APL64 CPC IntelliSense

Contents

Summary

.Net IntelliSense is a form of context-sensitive documentation which is useful to the .Net programmer (end user) of an APL64 CPC (Cross-platform Component) Nuget package.

An APL64 Cross-platform Component (CPC in Nuget package format) should incorporate .Net standard xml-format documentation which can be accessed via the IntelliSense features of Microsoft Visual Studio and other compatible IDEs.

All public functions in an APL64 CPC should provide IntelliSense documentation. The APL64 programmer who creates an APL64 CPC can include applicable public comments in the public functions in the CPC.

When an APL64 CPC is created, using the APL64 CPC creation utility, the applicable public comments in the public functions in the workspace are converted to .Net xml-format IntelliSense documentation available for the end user of that CPC.

Public Comments for CPC IntelliSense Documentation

APL64 programmer-defined, function public comments are used to specify the IntelliSense documentation in a public function in an APL64 CPC. The public comment prefix is not case-sensitive.

Prefix	**Type**	**Syntax**
⍝∇S	Summary	⍝∇S text
⍝∇P	Param	⍝∇P argName:text
⍝∇PC	Param Continuation Line	⍝∇PC text
⍝∇R	Returns	⍝∇R text
⍝∇⍝	Remarks	⍝∇⍝ text

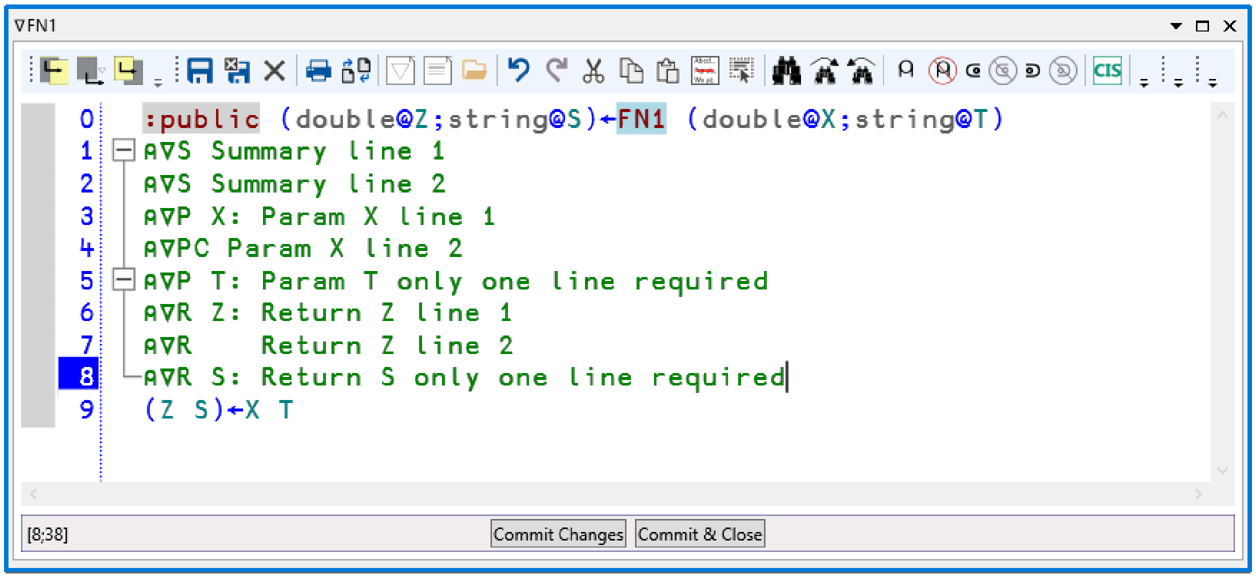
If the summary documentation requires multiple lines of text, each text begins with the Summary public comment prefix, ⍝∇S.

There should be one Param public comment for each right argument of the APL64 public function. The first line of param documentation contains the function argument name, followed by a colon, followed by the documentation text for that line. If a param documentation requires more than one line, the subsequent lines should begin with the Param Continuation Line public comment, ⍝∇PC.

There should be one Returns public comment for each result of the APL64 public function. If a returns documentation requires multiple lines of text, each text line begins with the Returns public comment prefix, ⍝∇R.

