APL64: ⎕JS System Function

Contents

Summary 2

JavaScript Datatypes 2

⎕JS Syntax 3

⎕JS Actions 3

EVALUATE 3

Example: Numbers 3

Example: Text 4

Example: Boolean 5

Value Substitution 5

Example: Numbers 6

Example: Text 6

Example: Boolean 6

Example: Array: 7

Example: Nested Array: 7

EXECUTE 8

Example: Define a JavaScript variable: 8

Example: Multi-line Script 8

Example: Create a JavaScript function 9

GETVALUE 10

HASCIRCULARREF 10

Example: Object with a circular reference 10

Example: Object without a circular reference 11

HELP (?) 11

INITJS 12

Example: Second use clears the JavaScript engine state: 12

ISJSFUNCTION 13

Example: 13
SETVALUE 14
Example: Numbers 14
Example: Text 15
Example: Boolean 16
Example: Array 16

Summary

The APL64 ⎕JS system function is an interface to the JavaScript programming language.

Learn more about JavaScript here and here.

⎕JS is cross-platform, so ⎕JS can be used in the APL64 developer GUI, Windows Runtime Executable (WRE) and Cross-platform Component (CPC).

The APL64 interpreter contains all the necessary components of the ⎕JS system function, including the Jint C# interface to JavaScript.

JavaScript Datatypes

JavaScript numeric information is represented as 64-bit floating-point numbers (doubles). JavaScript does not have an Int32 data type. APL numeric information is converted to JavaScript double information, and JavaScript numeric information is converted to APL64 double information.

JavaScript text information is represented as string, not character vectors. APL character data is converted to JavaScript string data, and JavaScript text information is converted to APL64 string data.

JavaScript arrays are analogous to rank-1 APPL arrays, where each element of a JavaScript array can be a scalar or a JavaScript array. This JavaScript design means only APL arrays of rank one or less can be supported by ⎕JS.

The JavaScript object model supports objects containing circular references. ⎕JS actions do not support JavaScript objects with circular references. If a circular reference is encountered during value conversion, the operation will fail with a WS FULL or RECURSION ERROR.

⎕JS Syntax

⎕JS is a monadic system function.

[result] ← ⎕JS action [actionArg1] [ActionArg2] ...

The action is not case sensitive.

The action arguments depend on the selected action.

⎕JS Actions

EVALUATE

Synonyms: EVAL, EV

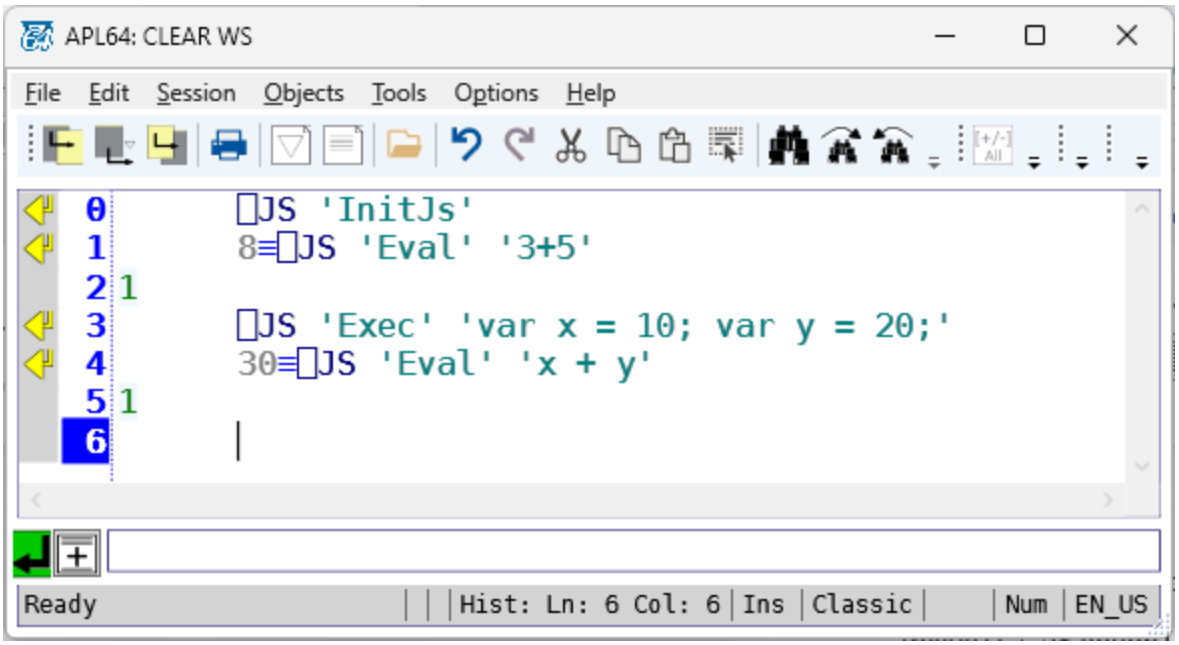
Syntax: aplValue ← ⎕JS 'Evaluate' expression [subValue1] [subValue2]...

expression is the JavaScript expression to evaluate.

aplValue is the result, if any, of the JavaScript evaluation of the expression.

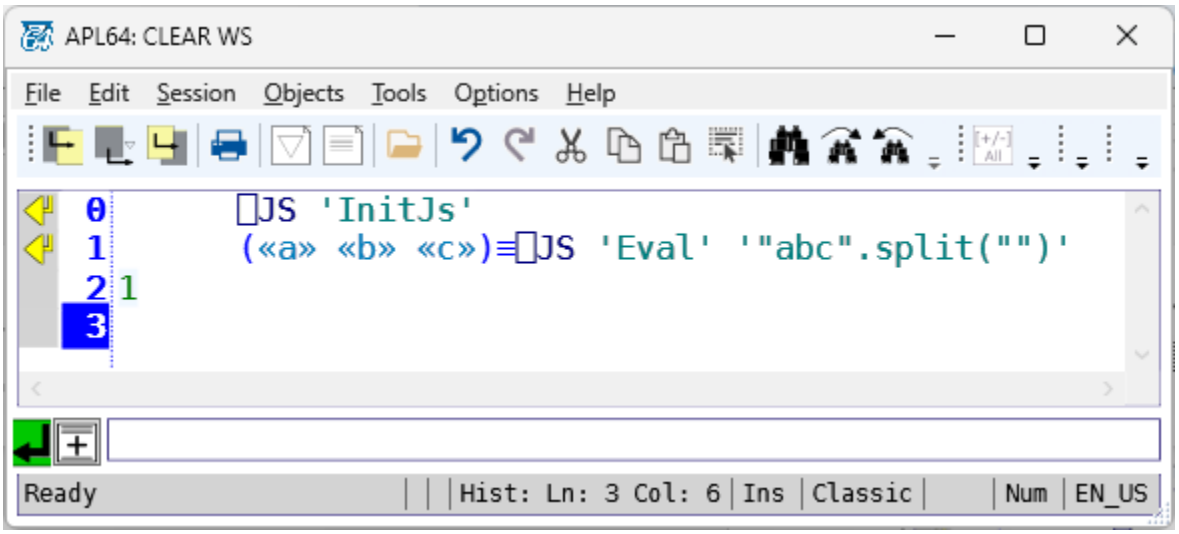
Example: Numbers

```
⎕JS 'InitJs'
8≡⎕JS 'Eval' '3+5'
⎕JS 'Exec' 'var x = 10; var y = 20;'
30≡⎕JS 'Eval' 'x + y'
```



