JScript Conditional Compilation

-- Draft –

June 20, 2007

Pratap Lakshman
Microsoft Corporation

This draft provides a complete description of JScript's Conditional Compilation Functionality. There are still be a few semantic subtleties to document or correct. The Grammar in Section 7 still needs significant work to make it correct and to properly integrate it with the complete language grammar.

1 Introduction

Conditional compilation allows conditionally incorporating source text into a program depending on the status of various symbols or constant values in the program. It is not a separate pre-processing step; instead, it is done as part of the *lexing* and *parsing* of script text. Conditional compilation statements are identified by the lexer and conditionally incorporated into the source stream for parsing. This facility has been available in JScript since IE4.

2 Conditional compilation statements

Syntactically, conditional compilation statements may occur anywhere in source text. However, if they occur within comments they must lexically follow the character pattern that starts a comment. i.e. they must immediately follow the `/*` or `//` character with no whitespace in-between.

The parser activates conditional compilation when it encounters a `@cc_on` statement, the `@if` statement, or the `@set` statement. Once activated it is in effect for all subsequently parsed source text, including source text from subsequent script blocks, or even from script includes.

A set of pre-declared conditional compilation variables (§2.4) is made available when conditional compilation is enabled. New conditional compilation variables can be declared using the `@set` command. Conditional compilation variables are generally used in conditional compilation statements, but can be referenced anywhere in JScript code, including `eval`. When such variables are referenced at a particular point in the source text, conditional compilation must have been explicitly enabled at a point in the source text that has already been parsed. For e.g. the following code snippet will cause an error while parsing:

```
// error! Need to explicitly enable conditional compilation before
// referencing the var.
document.write(@_jscript_version);
```

The following code snippet will cause a parse error too:

```
// error! Need to explicitly enable conditional compilation before
// referencing the var. The parser will not peek into the literal!
var s = "@cc_on";
document.write(@_jscript_version);
```

Note that conditional compilation variables will not be substituted within string literals either. The following code snippet will print the string “@_jscript_version” instead of the value of that pre-defined variable:

```
@cc_on;
document.write("@_jscript_version"); // no substitution within literals
```

2.1 @cc_on statement

The `@cc_on` statement activates conditional compilation in the scripting engine. It also enables the capability to use conditional compilation statements within comments.

Example:

```
@cc_on
```

Conditional compilation is enabled for all subsequently parsed source text - such source text could come from subsequent script blocks, or even from script includes.

2.2 @set statement

The `@set` statement declares a conditional compilation variable. It takes the following syntactic form:

```
@set @vname = expression
```

Where,

`vname`

    A valid JScript variable name

`expression`

    an optional unary operator followed by a numeric value or the primitive Boolean values `true` and `false`, or a conditional compilation variable, or a parenthesized expression.

The following operators are supported in parenthesized expressions:

```
! ~
* / %
+ -
<< >> >>>
< <= > >=
== != === !==
& ^ |
&& ||
```

Example:

```
@set @foo = 11
@set @bar = (@foo * 10)
@set @foobar = @_jscript_version
```

If a variable is referenced before it has been defined, its value is `NaN`.

Conditional compilation variables are generally used in conditional compilation statements, but can be referenced anywhere in JScript code, including `eval`.

Variables declared using the `@set` statement are added to the global scope.

2.3 @if statement

The `@if` statement conditionally incorporates a group of statements, depending on the value of an expression. It takes the following syntactic form:

```
@if (condition1) text1opt
[@elif (condition2) text2opt]
[@else text3opt]
@end
```

Where,
`condition1`

An expression that can be coerced into a Boolean expression.

`text1`

Optional. Text to be parsed if `condition1` is `true`.

`condition2`

An expression that can be coerced into a Boolean expression.

`text2`

Optional. Text to be parsed if `condition1` is `false` and `condition2` is `true`.

`text3`

Optional. Text to be parsed if both `condition1` and `condition2` are `false`.

Example:

if-else	**if-elif-else**	**if-elif-elif-else**

`@set @val = 1;`
 	
`// if-else`
`@if (@val == 1)`
`  document.write("x");`
`@else`
`  document.write("z");`
`@end`
	`@set @val = 1;`
	
	`// if-elif-else`
	`@if (@val == 1)`
	`  document.write("x");`
	`@elif (@val == 2)`
	`  document.write("y");`
	`@else`
	`  document.write("z");`
	`@end`
		`@set @val = 1;`
		
		`// if-elif-elif-else`
		`@if (@val == 1)`
		`  document.write("x");`
		`@elif (@val == 2)`
		`  document.write("y");`
		`@elif (@val == 3)`
		`  document.write("e");`
		`@else`
		`  document.write("z");`
		`@end`

There can be 0 or more `elif` clauses. However, all `elif` clauses must come before the `else` clause.