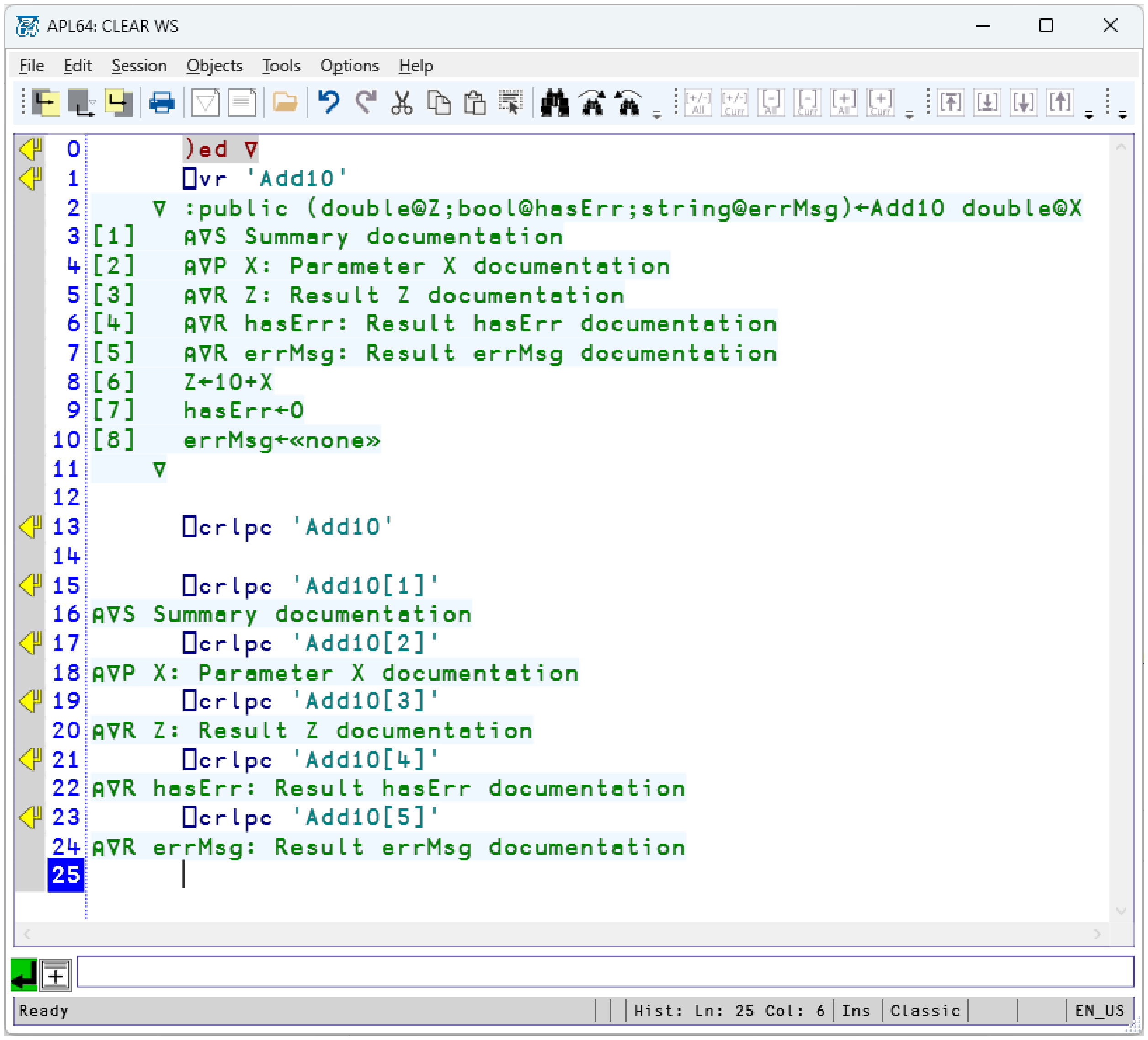
If desired, additional documentation may be included with each line beginning with the Remarks public comment prefix, ⍝∇R.

CPC IntelliSense Public Comment Example



```
:public (double@Z;string@S)←FN1 (double@X;string@T)
⍝∇S Summary line 1
⍝∇S Summary line 2
⍝∇P X: Param X line 1
⍝∇PC Param X line 2
⍝∇P T: Param T only one line required
⍝∇R Z: Return Z line 1
⍝∇R    Return Z line 2
⍝∇R S: Return S only one line required
(Z S)←X T
```

Public comments for CPC IntelliSense documentation are ‘ordinary’ public comments when the public function is not used in an APL64 CPC.