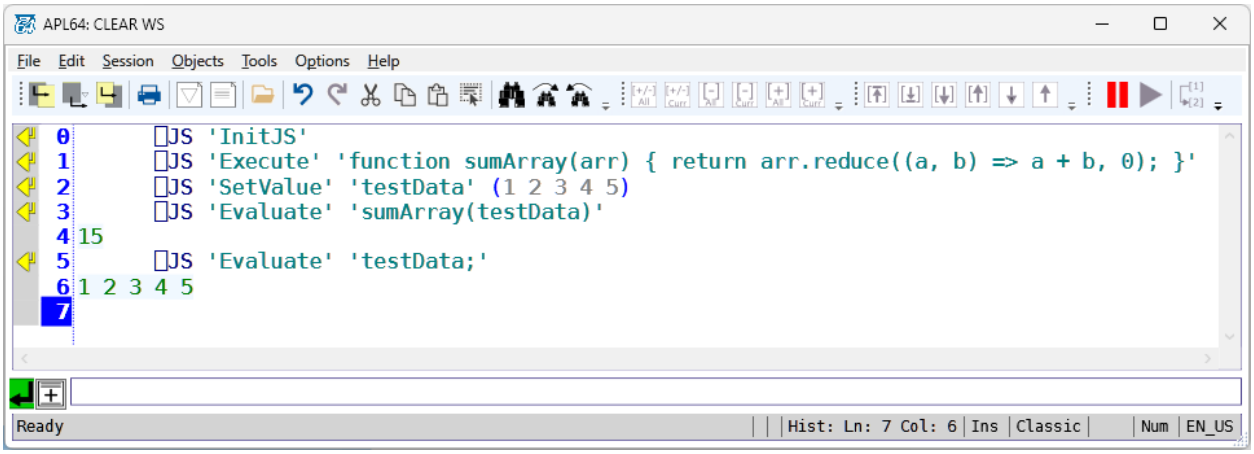
Example: Text

```
⎕JS 'InitJs'
(«a» «b» «c»)≡⎕JS 'Eval' '"abc".split("")'
```



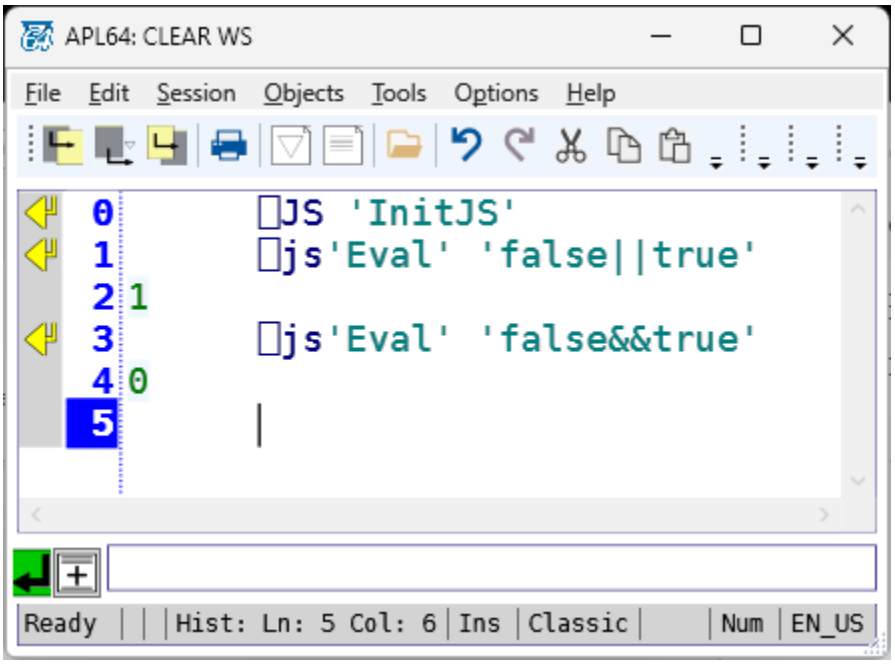
Example: Array

```
⎕JS 'InitJS'
⎕JS 'Execute' 'function sumArray(arr) { return arr.reduce((a, b) => a + b, 0); }'
⎕JS 'SetValue' 'testData' (1 2 3 4 5)
⎕JS 'Evaluate' 'sumArray(testData)'
⎕JS 'Evaluate' 'testData;'
```



Example: Boolean

```
⎕JS 'InitJS'
⎕js'Eval' 'false||true'
⎕js'Eval' 'false&&true'
```

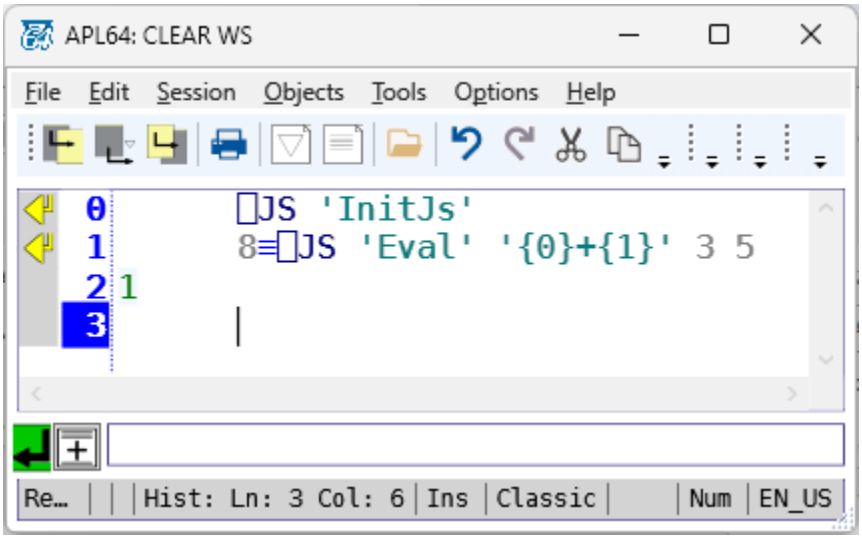


Value Substitution

APL64 values may be substituted into the evaluation expression. The substitution location in the Evaluation expression is indicated by {n}, where n is the index, origin zero, of substitution value. Substitution values may be used more than once. All substitutions are performed prior to evaluation of the evaluation expression.

