null and Undefined.

Patrick asks what if you defined a complex number class. How then would you be able to get the right semantics for tripple equals? He adds, tripple equals is a slippery path.

Waldemar suggests we restrict tripple equals overloading to require both arguments to be in the same class.

Resolved: We agree to allow overloading for tripple equals.

Herman asks for the rational for function call overloading.

Chris responds with the implementation space saving from avoiding the need for full functions.

Herman asks about overloading new and delete.

Waldemar notes that overloading new applies to instances and is kind of like object factories.

Herman is unconvinced.

This remains controversial, and something we should revisit in a month or two.

We turn our attention to overloading delete for array elements. Herman notes that it currently only adds value when you want to alter the prototype chain.

```
"[]"            x = a[17]
"[]="           a[17] = s
"delete[]"      delete a[17]
```

Herman can live with this.

We next discuss overloading the += and -= operators.

No problems.

For Boolean operators, Herman notes that if you want to support SQL, you need a tristate logic. This means that you don't want to force a coercion to boolean. Instead, you need the ability to compare with true or false.

Herman recommends that Waldemar look at how CLI has addressed this issue.

Resolved: We agree to remove :Boolean.

We next discuss overloading the ! operator. Herman notes that this too involves the consideration of whether we want to support tri-state logic as per SQL. CLI has to support SQL. It was not an option. Microsoft will support tr-state logic in their compiler.

```
!(x && y)  vs  !x || !y
```

Chris notes that his implementation assumes the above are always equivalent. He asks if we want ECMAScript to generate tri-state values or instead is it sufficient to accept such values.

Resolved: We will go with the proposal as is, and note the issue in the rationale.

Herman asks if Waldemar would consider using interfaces as the means for overloading `for...in` as is done in CLI.

Waldemar says yes, but we haven't yet agreed on how to support interfaces.

Herman says they support a mechanism to return an enumeration object, you can then enumerate over.

Patrick notes that another possibility would be to exploit multiple return values. Chris lights up with enthusiasm.

We talk through the possibilities. We are unable to reach agreement.

Patrick proposes to write some pseudo code for this. Chris agrees, that this is something to discuss via email.

Waldemar next turns to "super"

```
this.super::m()  vs super.m()
```

He then proposes the following syntax:

```
super x.m
super this.m
- super x
```

He adds that `super super x` would not be supported for security reasons. He goes on to say that "super x" is not allowed. You must use this syntax with an operator that performs virtual dispatch.

Patrick notes that we need to be able to reference the constructor name since unlike C++ we permit constructors to have different names.

Herman would prefer `super.constructor`

Chris writes:

```
class C
{
    constructor alpha ( A )
    {
        this.beta ( A' )
    }

    constructor beta ( A'' )
    {
    }
}
```

This meets with general approval.

we break for lunch ...

Compact Profile

Waldemar proposes that we replace 5.1 and 5.2 by the following, keeping the existing notes as rationale.

5.1 Run-time Compilation

A conforming implementation of ES-CP SHALL support the global built-in object, but is NOT REQUIRED to support the Global.eval() method, see ES3 15.1.2.1.

ES3 sections 15.3.2.1 and 15.3.1.1 define a mechanism for creating Function objects.

An implementation that does not support global eval() SHALL report an EvalError exception whenever the global eval() (ES3 15.1.2.1), Function(p1, p2, ..., pn, body) (ES3 15.3.1.1), or new Function(p1, p2, ..., pn, body) (ES3 15.3.2.1) is called.

.... see paper notes.

Property Lookup

Waldemar has revised the property lookup algorithm to fix certain problems with the previous one. He talks through each of the cases in the table.

Herman wants time to think the proposal over in more detail.

We discuss whether you need to write `this.x` rather than `x`. Herman notes that Microsoft has shipped an implementation which allows you to omit the `this`.

We then discussed the visibility of statics. There is a lot of grumbling, but no change ensures.

Function Overloading

Can we find a way to do this that supports the compiled approach in Microsoft's beta release, and the dynamic approach in the Netscape code base?

One solution would be for the standard to preclude overloading by subclasses, except with an exact signature match.

Patrick suggests we should study (offline) a subset which is compatible with both the dynamic and static approaches to overloading.

Waldemar notes that type coercion and overloading interact in complex ways. He would prefer to avoid dealing with the case where two functions differ only by the types of arguments which are not disjoint, i.e. which have some class-subclass relationship. If any of the arguments are Object, then this is particularly painful.

we take a break ...

Packages

Herman notes that the current proposal doesn't allow for circular references. This makes life harder for programmers. He goes on to describe his view of packages as consisting only of a collection of classes and a way to associate them with a name.

We discuss the order in which classes are initialized. Patrick argues for deferred initialization. Chris notes that people have problems with Java's deferred approach. He would prefer an approach that relates to the lexical order of declarations rather than something based on what gets executed when.

Herman tells us that their implementation allows a package to be spread across multiple files. The package statement effectively reopens the package. He adds that all the files are treated as a single compilation unit.

The need to minimize disk accesses motivates the initialization of all classes in a package in one go.

We search for a compromise, but don't make much progress.

Herman then asks about the imput syntax. He is concerned about the appearance of regular expressions for the use namespace construct.

Waldemar and Chris respond that these are limited to literal regular expressions. This is analogous to the use of wildcards for importing Java packages.

Where are we now?

Herman says hoisting is a big issue for Microsoft. Indexable/Non-indexable (enumerations) is another issue. Herman can't support "weak" as an attribute.

Waldemar agrees to remove "weak".

Other than these issues, Herman is generally happy with the way the spec is evolving.

It would be nice to come up with a simple reflection mechanism. However, this should be something you get only when you need it.

Patrick asks how we come to a closure? We discuss the goals we have for ECMAScript. There seems to be a smaller gap than we previously thought.

Waldemar has a list of issues he would like to discuss in the next meeting, e.g. scope issues.

Patrick asks if we can identify the most contentious issues and move them out into extensions rather than in the standard.

Could we reach an agreement via email on overloading? Microsoft has shipped one approach. Perhaps we should defer this to a later stage when we have covered other more tractable topics.

Herman notes that his implementation is very close to being a subset of Waldemar's spec.

Waldemar explains that his spec is basically a delta document designed for easy reading and review. He see's the standard as being written somewhat differently, and thinks we will be able to re-use significant chunks from the existing standard.

Herman proposes we spend the next two meetings getting the spec to a position where we have a consensus, and that then we work on restructuring the text into the format needed for the standard.

We maintain two documents, one with proposals and the other with the parts that we have frozen and committed to.

Resolved: Waldemar will accept responsibility for maintaining the two documents.