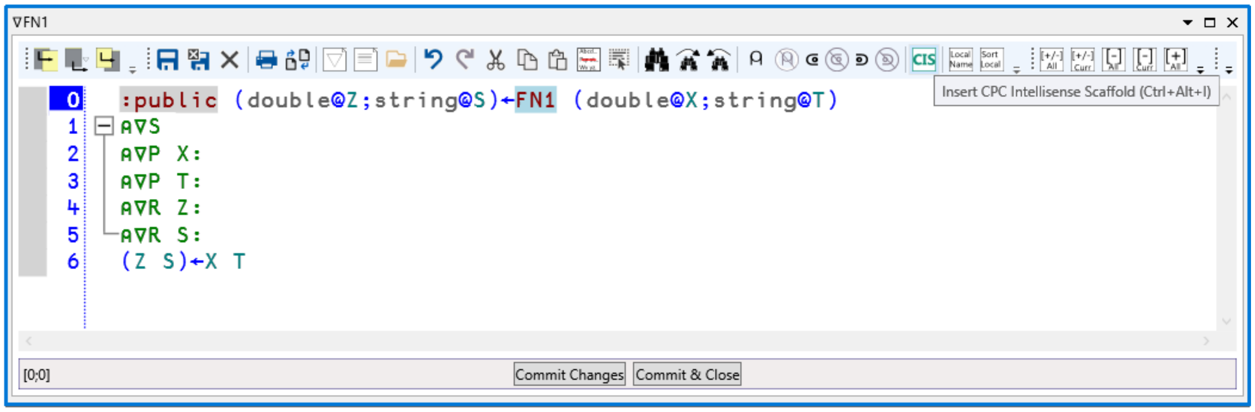
```
      )ed ∇
      ⎕vr 'Add10'
    ∇ :public (double@Z;bool@hasErr;string@errMsg)←Add10 double@X
[1]   ⍝∇S Summary documentation
[2]   ⍝∇P X: Parameter X documentation
[3]   ⍝∇R Z: Result Z documentation
[4]   ⍝∇R hasErr: Result hasErr documentation
[5]   ⍝∇R errMsg: Result errMsg documentation
[6]   Z←10+X
[7]   hasErr←0
[8]   errMsg←«none»
    ∇

      ⎕crlpc 'Add10'

      ⎕crlpc 'Add10[1]'
⍝∇S Summary documentation
      ⎕crlpc 'Add10[2]'
⍝∇P X: Parameter X documentation
      ⎕crlpc 'Add10[3]'
⍝∇R Z: Result Z documentation
      ⎕crlpc 'Add10[4]'
⍝∇R hasErr: Result hasErr documentation
      ⎕crlpc 'Add10[5]'
⍝∇R errMsg: Result errMsg documentation
```

Insert the CPC IntelliSense Scaffold

Use the CIS toolbar button

When a public function is being edited, use the CIS (CPC IntelliSense Scaffold) toolbar button to insert the IntelliSense scaffold into the function.

```
⎕def ':public (double@Z;string@S)←FN1 (double@X;string@T)' '(Z S)←X T'
```

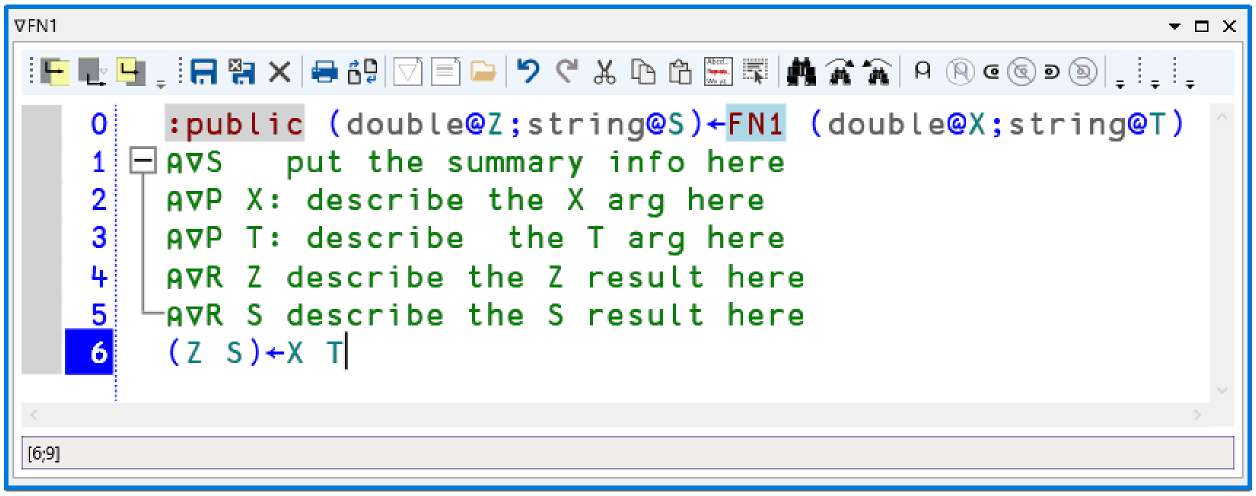


```
:public (double@Z;string@S)←FN1 (double@X;string@T)
⍝∇S
⍝∇P X:
⍝∇P T:
⍝∇R Z:
⍝∇R S:
(Z S)←X T
```

The ⎕CPCIS system function may be used to insert the CPC IntelliSense Scaffold into a public function under program control.