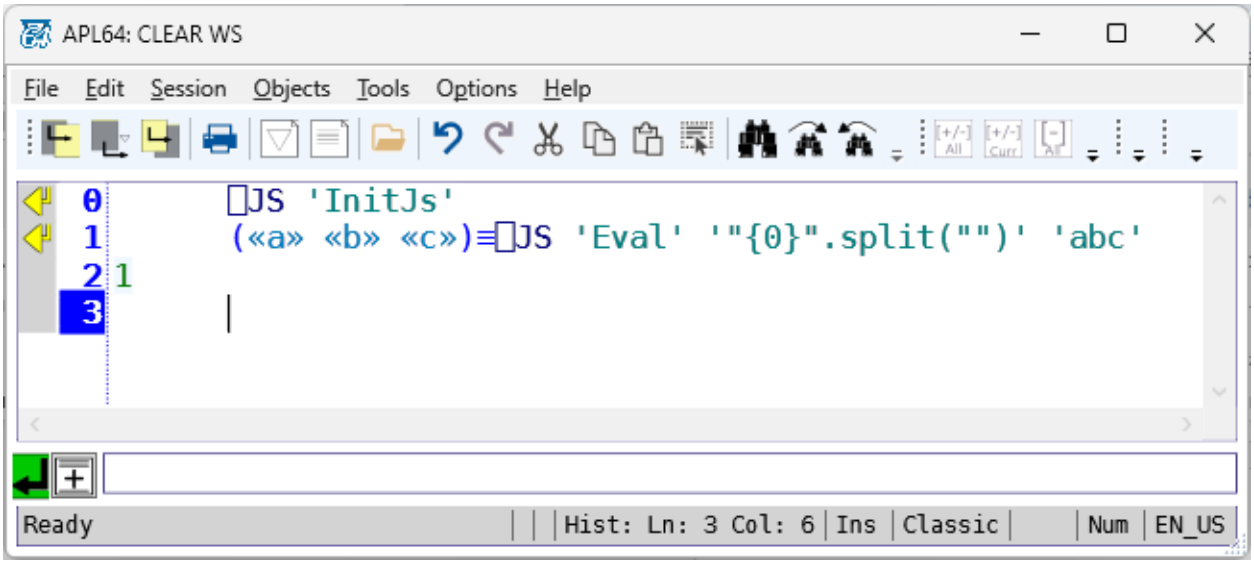
Example: Numbers

```
]JS 'InitJs'
8≡]JS 'Eval' '{0}+{1}' 3 5
```



Example: Text

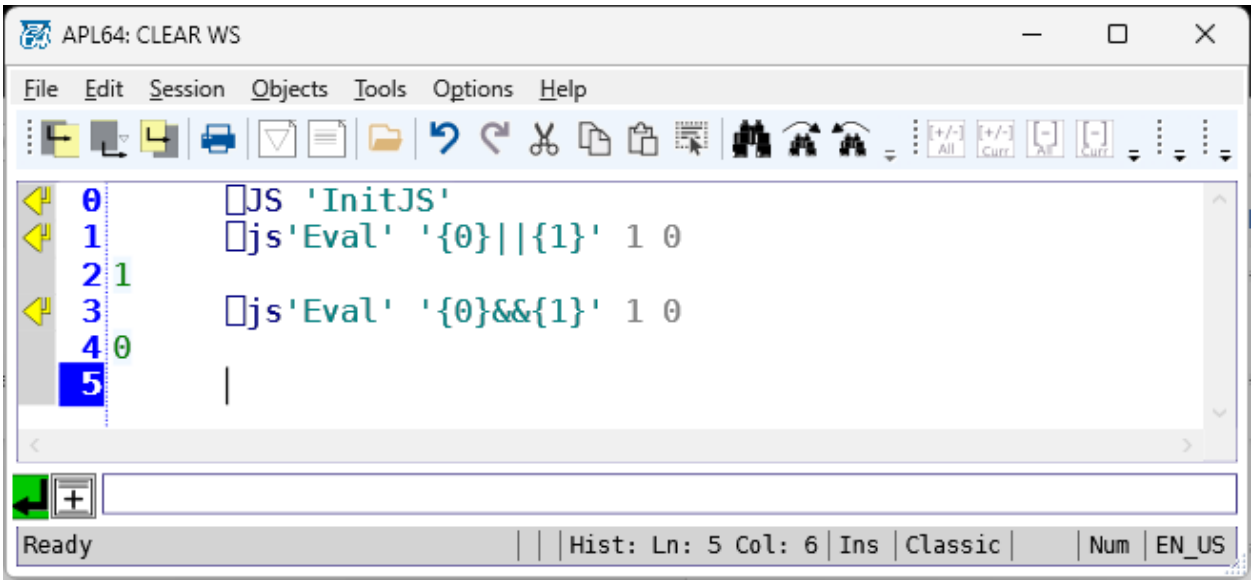
```
]JS 'InitJs'
(«a» «b» «c»)≡]JS 'Eval' '"{0}".split("")' 'abc'
```



Example: Boolean

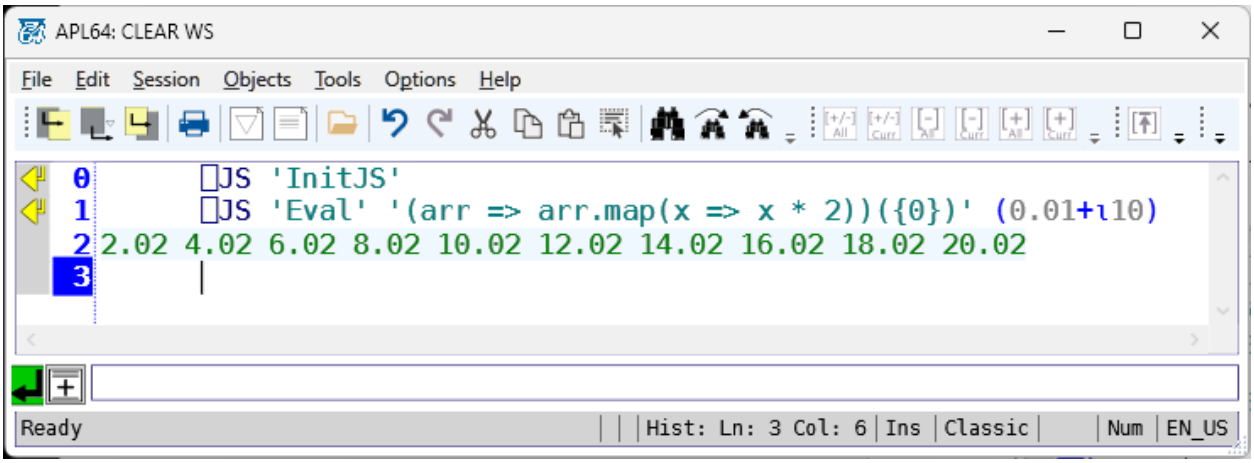
```
]JS 'InitJS'
]js'Eval' '{0}||{1}' 1 0
```

```
]js'Eval' '{0}&&{1}' 1 0
```