2.4 Conditional compilation variables

The following are pre-defined conditional compilation variables:

`@_win32`	Returns **true** if running on a Win32 system, else **NaN**.
`@_win16`	Returns **true** if running on a Win16 system, else **NaN**.
`@_mac`	Returns **true** if running on a Mac, else **NaN**.

`@_alpha`	Returns **true** if running on a DEC Alpha processor, else **NaN**.
`@_x86`	Returns **true** if running on an Intel processor, else **NaN**.
`@_mc680x0`	Returns **true** if running on a Motorola 680x0 processor, else **NaN**.
`@_PowerPC`	Returns **true** if running on a Motorola PowerPC processor, else **NaN**.
`@_jscript`	Always returns **true**.
`@_jscript_build`	The build number of the JScript scripting engine.
`@_jscript_version`	A number representing the JScript version number in major.minor format. IE4 supports JScript 3.x IE5.x supports JScript 5.5 or less IE6 supports JScript 5.6 IE7 supports JScript 5.7

Note that these are not read-only variables, and can be redeclared using the `@set` statement.

3 Activating conditional compilation

Conditional compilation is activated by using the `@cc_on` statement or by directly using an `@if` or `@set` statement. Here is an example of conditionally incorporating code based on the value of a symbol (`@trace`). We will use this as a running example for the rest of this document.

```
// file: ver1.html
<script>
@set @trace = 1;

function foo() {
    @if (@trace == 1)
        document.write("enter foo in JScript" +"<br>");
    @end

    document.write("function logic goes here" +"<br>");

    @if (@trace == 1)
        document.write("exit foo in JScript" +"<br>");
    @end
}

foo();
</script>
```

While this script may fail to compile on other browsers, it will compile fine on version 4 or later of IE; the final form of the script that gets executed is as follows:

```
<script>
function foo() {
        document.write("enter foo in JScript" +"<br>");
```

```
    document.write("function logic goes here" +"<br>");

        document.write("exit foo in JScript" +"<br>");
}

foo();
</script>
```

4 Making code available only to JScript

In order that it may run gracefully in other browsers that do not support these conditional compilation extensions, conditional compilation statements must be put within comments. Browsers that don't understand condition compilation will simply eat away the comments.

But how will JScript know to look into the comments, then? By looking for a specific pattern in the comment. When JScript sees this pattern `/*@` or `//@`, it will treat the following lexeme as either a conditional compilation keyword (`set, cc_on, if, elif, else, end`), or a conditional compilation variable as appropriate; the `/*` or the `//` will not be considered as starting a comment. Similarly when JScript sees a `@*/`, the trailing `*/` will not be considered as ending a comment. The special handling of such patterns needs to be explicitly turned ON using the `@cc_on` statement (which itself is embedded within this pattern).

Here is our running example modified such that it compiles fine on a browser that does not support conditional compilation (the differences from the earlier script are marked in **red**):

```
// file: ver2.html
<script>
/*@cc_on @*/
/*@set @trace = 1; @*/

function foo() {
    /*@if (@trace == 1)
        document.write("enter foo in JScript" +"<br>");
    @end @*/

    document.write("function logic goes here" +"<br>");

    /*@if (@trace == 1)
        document.write("exit foo in JScript" +"<br>");
    @end @*/
}

foo();
</script>
```

The final form of the script that gets compiled by JScript is as follows:

```
<script>
function foo() {
        document.write("enter foo in JScript" +"<br>");

    document.write("function logic goes here" +"<br>");
```

```
        document.write("exit foo in JScript" +"<br>");
}

foo();
</script>
```

On other browsers, the script that gets compiled is as follows:

```
<script>
function foo() {

    document.write("function logic goes here");

}

foo();
</script>
```

Note that we could get the same result using C++ style comments (`//`):

```
<script>
//@cc_on
//@set @trace = 1;

function foo() {
    //@if (@trace == 1) document.write("enter foo in JScript" +"<br>"); @end

    document.write("function logic goes here" +"<br>");

    //@if (@trace == 1) document.write("exit foo in JScript" +"<br>"); @end
}

foo();
</script>
```