Complete the CPC IntelliSense Documentation

Add the application specific details of the public function arguments, results and summary information into the CPC IntelliSense Scaffold, and save the function to the workspace.



```
:public (double@Z;string@S)←FN1 (double@X;string@T)
⍝∇S   put the summary info here
⍝∇P X: describe the X arg here
⍝∇P T: describe  the T arg here
⍝∇R Z describe the Z result here
⍝∇R S describe the S result here
(Z S)←X T
```

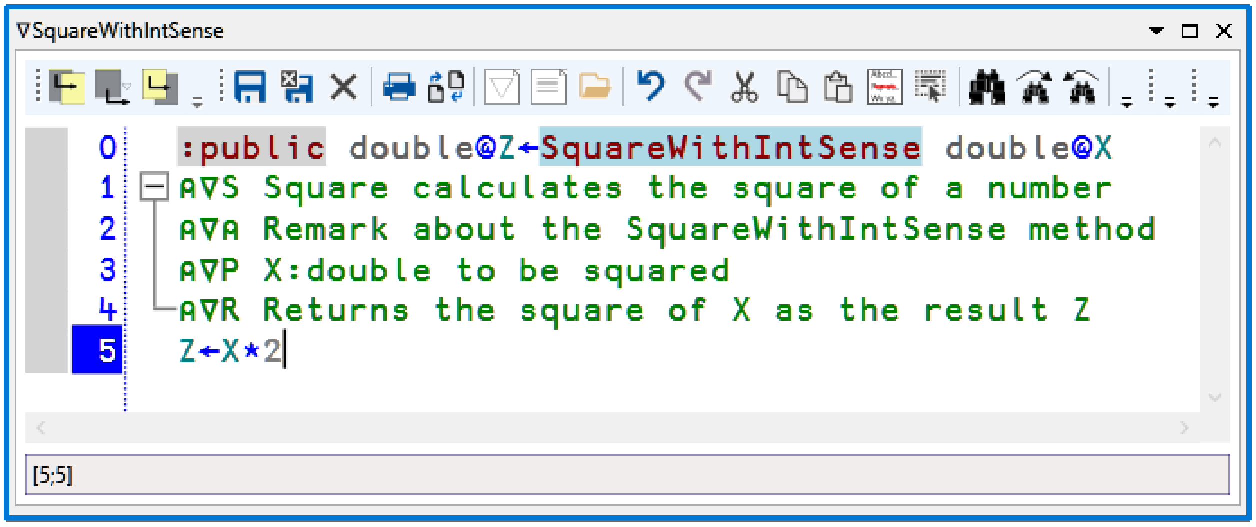
⎕CSR To Check the Result

The APL64 executable statement, 'full' ⎕csr 'publicFunctionName', will return the .Net C# method source code of a public function which will be used in an APL64 CPC.