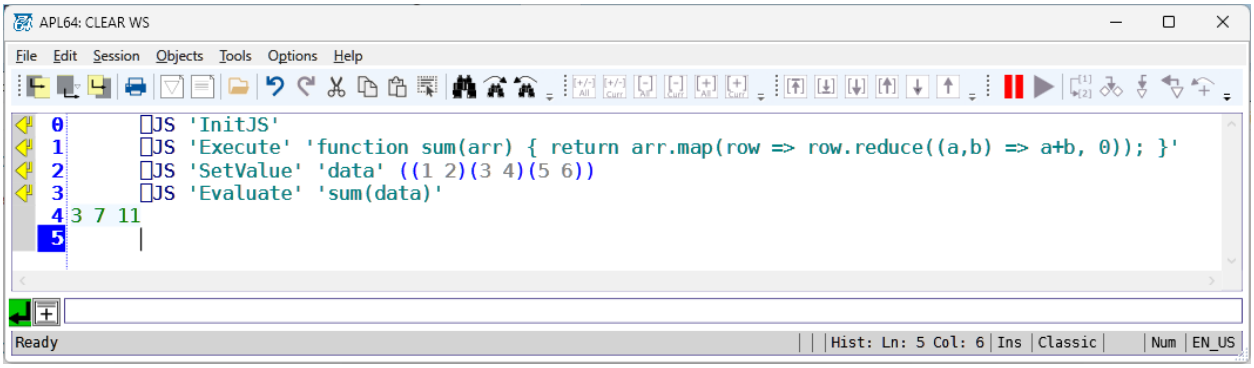
Example: Array:

```
]JS 'InitJS'
]JS 'Eval' '(arr => arr.map(x => x * 2))({0})' (0.01+⍳10)
```



Example: Nested Array:

```
]JS 'InitJS'
]JS 'Execute' 'function sum(arr) { return arr.map(row => row.reduce((a,b) => a+b, 0)); }'
]JS 'SetValue' 'data' ((1 2)(3 4)(5 6))
]JS 'Evaluate' 'sum(data)'
```



EXECUTE

Synonyms: EXEC, EX

Syntax: ⎕JS 'Execute' scriptText

scriptText can be any APL64 text value. Multi-line scripts require new line delimiters. The APL64 multi-line string type («...») can be used within an APL64 programmer-defined function to easily create multi-line scripts.

The ⎕JS 'Execute' action has no result.

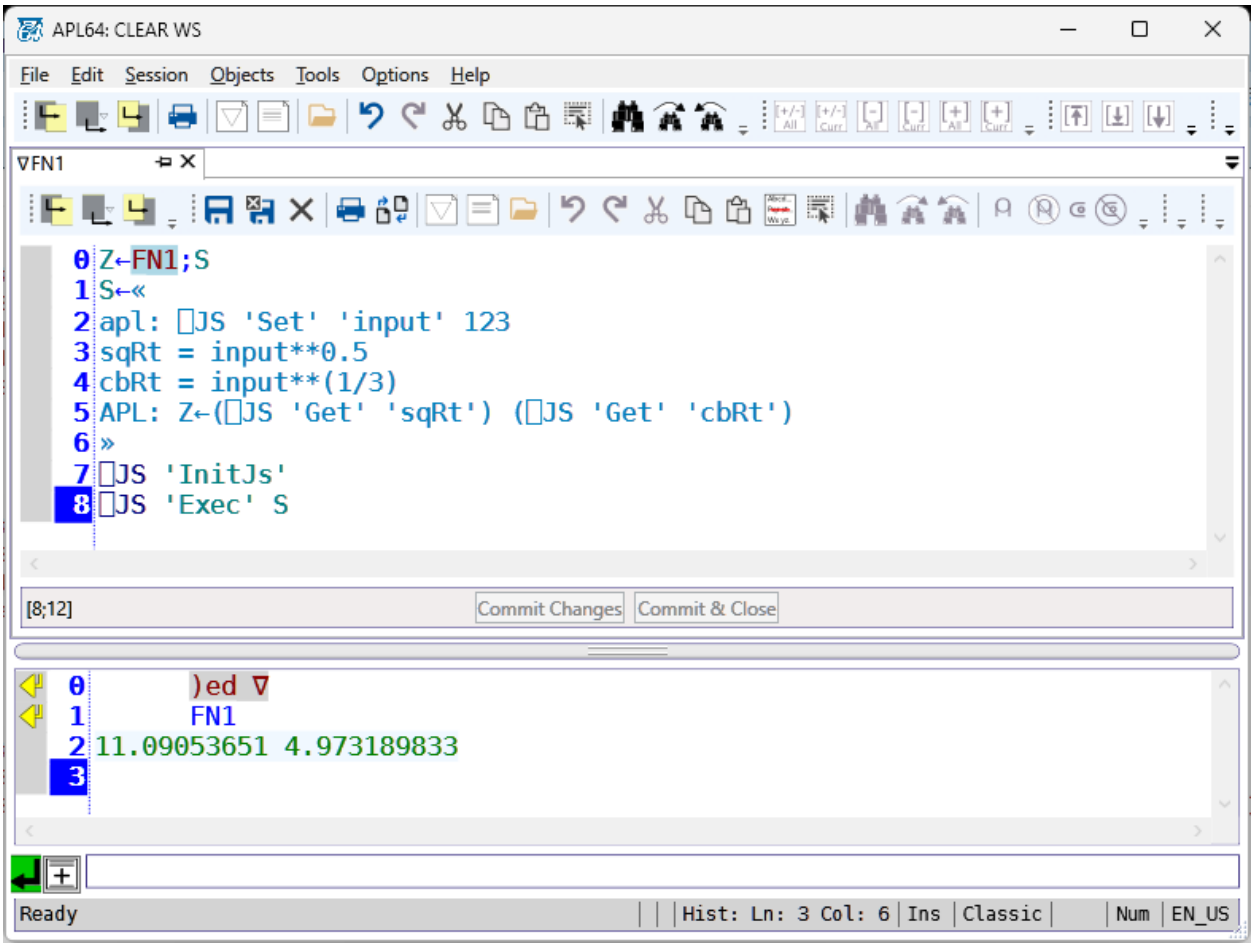
Example: Define a JavaScript variable:

```
⎕JS 'InitJS'
⎕JS 'Execute' 'var x = 10; var y = 20;'
⎕JS 'Evaluate' 'x + y'
⎕JS 'InitJS'
⎕JS 'Execute' 'var x = 10; var y = 20;'
⎕JS 'Evaluate' 'x + y'
```

