ECMAScript TC39 TG1 meeting 27th September 2000, hosted by HP

ECMA/TC39/00/6

Participants

Andrew Clinick	Microsoft	andrewc@microsoft.com
Clayton Lewis	Netscape	clayton@netscape.com
Waldemar Horwat	Netscape	waldemar@netscape,com
Chris Dollin	HP	kers@hplb.hpl.hp.com
Louis Theran	Nokia	louis.theran@nokia.com
Markuu Vartiainen	Phone.com	mvartia@corp.phone.com
Herman Venter	Microsoft	hermanv@microsoft.com
Patrick Beard	Netscape	beard@netscape.com
Bill Gibbons	Pixo	bgibbons@pixo.com
Sam Ruby	IBM	rubys@us.ibm.com
Jeff Dyer	Compiler Company	jeff@compilercompany.com
Dave Raggett	HP	dsr@w3.org
Roger Lawrence	Netscape	rogerl@netscape.com

Next Meeting

????

Agenda suggestions

- Decimal arithmetic
- WAP subset
- Main topic - E4 proposals

Decimal Arithmetic - Mike Cowlishaw

Mike has put together some new specifications on his web site (insert URL). These include a nearly frozen spec for the concrete representation, a basic functional spec, and an extended spec. This work is based upon the ANSI ??? standard. He has also improved the alignment with IEEE 854 standard, this motivated the development of the extended spec. Mike invites suggestions for ideas for how this should be progressed.

Herman: the open question from the last meeting was whether decimal should be the default number type. We need to determine what would break (if any) if we did this.

Mike: given the legacy of ECMAScript, perhaps it's unwise to make it the default.

Chris: what about switching to decimal when numbers have a fractional part. He wants decimals to be more pervasive than would be the case if you had to explicitly declare numbers as being of the type decimal.

Herman: it is a question of when/how you convert.

Waldemar: thinks it is mainly a question of numeric literals.

Mike: you could make an argument that literals are already decimal, the question is when numbers are converted to binary. Another issue is when binary numbers are passed in from outside.

Herman: if you used a decimal package, then the overloading will fall out. This calls for a switch or mode which determines that literals are to be interpreted as decimals rather than as binary numbers.

Chris: talks about how units could be applied to numeric literals for decimal numbers.

Louis: if the script specifies it's using decimals, you still need to deal with conversion to/from binary.

Herman: if you call an external module and the parameter is untyped, then the code in that module will need to deal with decimal or binary as appropriate. If the parameter is typed, then the value should be converted as needed before passing it to the module.

For small devices without hardware floating point, the size of binary and decimal emulators are roughly equivalent. The problem would be if you have to include both.

Current mobile devices either have no floating point hardware or have just 32 bit floating point. There are some problems with the limited precision of 32 bit floating point, however.

Waldemar: if a=2 and b=2.0 then a==b but a.toString() is not the same as b.toString() if we are using decimal arithmetic.

Chris: this could cause problems if you use toString() and then use the result as a property name.

Mike: does this matter to users though?

Sam: is there a concensus that decimals are in E4?

Herman: We have ruled out a silent switch. There two options open to us:

- it is a named package
- there is a switch (i.e. decimal mode)

The standard does need to cover decimals if we use the package route, the reason being to deal with numeric literals.

Dave: programmers will want an exception if the mode switch to decimal fails.

We agree to the idea of a decimal package in E4 and a mode, provided it's optional. The details of modes will be discussed later. We will need to work on the toString issue.

WAP Subset of ECMA 262

Markku: proposal is to make a subset of ECMA 262 for small devices with limited memory and CPU. The first requirement is to avoid the need for compilation on the device. Currently, WAP compiles scripts on a gateway and the device only deals with a binary encoding.

Markku: proposes to avoid eval and regular expressions.

Louis: how about limiting regular expressions to those which are statically compilable

Waldemar: you should restrict the global object to prevent the dynamic addition of properties to it.

Markku: we have shown that the size of the code for double precision floating point is roughly the same size as that for single precision. We therefore feel we can drop the single precision floating point type (part of WMLScript).

Is there a need for a subset without floating point?

If so then what is the precision of the integers. The expectation is that this would be presumably 32 bit signed integers (the 53 bit proposal in the document is considered inappropriate).

Mike: there are some issues about handling overflow for integers.

Bill: is interested in a strict subset of ECMASCript and is not interested in the integer type. Pixo doesn't have any need for gateways. Bill adds that he agrees with the proposal to avoid eval etc.

Perhaps we can agree to drop "5.5.2 The integer type".

The WAP Forum needs a way to identify the version of a library. Is the work on this for E4 sufficiently advanced?

Bill: What is the most important part of versioning, the pragma or the means to ask which version has been loaded.

Markku: probably the means to ask which version has been loaded.

Herman: adding a getVersion() call to libraries would avoid the need to a change to the language.

Waldemar: suggests the use of "import" for loading a designated library version. We have already agreed to use the reserved words where we can.

Louis: the Phone.com proposal doesn't cater for a full version.