Example: Single Line IntelliSense Elements

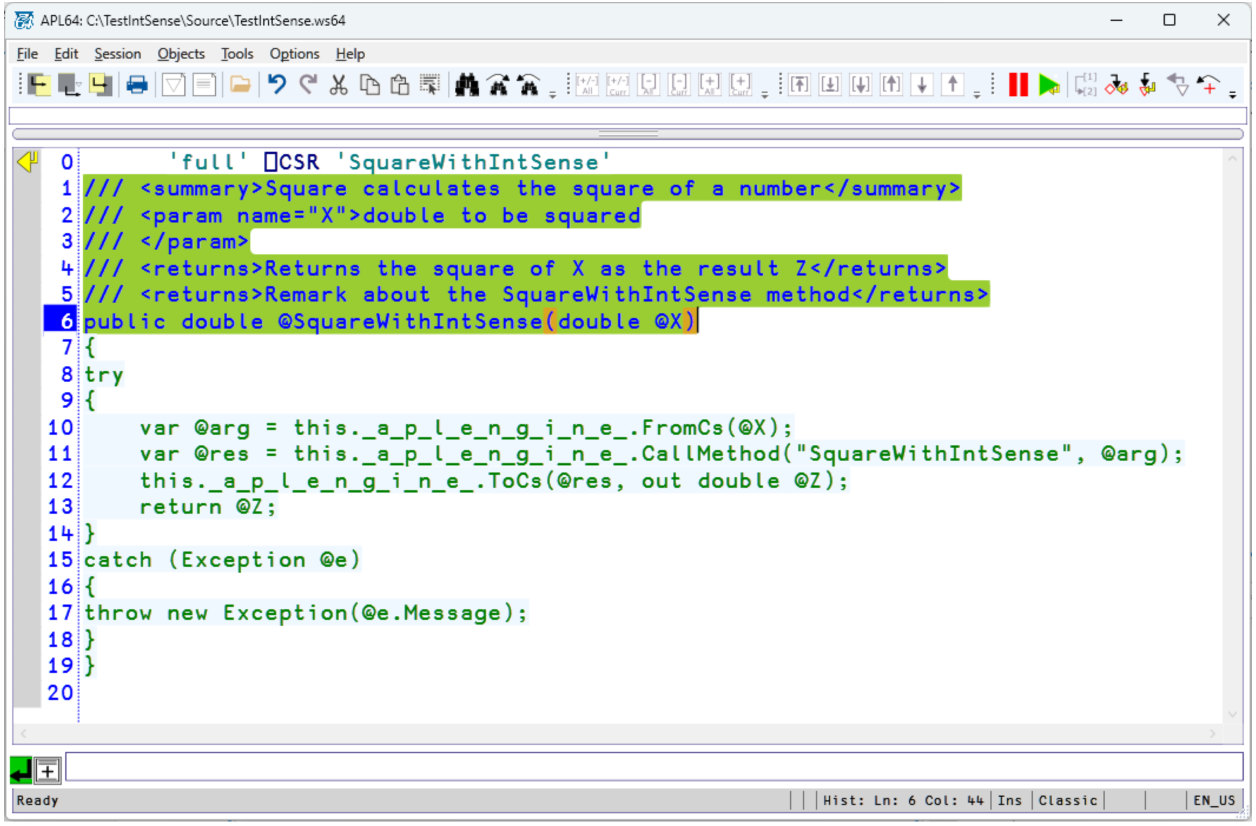
The APL64 programmer-developed public function SquareWitIntSense includes public comments to create .Net IntelliSense when it is used in an APL64 CPC. In this example the

.Net IntelliSense documentation for Summary, Remarks, Param and Returns are each a single line of text.



```
:public double@Z←SquareWithIntSense double@X
⍝∇S Square calculates the square of a number
⍝∇A Remark about the SquareWithIntSense method
⍝∇P X:double to be squared
⍝∇R Returns the square of X as the result Z
Z←X*2
```

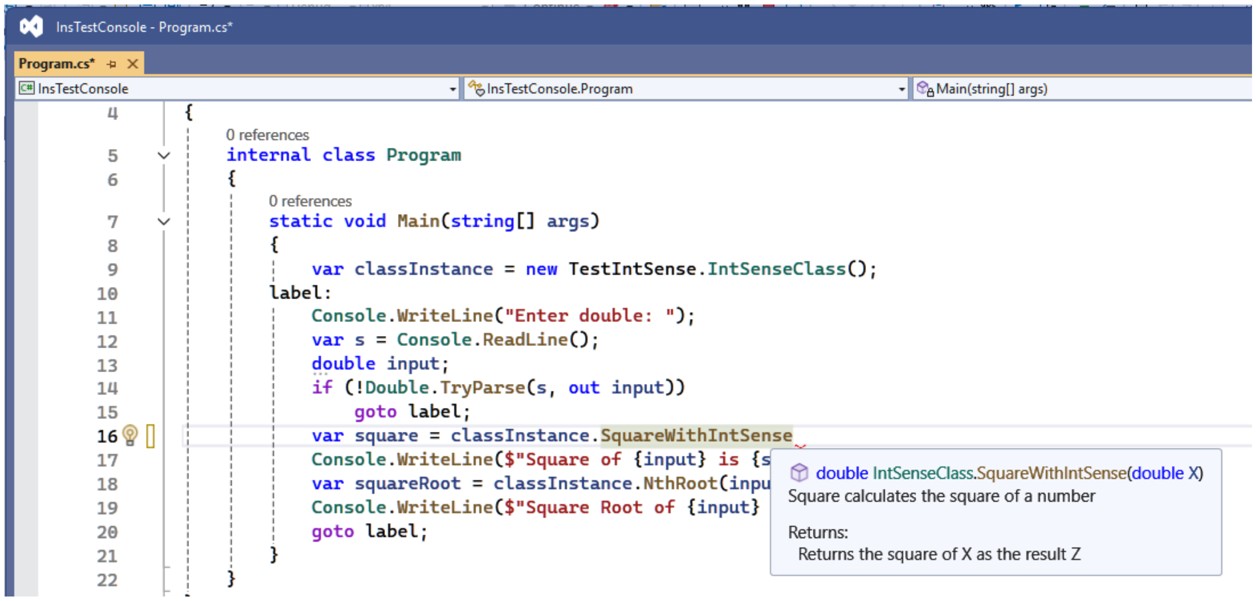
[5;5]



```
     'full' ⎕CSR 'SquareWithIntSense'
/// <summary>Square calculates the square of a number</summary>
/// <param name="X">double to be squared
/// </param>
/// <returns>Returns the square of X as the result Z</returns>
/// <returns>Remark about the SquareWithIntSense method</returns>
public double @SquareWithIntSense(double @X)
{
try
{
    var @arg = this._a_p_l_e_n_g_i_n_e_.FromCs(@X);
    var @res = this._a_p_l_e_n_g_i_n_e_.CallMethod("SquareWithIntSense", @arg);
    this._a_p_l_e_n_g_i_n_e_.ToCs(@res, out double @Z);
    return @Z;
}
catch (Exception @e)
{
throw new Exception(@e.Message);
}
}
```