Example: Multi-line Script

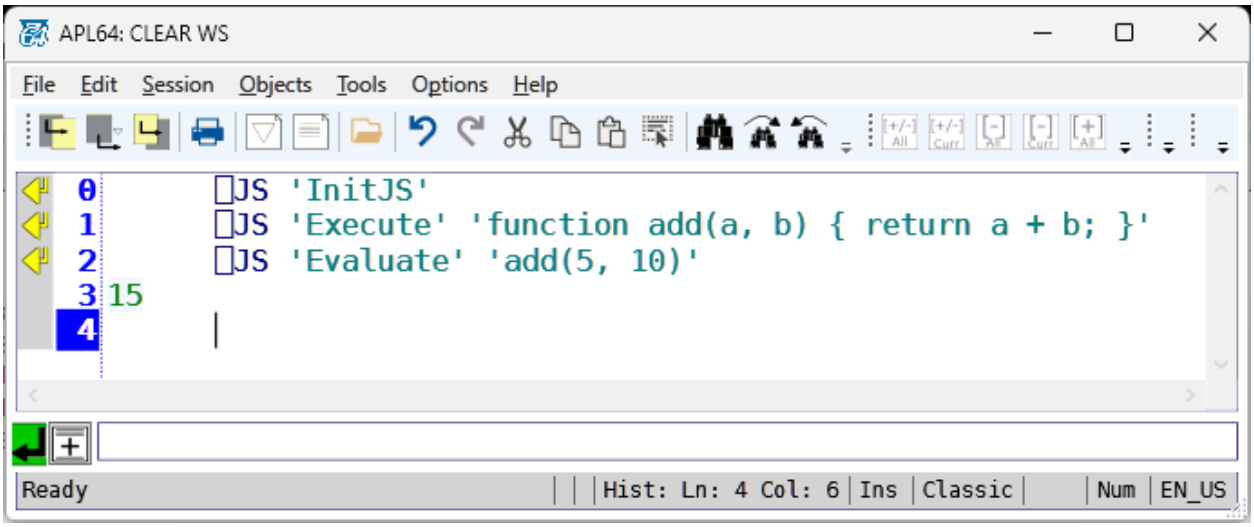
Create, define and run this function:

```
Z←FN1;S
S←«
apl: ⎕JS 'Set' 'input' 123
sqRt = input**0.5
cbRt = input**(1/3)
APL: Z←(⎕JS 'Get' 'sqRt') (⎕JS 'Get' 'cbRt')
»
⎕JS 'InitJs'
⎕JS 'Exec' S
```



Example: Create a JavaScript function

```
⎕JS 'InitJS'
⎕JS 'Execute' 'function add(a, b) { return a + b; }'
⎕JS 'Evaluate' 'add(5, 10)'
```



GETVALUE

Synomyms: GETVAL, GET

Syntax: aplValue ← ⎕JS 'GetValue' variableName

jsObjName is the name of the JavaScript object to test.

Examples:

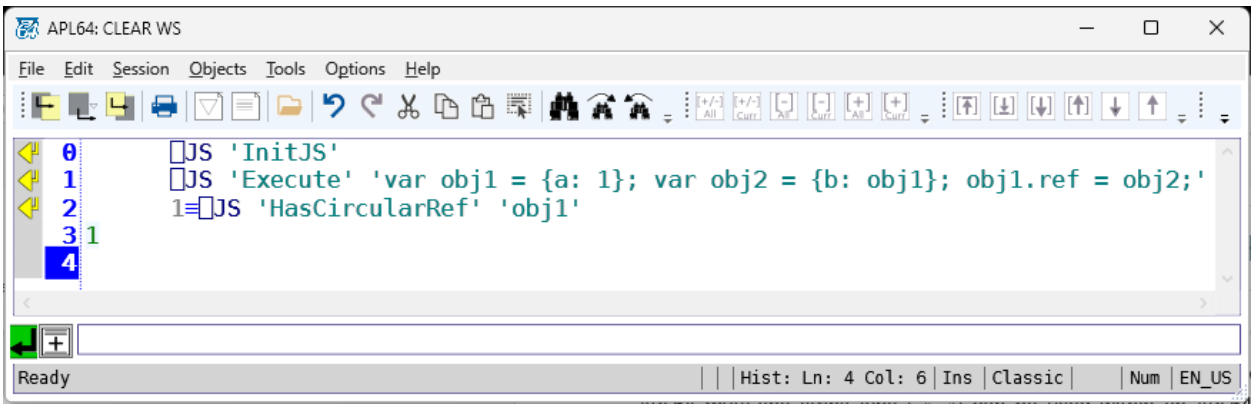
HASCIRCULARREF

Syntax: bool ← ⎕JS 'HasCircularRef' jsObjName

Use this bool ⎕JS to determine if the named JavaScript object includes a circular reference, in which case the other ⎕JS actions cannot be applied to this object. This operation is relatively time consuming.

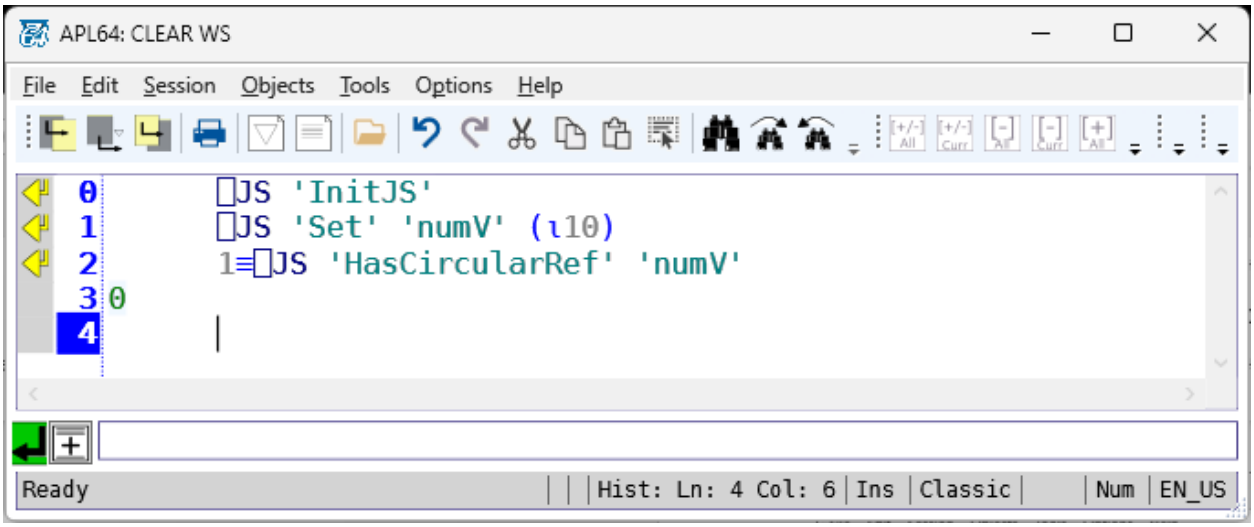
Example: Object with a circular reference

```
⎕JS 'InitJS'
⎕JS 'Execute' 'var obj1 = {a: 1}; var obj2 = {b: obj1}; obj1.ref = obj2;'
1≡⎕JS 'HasCircularRef' 'obj1'
```