5 Introducing an alternate code path

The `@if` statement can be used to introduce non-JScript code paths too through the `@else` clause. Here is our running example updated with a non-JScript code path added (the differences from the earlier script are marked in **red**):

```
// file: ver3.html
<script>
/*@cc_on @*/
/*@set @trace = 1; @*/

function foo() {
    /*@if (@trace == 1)
        document.write("enter foo in JScript" +"<br>");
    /*@else @*/
        document.write("enter foo in non-JScript " +"<br>");
    /*@end @*/

    document.write("function logic goes here" +"<br>");

    /*@if (@trace == 1)
        document.write("exit foo in JScript" +"<br>");
    /*@else @*/
        document.write("exit foo in non-JScript" +"<br>");
    /*@end @*/
}

foo();
</script>
```

Notice that because we have a `@else` clause embedded between the `@if` and `@end`, we have to suitably balance the `/*@` and `@*/`.

The final script that gets compiled by JScript is the same as in the previous version:

```
<script>
function foo() {
        document.write("enter foo in JScript" +"<br>");

    document.write("function logic goes here" +"<br>");

        document.write("exit foo in JScript" +"<br>");
}

foo();
</script>
```

However, other browsers now end up compiling the following:

```
<script>
function foo() {
        document.write("enter foo in non-JScript" +"<br>");

    document.write("function logic goes here" +"<br>");

        document.write("exit foo in non-JScript" +"<br>");
}

foo();
</script>
```

To introduce alternate code paths we must use the `/* */` style of comments (in order to suitably hide/expose source text based on matching a `/*` with its suitable `*/`).

6 Multiway branching

The `@elif` clause affords multi way branching. Here is our running example modified to show a 3-way branch (the differences from the earlier script are marked in **red**):

```
//file: ver4.html
<script>
/*@cc_on @*/
/*@set @trace = 2; @*/

function foo() {
    /*@if (@trace == 1)
        document.write("enter foo in JScript" +"<br>");
    /*@elif (@trace == 2)
        document.write("enter foo in JScript via elif clause" +"<br>");
    /*@else @*/
        document.write("enter foo in non-JScript" +"<br>");
    /*@end @*/
    document.write("function logic goes here" +"<br>");

    /*@if (@trace == 1)
        document.write("exit foo in JScript" +"<br>");
    /*@elif (@trace == 2)
        document.write("exit foo in JScript via elif clause" +"<br>");
    /*@else @*/
        document.write("exit foo in non-JScript " +"<br>");
    /*@end @*/
}

foo();
</script>
```

Again, notice that because we have a `@elif` clause embedded, we have to suitably balance the `/*@` and `@*/`.

Now the final script that gets compiled in JScript is as follows:

```
<script>
function foo() {
        document.write("enter foo in JScript via elif clause" +"<br>");

    document.write("function logic goes here" +"<br>");

        document.write("exit foo in JScript via elif clause" +"<br>");
}

foo();
</script>
```

Other browsers end up compiling the same script as before:

```
<script>
function foo() {
        document.write("enter foo in non-JScript" +"<br>");

    document.write("function logic goes here" +"<br>");

        document.write("exit foo in non-JScript" +"<br>");
}

foo();
</script>
```

7 Grammar

(Note: text in **red bold**, is the actual lexeme we look for when parsing.).

```
ccProgram:
    ccSingleCmd

ccCmd:
    ccSingleCmd (; ccSingleCmd) *

ccSingleCmd:
    ccEnableCmd
    ccSetCmd
    ccIfCmd
    ccCmd
```

```
ccEnableCmd:
    @cc_on

ccSetCmd:
    @set ccDeclaration

ccDeclaration:
    ccSingleDeclaration

ccSingleDeclaration:
    ccIdentifier = ccExpression

ccExpression:
    ccPrimaryExpression
    ccPrimaryExpression ccOperator ccPrimaryExpression

ccPrimaryExpression:
    ccBooleanLiteral
    ccIntegerLiteral
    ccVname
    ccParenthesizedExpression

ccParenthesizedExpression:
    ( ccExpression )

ccIfCmd:
    @if ccParenthesizedExpression ccSingleCommand @end
    @if ccParenthesizedExpression ccSingleCommand @else ccSingleCommand @end
    @if ccParenthesizedExpression ccSingleCommand ccElifGroup @else ccSingleCommand @end

ccElifgroup:
    @elif ccParenthesizedExpression ccSingleCommand
    /* empty */

ccVname:
    @ccIdentifier

ccBooleanValue:
    /* same as JScript primitive Boolean values true and false */

ccIntegerLiteral:
    /* same as JScript int literal */

ccIdentifier:
    /* same as JScript identifier */

ccOperator:
    /* same as Jscript operator */
```

8 Revision history

18 June 2007	pratapL	Creation