Ready | Hist: Ln: 6 Col: 44 | Ins | Classic | EN_US

When the APL64 CPC Nuget package containing this APL64 public function is used in a .Net C# application, the public comments converted to .Net IntelliSense documentation are visible in Visual Studio:



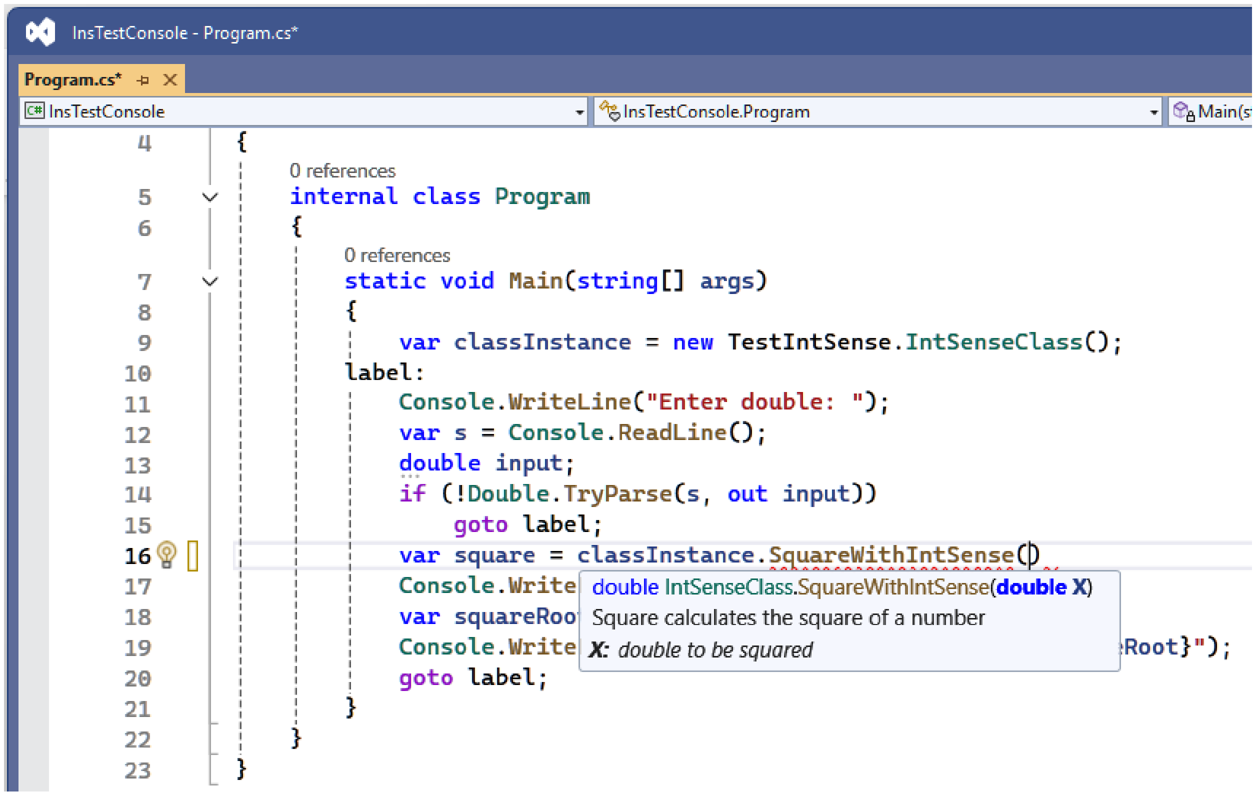
InsTestConsole - Program.cs*

Program.cs*

InsTestConsole | InsTestConsole.Program | Main(string[] args)

```
{
    0 references
    internal class Program
    {
        0 references
        static void Main(string[] args)
        {
            var classInstance = new TestIntSense.IntSenseClass();
        label:
            Console.WriteLine("Enter double: ");
            var s = Console.ReadLine();
            double input;
            if (!Double.TryParse(s, out input))
                goto label;
            var square = classInstance.SquareWithIntSense
            Console.WriteLine($"Square of {input} is {s
            var squareRoot = classInstance.NthRoot(inpu
            Console.WriteLine($"Square Root of {input}
            goto label;
        }