Example: Object without a circular reference

```
⎕JS 'InitJS'
⎕JS 'Set' 'numV' (⍳10)
1≡⎕JS 'HasCircularRef' 'numV'
```

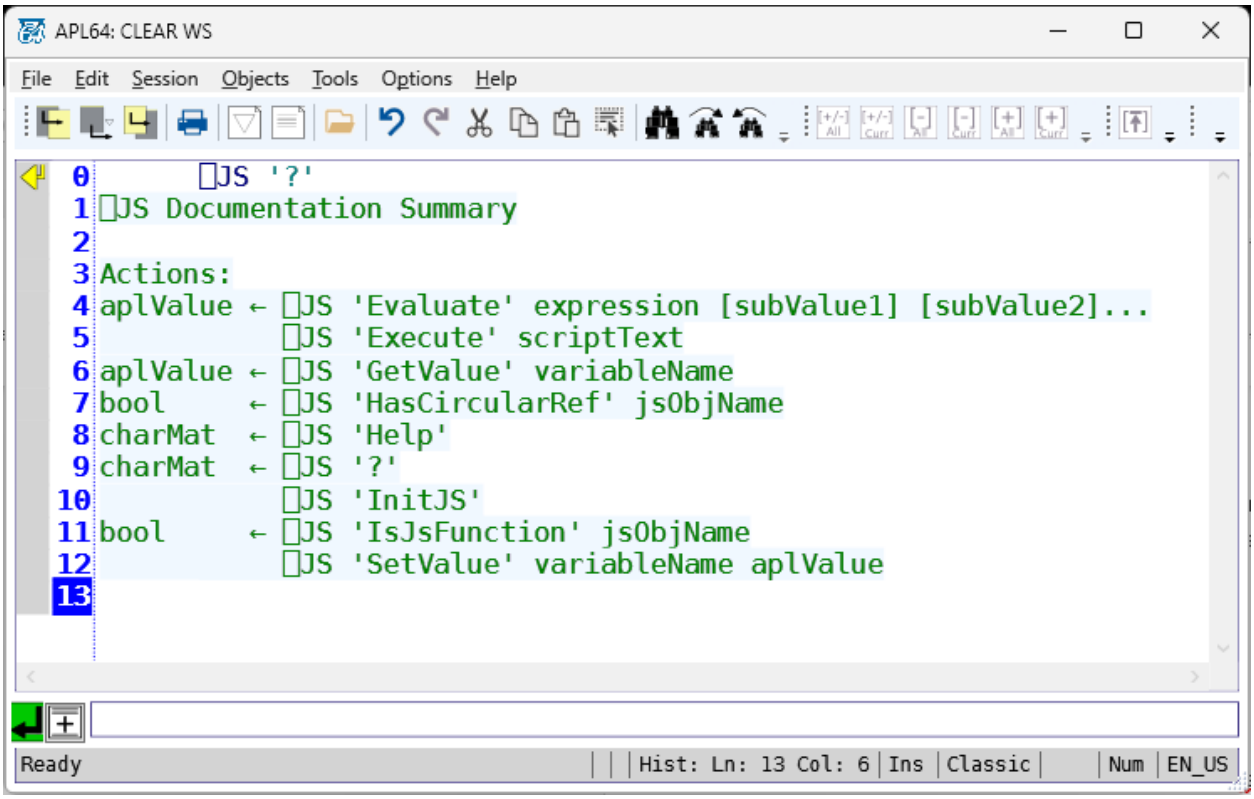


HELP (?)

Syntax:

charMat ← ⎕JS '?'

charMat ← ⎕JS 'Help'



INITJS

Syntax: ⎕JS 'InitJS'

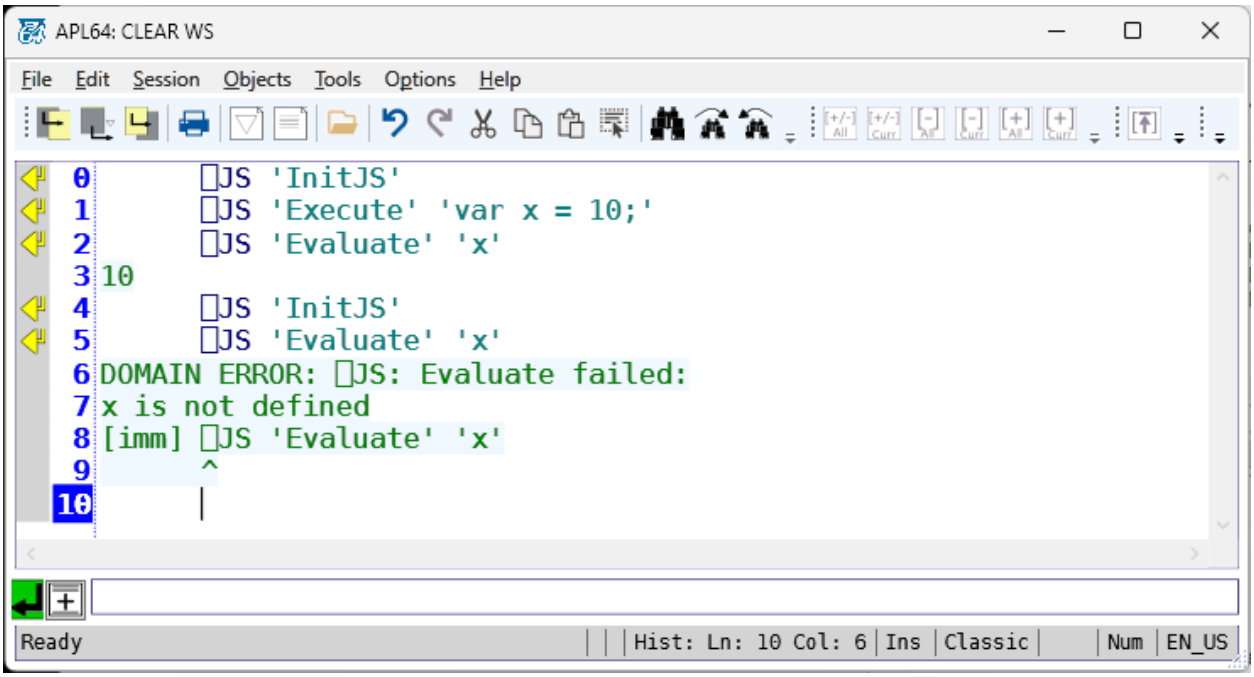
The first time the ⎕JS system function is used in an APL64 instance, the ⎕JS 'InitJS' action must be used to assure that the JavaScript engine instance is available in APL64.

