~~Quasi~~ String Templates Specification

TC39 – July 2012

Allen Wirfs-Brock

Mozilla

Quasis now in July ES6 draft

- $identifier substitution dropped, must use ${identifier} instead
  - Lexing/parsing issues
  - Confusing. Which of these is a substitution:

    `I say: $⊳\` // ⊳\ is U+142B

    `I say: $⏃` // ⏃ is U+23C5
- Substitutions may be arbitrary expressions
- Quasis may nest unambiguously within substitution expressions:

  `outer [${`middle (${`inner ${Date()}`})`}]`

  Yields

  “outer [middle (<date>)]”

Untagged and tagged quasi have different expression precedence

- An untagged quasi is a *PrimaryExpression*

  PrimaryExpression :

  …

  QuasiLiteral

- An tagged quasi is a *CallExpression*

  CallExpression :

  …

  CallExpression QuasiLiteral

  – A tagged quasi really is just a special form for a call

  – The tag expression must evaluate to a function

Dealing with Lexing

- Treated similarly to RegExp
- Additional Lex grammar goal symbol *InputElementQuasiTail*
- Add grammar convention for identifying lexical goals

QuasiMiddleList:

[Lexical goal *InputElementQuasiTail*] *QuasiMiddle Expression*
QuasiMiddleList [Lexical goal *InputElementQuasiTail*] *QuasiMiddle Expression*

- Used with Quasi components that start with a }

QuasiSubstitutionTail ::
QuasiMiddle
QuasiTail

QuasiMiddle ::
} *QuasiCharacters*$_{opt}$ **${**

QuasiTail ::
} *QuasiCharacters*$_{opt}$ `

Default Escaping for Untagged Quasis

- Default substitutions for untagged quasi is cooked, not raw. All string escapes are expanded.
- *For any ES string literal that does not contain an unescaped ${, replacing the string delimiters (" or ') with back quote (`) should produce the same string value.*

```
const name = "Allen",  value="0.99";
console.log(`\u261E\t${name}\t\$${value}`);
```

Outputs:

☞ Allen $0.99

- *LineContinuations* are removed for cooked quasi literal text. This is required for consistency with the string replacement principal stated above:

Note however that quasis can still contain literal *LineTerminators* that become part of the string value. EG,

```
`1
2
3`
```

yields the same value as
"1\n2\n3”
(assuming that linefeed is the new line character used in the program text)

The raw Tag

- The raw tag is a property of the String constructor.

```
String.raw`In JavaScript '\n' is a line-feed.`
```

produces the same value as:

```
"In JavaScript '\\n' is a line-feed."
```

Simplified Call-site Object

- Simplified the "call site object" to be an array based upon the assumption that cooked text is the most common use case.

```
//pseudo JavaScript definitionof a call-site object
const ungeussableCallSiteId1234 = do {                    // it would sure be nice to have this in ES
    let CSid = [literalSegment1, literalSegment2, ...];   //... is meta
    CSid.raw = Object.freeze([rawLiteralSegment1, rawLiteralSegment2, ...]);  / /... is meta
    Object.freeze(CSid);
};
```

- Most tag functions will have a signature like:

```
let tag = (lits, ...subs) =>  { ... }
```

and within the body lits can be directly processed as a (frozen) Array. Only if the functions actually uses raw values would it have to do a qualified reference such as lits.raw.

Also not included

- Automatic thunking of substitution arguments
  - A tagged quasi call is basically just a special form for the argument list of a function call. Normal ES functions don't thunk their arguments. It isn't clear that these functions are special enough to change that convention.
- Assignable substitution arguments
  - Similar, this capability isn't available to regular functions.