The version could be held in the markup, e.g. in the HTML script element.

Sam: a simple fix e.g. based on import wouldn't solve the cross module versioning issues. It seems like that will need the fuller mechanisms we envisage for E4.

There is a general feeling that the E4 versioning mechanism may be stable enough for use. We will try to focus on this.

The WAP folk need to clarify their intent for integer types. Is this really based on a need for integer only implementations?

Another question is whether literal regular expressions would make sense?

n.b. don't forget the need to restrict the global object.

E4 Proposals

Herman has highlited areas of Waldemar's document and proposes to walk these areas.

Namespace

Herman: the namespace mechanism proposed in Waldemar's document is different from that in C#. He would like to use a different name. He could live with "version".

Chris like's "facet".

Herman: wouldn't like to explain "facet" to other people. "facet" is an unknown.

Patrick: the fact that it is an unknown could be an advantage. It would avoid preconceptions.

Dave: XML Schema uses "facet" in a different context which might cause problems given the pervasiveness of XML.

Peter: how about "projection"?

Patrick: version and namespace are too specific and he would like to eliminate them from consideration.

Waldemar: the name whatever we choose will have to be a reserved name, we can't get the grammar to work otherwise.

Herman: we already agreed not to introduce new reserved words. New keywords in unambigous contexts are, of course, okay. Herman can't agree to the proposal if a new reserved name is needed.

Waldemar: the ability to detect that this is in the new version of the language may allow us to work around this. He explains, that namespace attributes creates the problem.

Sam writes up some syntax to provoke discussion:

```
namespace foo;
[namespace] foo;
foo:namespace;
foo::;
```

We don't get a clear answer, so this remains a problem. We discuss the possibility for allowing some reserved words to be used as identifiers in some contexts.

Herman and Chris have found that it is possible to use context enabled keywords with the appropriate syntactic processing going beyond LALR gramars.

Does the standard define a baseline? Both Netscape and Microsoft have their own plans for extensions. Is there a pedantic mode which treats such extensions as errors?

Netscape reserved some words which aren't reserved in JScript.

Microsoft wants 100% backwards compatibility, i.e. for all old programs to continue to work with E4 implementations.

Chris asks Waldemar for an example of a piece of syntax where the introduction of a keyword would cause the language to not be backwards compatible with E3.

Herman says that he hasn't found any such cases.

Sam: Perhaps we can go with keywords and make them deprecated. This is weaker than defining new keywords.

Sam: how about "map" in place of "namespace" ?

Herman: Microsoft will not implement strict mode (see page 41).

Waldemar: has no objection to strict mode being optional, but prefers it being retained in the standard.

Patrick: it has explanatory power but doesn't constrain implementations.

Chris: would like a mode for enforced block scoping, whether this is called strict mode is neither here nor there.

Resolved: strict mode is optional and will be treated as an addendum to the standard.

Herman says that Microsoft uses "package" rather than "internal".

Waldemar explains that Netscape's implementation couldn't use "package" and introduced "internal" as a work around.

Herman: we could also accept "internal", and perhaps that would be best for users.

Immutability

We then turned to "const". Assigments to constant data should cause an exception to be thrown. Herman doesn't want the syntax:

```
const x = 42;
```

Neither does he want:

```
varn a = const [1,3,7]
```

Chris likes the idea and has used the name "immutable" in his own implementation. He suggests that this may be something for after E4.

Herman: I can't give a guarantee of immutability to objects outside of the scripting language.

Chris: how about throwing an exception if the object doesn't support immutability?

One idea is to always copy on reference

Waldemar writes syntax representing an array of immutable numbers:

```
const Number[]
```

Herman: that could be harder to deal with. He asks Waldemar to flesh out the details. For a compile-time environment it's simple, but we are dealing with run-time declarations here.

Patrick writes for Chris:

```
const {a:1, b:2}
```

In summary, Chris says: if we are going to include "const" in E4 then we are limited to the following syntax:

```
const [ ... ]
const { ... }
```

Annotated Blocks

For example:

```
public {
  var a;
  var b;
}
```

as a shorthand for:

```
public var a;
public var b;
```

Herman doesn't want to support this. He feels that this is bad on usability grounds. It's bad because the attribute isn't closely coupled to each declaration.

We discuss cases such as:

```
private {
  var x;
  public var y;
}
```

and

```
public {
  var x;
  private var y;
}
```

It may be too confusing for one annotation to cancel the outside. Waldemar suggests we could define "public private" and "private public", etc. as errors.

Patrick: how about allowing merging of non-contradictory attributes?

Dave: what about limiting the merging to orthogonal attributes? This would make "public public" an error.

The meaning of "static { ... }"

Microsoft's implementation uses "static { ... }" as a static initializer inside a class, which is executed when the class is **loaded**. By contrast, the Netscape proposal distributes the static attribute among the definitions within the block.

Herman notes that the Microsoft interpretation of "static { ... }" is inconsistent with in style with "public { ... }".