When the ⎕JS 'InitJS' action is used, any pre-existing state of the JavaScript engine in APL64 is cleared and replaced by a new JavaScript engine.

An APL64 JavaScript engine is cleared when the APL64 instance is closed.

Example: Second use clears the JavaScript engine state:

```
⎕JS 'InitJS'
⎕JS 'Execute' 'var x = 10;'
⎕JS 'Evaluate' 'x'
⎕JS 'InitJS'
⎕JS 'Evaluate' 'x'
```



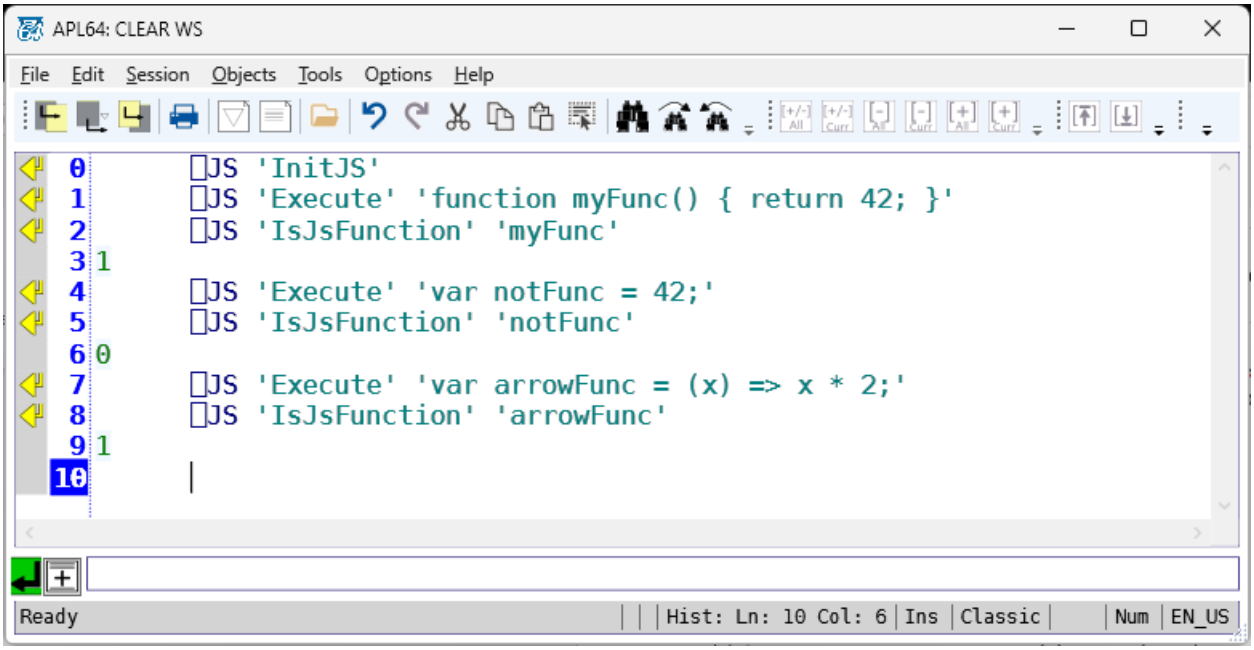
ISJSFUNCTION

Syntax: bool ← ⎕JS 'IsJsFunction' jsObjName

jsObjName is the name of the JavaScript object to test.

Example:

```
⎕JS 'InitJS'
⎕JS 'Execute' 'function myFunc() { return 42; }'
⎕JS 'IsJsFunction' 'myFunc'
⎕JS 'Execute' 'var notFunc = 42;'
⎕JS 'IsJsFunction' 'notFunc'
⎕JS 'Execute' 'var arrowFunc = (x) => x * 2;'
⎕JS 'IsJsFunction' 'arrowFunc'
```



