ECMA/TC39/97/33

Proposal for Conditional Compile Support in ECMAScript

Author: Andrew Clinick
File: ECMA Conditional Compile Support
Last saved: 07/11/97 9:23 AM
Last printed: 14.08.97 10:32~~14.08.97 10:20~~

Section Contents

1.

1. Introduction

1.1 Scenarios

1.1.1 Use new features of JScript

Developer wants to utilize new features in JScript Version 3.0 on a page but maintain compatibility with older versions of Internet Explorer and Netscape Navigator. Currently they would have to write some server side code to serve up a different page dependent on the browser and/or do some client side coding to determine what version of the language is supported. Using conditional compilation, the developer could put the JScript V3 features in a conditional compile section. This code would only be run if the user is running the correct version of JScript, users of older browsers would not see the code and would run the older JScript code.

1.1.2 Debug/Release

Many developers build a slightly different version of code in their development/debug environment compared to the final release code that will be deployed. Conditional compile makes this easier to achieve. The developer could define a debug variable and if true run some debug code (alert variable values for example) otherwise ignore the code.

1.1.3 Development tracing

During development, it can be advantageous to trace where the script code is being executed. Conditional compilation will allow developers to add tracing code to their script and only execute during the development/testing phase.

1.2 Requirements

Must not break any clients, code or content

Minimal impact on performance

Allow hosts to insert constants into the script engine

Compile time computations

1.3 Justification

Provide a useful addition to the language

Provide a graceful method of dealing with different versions of hosts and script engines

2. Feature Description

2.1 @ notation

All constructs and variables provided by conditional compilation have to be preceded by the @ symbol. The @ sign was chosen because it will allow scripts to be parsed by existing c preprocessors. If the # sign was used there could be conflict with existing conditional compile statements in the script

1.1@2.2 cc_on

In order for the script to parse conditional compile instructions, it is necessary to turn on conditional compile. This is achieved by the @cc_on command.

2.2.1 Examples

```
<script language=JScript>
@cc_on
@if @_jscript_version >= 3
   alert ("Jscript version 3");
@else
   alert ("you really need JScript 3 or above in order for this page
to work");
@end
</script>
```

This code when run in any browser that is currently hosting version 3 of Jscript will alert Jscript version 3. On any other browser that does not support JScript version 3, it will alert “you really need IE4 in order for this page to work”.

If @cc_on is not included in the script then all conditional compile statements in comments will be ignored

If the script contains conditional compile statements not in comments, it is assumed that the developer intended to have conditional compile. Therefore, if this is true the engine will honor all conditional compile statements in comments or not

```
<script language=JScript>
@if (@_win32)
document.write("<object id="FileList" border=0");
document.write("classid=");
document.write("clsid:1820FED0-473E-11D0-A96C-00C04FD705A2");
document.write("style="position: absolute; left: 30%; top: 0; ");
document.write(" width: 70%; height: 100%">");
@else
document.write("<embed src= "test.dcr">");
@end
</script>
```

2.3 Variables

Conditional compile allows variables to be created within the conditional compile namespace. The engine will only support numeric and Boolean types. Strings are not supported because of the garbage collection requirements imposed by string handling.

To set the value of a variable the following notation is required:

```
@set @varname = term
```

Term consist of a single constant, conditional compilation variable or parenthesized expression proceeded by zero or more unary operators. Undefined variables are treated as if they were NaN Variables can be used anywhere in the JScript code. For example, you could multiply the value of a conditional compile variable by a standard JScript variable.

2.3.1 Example

```
<script language=JScript>
@set @myvar1=12
@set @myvar2=(@myvar1 * 20)
document.write(@myvar1);
</script>
```

This will write out 240 to the page.

2.4 Constants

The script engine will provide a number of default constants to the preprocessor and an interface to allow the host to add its own constants. Constant can be reassigned a value by the user if required.

2.4.1 Default Constants

@_microsoft = Set to true if running in a Microsoft script engine else not defined

2.4.2 Operating System constants

@_win32 = Set to True if running on Win32 system else not defined

@_win16 = set to True if running on Win16 systems else not defined

@_mac = set to True if running on Macintosh systems else not defined

@_unix = set to True if running on Unix else not defined

2.4.3 Processor constants

@_alpha = set to True if running on an Alpha processor else not defined

@_x86 = set to True if running on an Intel processor else not defined

@_PowerPC = set to True if running on a PowerPC processor else not defined

@_mc680x0 = set to True if running on a 680x0 processor else not defined

@_mips = set to True if running on a MIPS processor else not defined

@_hitachi = set to True if running on an Hitachi processor else not defined

2.4.4 JScript

@_jscript = True

@_jscript_build = build number of JScript engine

@_jscript_version = version of JScript engine (major.minor)

2.5 If (….) elif (….) else (…) end (…)

To provide simple conditional logic, if and else will be provided. The syntax requires that every if be closed by an end. For example:

```
@if (@varname)
   code
@Else
   code
@End
```

To assist with multiple if blocks and the ensuing ends there is an @elif statement is available. This will perform the same function as an else and if without having to add an end statement. For example:

```
@if (@varname = 1)
   code
@else @if (@varname= 2)
   code
@end
@end
```

2 @end statements are required to close the 2 @if blocks.

@elif alternative:

```
@if (@varname = 1)
   code
@elif (@varname= 2)
   code
@end
```

2.5.1 Example

```
<script language=Jscript>
@if (@_win32)
   document.write ("I'm running Win32")
@elif (@_mac)
   document.write ("I'm a mac")
@elif (@_win16)
   document.write ("I'm running Win16")
@else
   document.write ("Unsupported platform")
@end
</script>
```

2.6 Operators

2.6.1 Unary

All the standard JScript Unary operators (+,-,!,~) are supported.

2.6.2 Binary

All the standard JScript Binary operators (*,/,%,+,-,<<,>>,>>>,<,>,<=,>=,==,!=.===,!==) are supported.

2.7 Commenting code

To allow for backward compatibility with older browsers this notation can be combined with comment tags. For example, `//@if` is acceptable. Code can be enclosed by either a single line comment, //, or within a comment block, /* */. If code is contained in a comment block the developer is required to close off the comment block with a conditional compile comment - @*/. It is necessary to start a conditional compile comment block with a /*@ unless the comment is followed immediately by a conditional compile statement.

2.7.1 Example

```
<script>
//@if (@_win32) then alert ("Win32)
</script>

<script>
/*@if (@_x86) then response.redirect("x86.htm")
@else response.redirect("other.htm")
@end
@*/
```

```
   </script>

   <script>
   // this will not work because the comment
   // was not ended with a @
   /*@if (@_mac) then document.write("Macintosh")
*/
   </script>

   <script>
   /*@ this will work because the comment
      // was started with a @
   @if (@_mac) then document.write("Macintosh")
      // and ended by a @
   @*/
   </script>
```

2.8 Interface to add engine constants

To be defined