Control structures

- If statements start their test argument on the same line as the declaration and complex statements should be split into multiple lines.

```plaintext
If ( 2 + 2 = 4;...
good
```

```plaintext
If ( 2 + 2 = 4;...
bad
```

```plaintext
If ( 2 + 2 = 4;
   True;
   // Else
   False
) // each result should be on its own line
good
```

```plaintext
If ( 2 + 2 = 4;
   Let ( [
      ~A = 2;
      ~B = 2
   ];
   ~A + ~B
];
   // Else
   False
) // embedded functions use indenting so the outer function can still be identified
good
```

Note the use of the simple // Else comment. Using the // Else for better readability is suggested. It follows other language styles where else is part of the syntactic structure.

- When boolean operators are used, they should, when possible, precede the line they are connected to.

```plaintext
If ( Evaluate( "Let ( [" & ~contents & "]; False )" ) = "?" // generates an error or
   or ~empty // empty contents
   or ~missingParams; // missing expected parameters
   good
```

```plaintext
If ( Evaluate( "Let ( [" & ~contents & "]; False )" ) = "?" or ~empty or ~missingParams;...
bad
```

- Case statements provide their tests and results on individual lines. Result values are indented from their test. Each new text/result pair should be separated by at least one blank line. Complex calculations can identify their result using a comment on the first line of the result or using a blank line between the test and result.

```plaintext
Case ( 2 + 3 = 4;
      False;

   2 + 2 = 4;
      True;

   False // default result on its own line
)
good
```
Case {
    Let ( {
        ~A = 2;
        ~B = 2
    });
    Let ( $value = ~A + ~B; $value = 4 )
};

    // result 1
    Substitute ( "The result is $VALUE and its $CONDITION";
        [ "$VALUE" ; $value ];
        [ "$CONDITION" ; If ( $value = 4; "GOOD"; "BAD"; )
    );

    2 + 2 = 4;
    // result 2
    True;

    False // default result on its own line
}