    }
```

```
double IntSenseClass.SquareWithIntSense(double X)
Square calculates the square of a number

Returns:
 Returns the square of X as the result Z
```



InsTestConsole - Program.cs*

Program.cs*

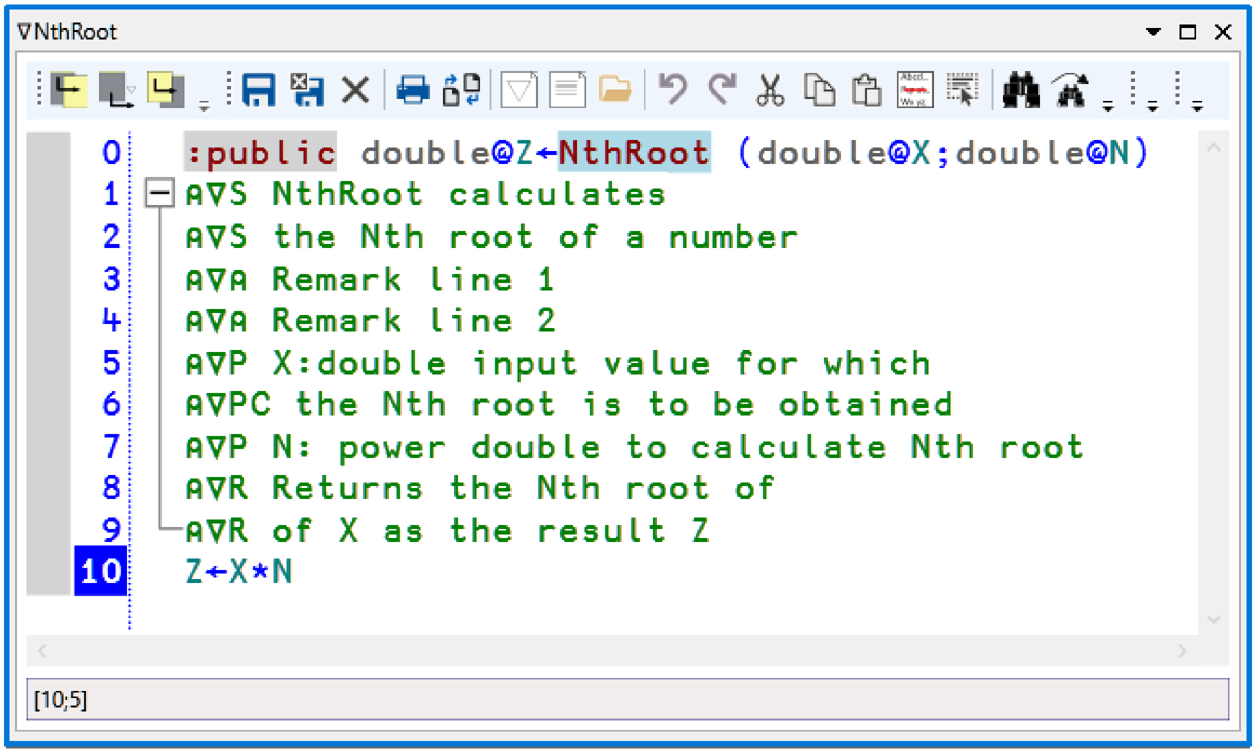
InsTestConsole | InsTestConsole.Program | Main(s

```
{
    0 references
    internal class Program
    {
        0 references
        static void Main(string[] args)
        {
            var classInstance = new TestIntSense.IntSenseClass();
        label:
            Console.WriteLine("Enter double: ");
            var s = Console.ReadLine();
            double input;
            if (!Double.TryParse(s, out input))
                goto label;
            var square = classInstance.SquareWithIntSense()
            Console.Write
            var squareRoo
            Console.Write                                         Root}");
            goto label;
        }
    }
}
```

```
double IntSenseClass.SquareWithIntSense(double X)
Square calculates the square of a number
X: double to be squared
```