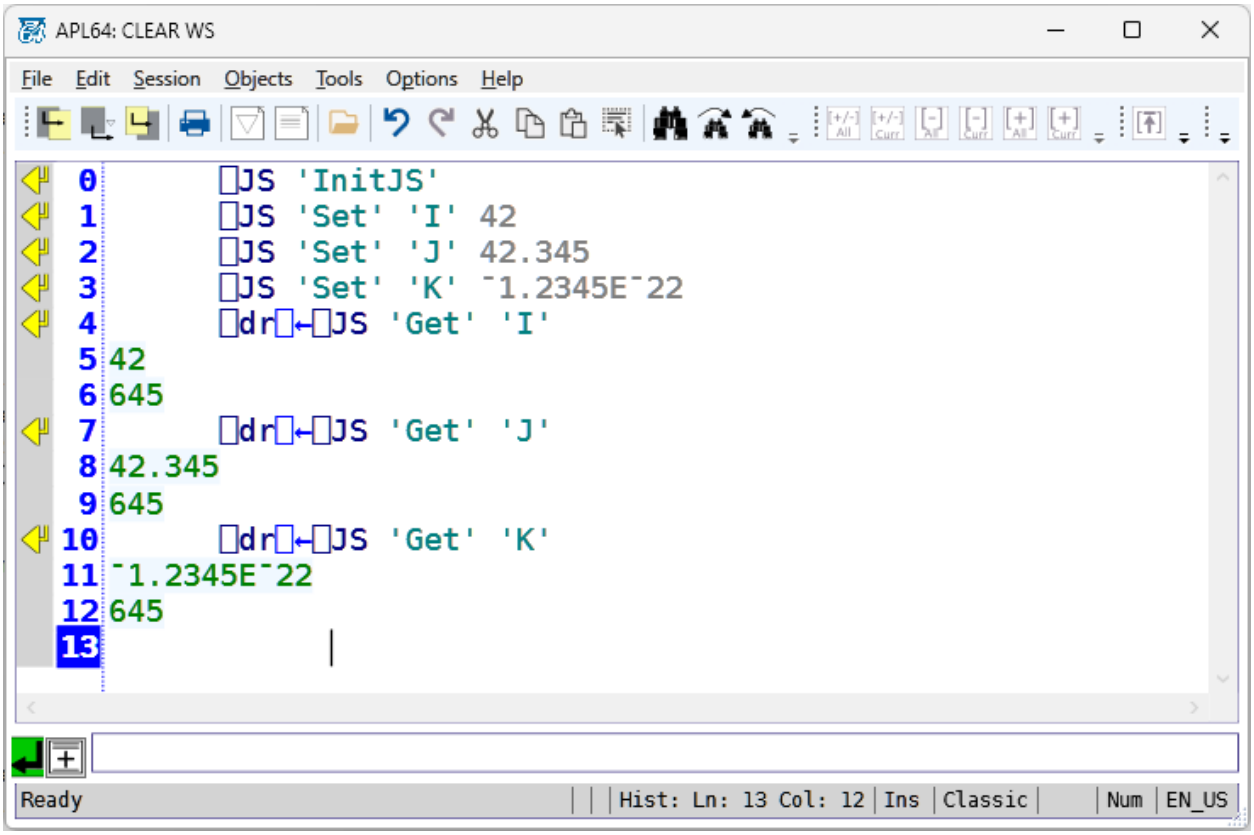
SETVALUE

Synonyms: SETVAL, SET

Syntax: ⎕JS 'SetValue' variableName aplValue

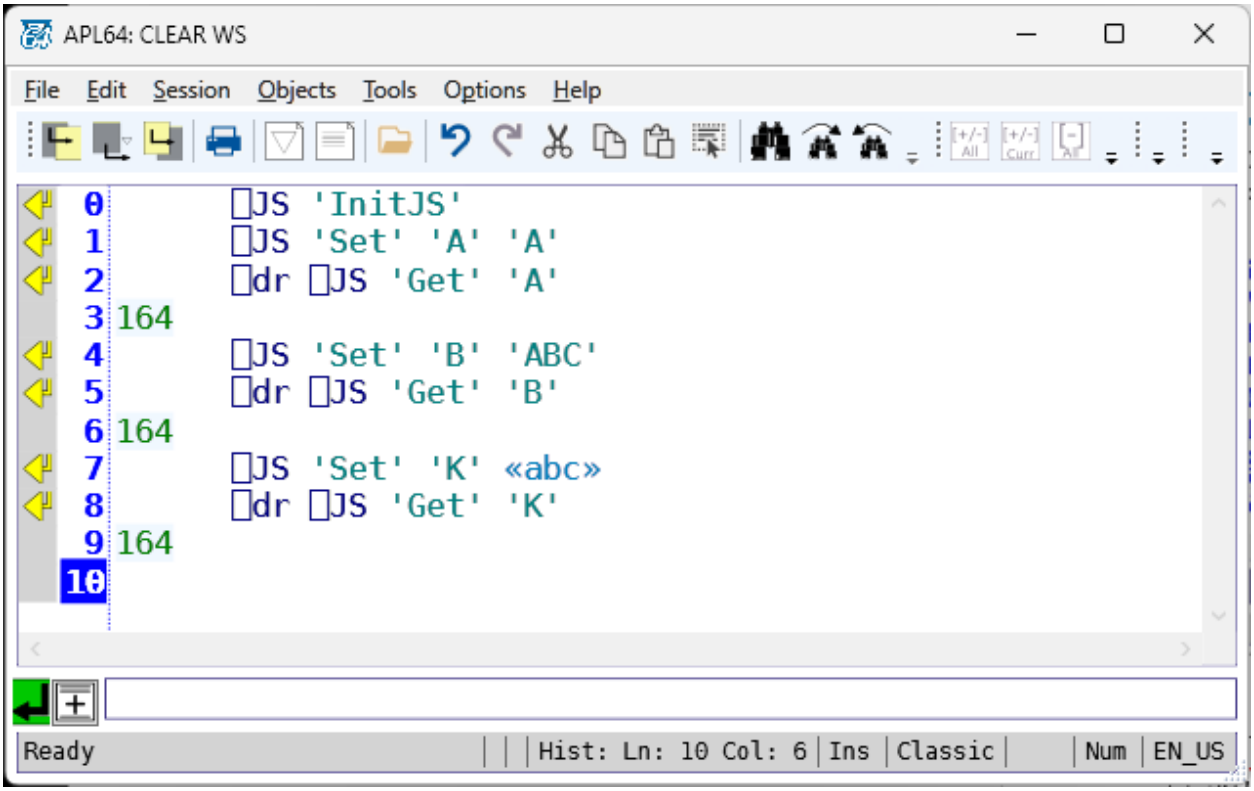
Example: Numbers

```
⎕JS 'InitJS'
⎕JS 'Set' 'I' 42
⎕JS 'Set' 'J' 42.345
⎕JS 'Set' 'K' ¯1.2345E¯22
dr←⎕JS 'Get' 'I'
dr←⎕JS 'Get' 'J'
dr←⎕JS 'Get' 'K'
```



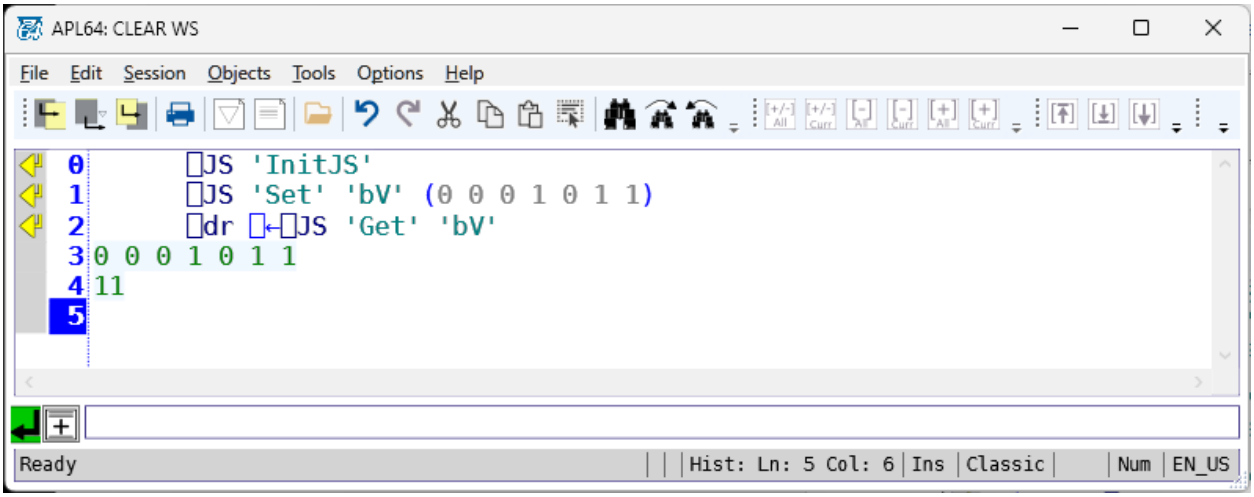
Example: Text

```
⎕JS 'InitJS'
⎕JS 'Set' 'A' 'A'
⎕dr ⎕JS 'Get' 'A'
⎕JS 'Set' 'B' 'ABC'
⎕dr ⎕JS 'Get' 'B'
⎕JS 'Set' 'K' «abc»
⎕dr ⎕JS 'Get' 'K'
```



Example: Boolean

```
⎕JS 'InitJS'
⎕JS 'Set' 'bV' (0 0 0 1 0 1 1)
⎕dr ⎕←⎕JS 'Get' 'bV'
```



Example: Array

```
⎕JS 'InitJS'
⎕JS 'Set' 'dV' (0.01×⍳10)
```

```
      ⎕JS 'Eval' 'dV.map(x => x + 100)'
```