TC39 TG 1 (ECMAScript) Meeting, October 2, 2002

Kona, Hawaii

Attendees

Andrew Clinick (Microsoft)
Rok Yu (Microsoft)
Peter Torr (Microsoft)
John Schneider (AgileDelta / BEA)
Waldemar Horwat (Netscape)
Steve Adamski (Netscape)

Agenda

Confidentiality
Upcoming Meetings
Document Status
JScript Numerics

Confidentiality

After the last ECMAScript meeting, an article appeared in the press outlining some details of the XML work being undertaken by TG1. Andrew stressed the importance of keeping the proceedings of the TG confidential; they are not to be broadcast to the public. It is paramount that any 3rd parties invited to TG1 meetings, or any business partners involved in the work outside of ECMA are also aware of the importance of keeping the work confidential.

Upcoming Meetings

The next TG1 meeting will be in Mountain View, CA on November 21-22. The following TG1 meeting will be in Redmond, WA on January 15-16

The next TC39 meeting will be in Cupertino on the morning of March 26, 2003. The TG1 members will meet for ECMAScript in the afternoon of the 26th and the morning of the 27th, and for E4X during the afternoon of the 27th and the morning of the 28th.

Document Status

Rok and Waldemar had prepared a spreadsheet outlining each section of the Edition 4 document, how it related to the Edition 3 document, its current status, and the current owner. The spreadsheet was used to drive discussions about breaking up the work and setting milestones, and a lot of progress was made assigning tasks to individuals, removing duplicate sections, and so forth.

It was also decided to consider cutting user-defined packages from Edition 4. This would remove the ability for ECMAScript developers to define their own packages with the `package` keyword, but they would still be able to utilise host-provided packages written in other languages. If no objections are heard from the list, and if the removal of the feature will not cause a drastic change in the current draft specification, it will be removed at the next meeting.

Rok will send out an updated spreadsheet (including work items and schedule) along with a formalised process for tracking action items and moving them through the approval process. In particular, if a topic is "function-agreed" at one meeting, attendees should become familiar enough with the topic such that it can be promoted to "content-agreed" at the next meeting.

Waldemar will send out directions for accessing his LISP program that is used to produce the draft document, so that other members can incorporate their changes into the source. (If it is deemed too difficult for individuals to update the sources, the raw text will be sent to Waldemar for incorporation).

JScript Numerics

Rok led a discussion on the behaviour of JScript .NET when dealing with numbers. The JScript design was driven by two main factors:

- Maintaining Edition 3 semantics for legacy code
- Producing "expected" results for common operations in new code

This has resulted in semantics that perform well for mainline cases, but are inconsistent when boundary cases are encountered. There are also some discrepancies between early bound and late bound behaviour. Most of these are bug-level defects rather than core design defects, and thus can be rectified in future releases to conform to the standard.

Historically, the 64-bit integer types have been problematic because they can hold values that cannot be accurately represented in a double. Microsoft believes there are two main reasons for using long values in ECMAScript:

- As bit fields or enum values
- As inputs to, or outputs from, host-provided APIs (such as database access or file I/O)

In particular, we do not believe that users intentionally use the long data type in their programs to exploit the full range of 64-bit integers; rather it is a side effect of using an API or enumeration that is defined by the host. For this reason, our main goals with respect to the long data type were to provide silent (lossless) conversions between longs and doubles, and to extend the bitwise operators defined in Edition 3 to work on 64 bit values.

The discussion then turned to how numbers should behave in Edition 4, based on our implementation experience and the long history of discussion within the group.

It was decided that all numeric types that can be exactly represented as doubles should behave as doubles. This includes the signed and unsigned `int`, `byte`, and `short` types. Coercion between all numeric types should be silent where no loss of precision occurs (possibly with a diagnostic warning), whilst type conversion errors should be thrown when loss of precision would occur. Explicit conversion (casting) between types is always allowed and may result in truncation or rounding.

If a `long` or `ulong` value is present in an expression, all operands will be converted to `long` (`ulong`) values if possible. If the conversion is not possible without loss of precision, a `double` operation will be performed instead. Operations on long values may result in values that do not fit into a long; in this case they will overflow to a double result (which may fail on assignment back to a long variable, as outlined above). For division operations, if the denominator is of an integer type, first attempt long division. If the remainder is zero, the answer is kept as a long value. Otherwise, convert the long value to a double and perform double division.

We also discussed the issue of preserving negative zero values when one of the operands was a long value. (This is problematic because it would require all operations to return double results, which would lose precision in some cases). It was decided that, pending expert opinion on the matter, we would only preserve negative zero results if both operands were double. Some informal discussions held with other members of TC39 indicated that negative zero was indeed not needed for typical ECMAScript scenarios.

Finally, it was decided that there would be no explicit support for single-precision (float) arithmetic in the standard. Although ECMAScript will be able to deal with floats provided to and by a host API, internally all floats will be converted to doubles and operated upon as such, being re-converted to floats as needed when being fed back to the host or when assigned to variables of type `float`.