Sam: Microsoft gave the committee advance notice that they were going to implement this feature. We shouldn't now introduce a conflicting semantics for the same syntax.

Chris proposes a prefix as a disambiguator:

```
AB static { ... }
```

where we would need to choose a spelling for "AB".

Bill: I propose we drop both annotated blocks and the Microsoft static block feature. You would still be able to achieve the same effects by placing the annotations directly on each definition.

Dave: if in doubt, leave it out.

Herman: we would still need to avoid Netscape and Microsoft introducing conflicting extensions to the standard.

Waldemar is committed to implementing annotated blocks.

Chris proposes we spell AB as "attributes" and require the presence of this keyword. Can we all live with this?

```
attributes static { ... }
```

Waldemar would prefer to explicitly exclude the use of static as an annotation.

Another possible syntax for annotations is:

```
extend (string) {
    function f() ....
    function g() ....
}
```

Bill argues in favor of making the "attributes" keyword required rather than optional for annotated blocks.

Peter would like to find out how painful it would be for Microsoft to change the syntax for their statically initialized blocks. He is prepared to make this concession if this would move us to an agreement.

Sam: agrees to defer while Peter finds out more and some negotiation continues ...

Scope attributes

Herman dislikes Waldemar's proposal for altering the scope with a block annotation such as "local { ... }".

Dave is confused and writes his preferred syntax, where var y is scoped to the outer block, and the inner x is distinct from the outer x:

```
{
    var x;

    {
        local var x;
        var y;
    }
}
```

and the following where "scope" is needed for the block to define a scope.

```
{
    var x;

    scope {
        local var x;
        var y;
    }
}
```

Is local needed in the second case, i.e. is var y also scoped to the current block in this case?

Herman wants to require the use of an explicit "scope" prefix when the block defines a local scope. He doesn't see the need for the "local" modifier on the declarations.

We discuss "global". Herman doesn't find the following example compelling and cites problems with if and loops:

```
{
    local x = ... something complex ...;
    local y = ... something complex ...;
    global z = f(x,y);
}
```

We have an agreement that we will provide a means to define a local scope, but we don't agree on the use of local and global as scope modifiers.

Netscape wants to make all blocks create new scopes when in strict mode. Microsoft isn't going to implement strict mode.

We can't resolve this until we have discussed conditional compilation (as summarised by Chris).

override & mayOverride

Herman introduced "hide" in his implementation. Waldemar doesn't like it, though. The purpose of override & mayOverride are to help detect spelling errors. Waldemar could live without these if needs be.

Herman explains that "hide" tells the compiler that it's fine for a JScript method to override a method with the same name in an imported base class.

Herman agrees to live with override & mayOverride.

prototype

This hybridizes old and new classes. It causes a function to define its own prototype-based class.

weak

If the only way you can reach some value is via a variable defined as weak, then it is ok to use null for that value. Basically, you are telling the garbage collector that this value can be thrown away if it is only referenced via weak references. The code can always check for null and perhaps recreate the value as appropriate.

We agree that this is valuable!

Perhaps we also want to give folks "finalize" as the other half of the problem? This would open a can of worms says Herman. He has yet to decide on whether he will support "weak".

static & dynamic extents

Herman is concerned that these features will force him to implement dynamic compilation.

```
var b:integer = 1;

function f(c:Boolean):Integer
{
    var a = b;

    if (c)
        var b:Integer = 10;

    return a + b;
}
```

Herman says that in the statement "var a=b", b would be undefined. He sees the conditional declaration as being in the function's scope.

n.b. The original example used "const" rather than "var".

Chris writes:

```
const sys = exists("sys") ? sys : f(...);
```

Do we like the Pascal model? NO. Early version of Pascal referenced the outer variable lexically before the declaration of the inner variable with the same name. This was later recognized as too problematic.

We want to enable static compilation, but not to limit ourselves to a static language.

Patrick notes that a static compilation strategy is possible via generating code for both branches. This may lead to code bloat notes Waldemar.

Waldemar notes that Netscape's implementation treats constants as being accessible only after they are declared.

```
function f(c:Boolean):Integer
{
    var a = b;

    if (c)
        const b:Integer = 10;

    return a + b;
}
```

Here the const b is inaccessible in the first var statement since it occurs lexically before the declaration. This the way it works in the Netscape implementation, but not the way it works in the Microsoft implementation.

If c is a compile time constant then Herman can deal with this. If people want to introduce variables inside conditions when the condition isn't a compile time constant, then they could use a local prefix.

Herman treats the variables declared inside conditional code as being in the function scope. If there are more than one such variable declaration with the same name, he treats this as an error.

One idea floated by Chris is to raise a warning when a conditional variable declaration is encountered.

Waldemar cites writing cross-browser code as a use-case where you want to emulate some method or class if it isn't supported by a particular browser.

Herman asks if we could use namespaces to access the outer variable?

```
if (global::x)
    var x = global::x;

....
```

Chris writes:

```
function sys(x)
{
    if (outer("sys"))
        outer::sys(x);
    else
        ... MY CODE ...
}
```