Minutes of the: ***Ecma TC39-TG1***

held in: ***Phone conference***

on: ***5th December 2006***

Attendees

- Chris Pine, Opera Software
- Dick Sweet, Adobe Systems
- Cormac Flanagan, UC Santa Cruz
- Dave Herman, Northastern University
- Francis Cheng, Adobe Systems
- Graydon Hoare, Mozilla Foundation
- Brendan Eich, Mozilla Foundation
- Edwin Smith, Adobe Systems
- Pratap Lakshman, Microsoft
- Jeff Dyer, Adobe Systems

Agenda

More questions from Cormac and Dave:

- do we have a top-level `Namespace` type in the `intrinsic` namespace or something like that?
  - [brendan] Yes, all standard classes and other global bindings are in `intrinsic` too, for early binding.
- what is `AttributeIdentifier`?
  - [jeff] They are used to refer to XML attributes in E4X. The meaning will not be defined in edition 4.
- why does `IDENT_EXPR` have a `TypeIdentifier` case?
  - [jeff] In expressions like `f.<T>()`, `f.<T>` identifies a parameterised definition and can occur where the other kinds of identifiers occur.
- is the namespace portion of an identifier always resolved statically?
  - [brendan] No; in part for E4X (ECMA-357) compatibility, in part to allow computed names with full generality.
  - [dave] so even for an unqualified name, it could still be looked up dynamically?
- what happens in this example?

```
use namespace foo;
use namespace goo;

   class C {
       foo var x : int = 3
       function f() { print(x) }
   }

   class D extends C {
       goo var x : String = "goo"
   }
```

- [jeff] I assume you meant `D extends C`, in any case `goo::x` is out of scope in `function f`

- what about this example?

```
class C {
     use namespace foo;
     use namespace goo;

     foo var x : int = 3
     function f() { print(x) }
}

class D extends C {
     goo var x : String = "goo"
}
```

  - [jeff] ditto my previous comment
- there's a tricky "up-down" algorithm for method resolution, I think originally invented by Waldemar, that Ed described in Newton. is this captured somewhere?
  - [jeff] Waldemar's description is at http://www.mozilla.org/js/language/js20/core/namespaces.html
  - The wiki spec description is at https://intranet.mozilla.org/ECMA/wiki/doku.php?id=spec:chapter_5_names. With regard to name lookup in a class hierarchy, this description is intended to have the same meaning as the ns version. Note: in the ns proposal, bracket expressions don't capture the open namespaces, only 'public'.
  - Here is an excerpt from the wiki spec:

Unqualified object references

`o[expr]`

This is a reference to a property of the value of the expression `o` that has a name that matches one of the names of the set of names (multiname) composed in the following way:

- Let `id` be the string value of the expression `expr`
- Let `m` be an empty set of names
- For each namespace `q` in the set of open namespaces
  - Let `n` be a name with the qualifier `q` and the identifier `id`
  - Add `n` to the set of names `m`
- Return `m`

The single name of a multiname reference r is determined by the following steps:

- Let `t` be the least derived type of `x` that contains at least one of the names in the multiname set m of the reference `r`
- Let `m'` be the intersection of the set of names `m` and the property names in `t`
- Let `n` be the set of names in the most derived type of `x` and in `m'`
- If `n` is empty, return the name in `m` that is qualified by the public namespace
- If `n` contains one name, then return that name
- Report an ambiguous reference error

The base object of this reference is the value of the expression `o`. [end spec citation]

Here is an examply why top-down resolution protects base classes and clients that use them.

```
lib1.js
class A {
   n1 function f() { return 0 }
}

lib2.js
class B extends A {
   n1 function f() { return 1 }
   n2 function f() { return 2 }
}

client.js
use namespace n1
use namespace n2
function test(a:A) {
   print(a.f()) // 0 or 1, never 2, never ambiguous
}
```

Numbers

We discussed the 6 questions that Dick Sweet asks in the numbers discussion page. Here are the committee's repsonses

1. (nesting use <numbertype>) They should nest. If use int or use uint affects only non-floating numbers, the form of floating numbers should nest. Arithmetic on all numeric types will nonetheless use int (or uint) inside the scope of the pragma. The parsing of floating literals is not affected by the use int pragma.

2. (use rounding on toInt?) The conversion of a decimal to an int should truncate to be consistent with the behavior of double.

3. (negative number toUint) Further investigation is required to answer this.

4. (test understanding of arithmetic on decimal values inside use int pragma) Yes, it behaves as example

5. (default for rounding) Default should be HALF_EVEN, though could perhaps be locale dependent

6. (precision and literals) Needs further investigation to answer

Namespace question

Dave asked whether the following two lines of code represent the same namespace:

```
namespace x = "foo"
namespace y = "foo"
```

Answer: Yes. Namespaces are defined by an internal string and a "kind" (basically a numeric code that represents the context in which the namespace occurs in the code, such as package name versus a user defined namespace).

Use of default in Switch class

Jeff brought up question on the switch_class page about the need for type annotation on the default statement for type switches, and whether the default statement is necessary at all. The discussion engendered good arguments for various options, including dropping the type annotation, dropping the entire default statement, and making the type annotation on the default statement optional. It was agreed that more discussion is required and the issue was placed on the agenda for the next f2f meeting.

Proper tail calls and coercions

Ed brought up the issue of coercions and how they should be handled in the implementation of proper tail calls. Dave suggests deferral of in depth discussion to the f2f meeting because this issue is part of a more general discussion he'd like to have about coercion. Dave also made the point that the main concern with proper tail calls is to keep use of stack space constant, so if there is a way to compress the data about coercions in such a way that multiple calls consume a constant amount of space, then coercions can be carried out with each tail call. Another possibility is to discard all coercions until the final tail call, but Brendan points out that this may have side effects. Dave's main point, though, is that there are a few alternatives to consider and such consideration should be informed by the larger discussion of where and when coercions happen in the language generally.

Use number proposal question

Pratap asks what the motivation for the use number proposal is, and why can't people just use annotations? Brendan answered that people want to use decimal everywhere without having to annotate everything.