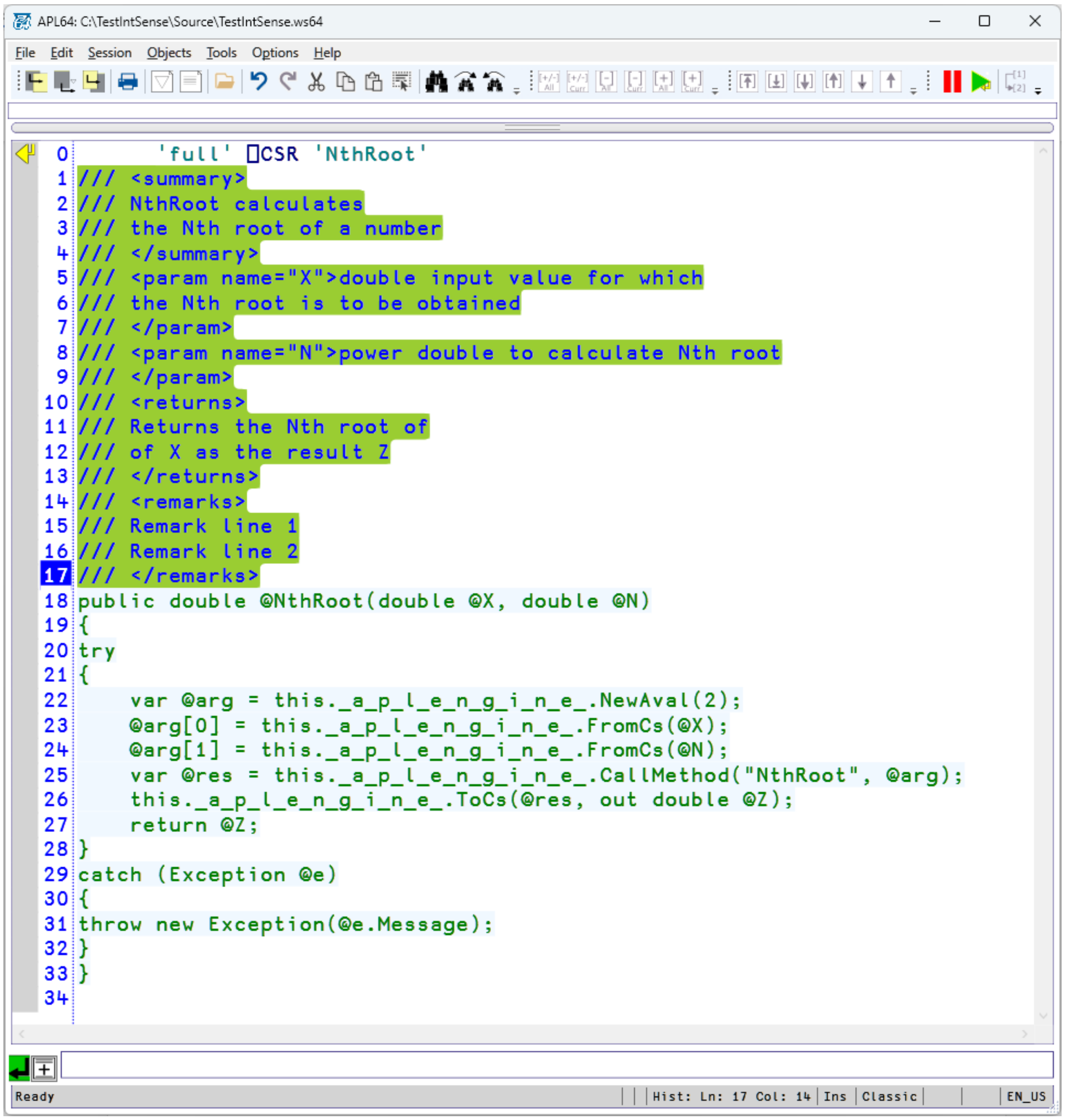
Example: Multi-line IntelliSense Elements

The APL64 programmer-developed public function NthSquare includes public comments to create .Net IntelliSense when it is used in an APL64 CPC. In this example the .Net IntelliSense documentation for Summary, Remarks and Returns are each multiple lines of text, and there are two Params. The params public comments for the argument X is multi-line and the params public comment for the N argument is a single line.


∇NthRoot

```
:public double@Z←NthRoot (double@X;double@N)
⍝∇S NthRoot calculates
⍝∇S the Nth root of a number
⍝∇A Remark line 1
⍝∇A Remark line 2
⍝∇P X:double input value for which
⍝∇PC the Nth root is to be obtained
⍝∇P N: power double to calculate Nth root
⍝∇R Returns the Nth root of
⍝∇R of X as the result Z
Z←X*N
```

[10;5]

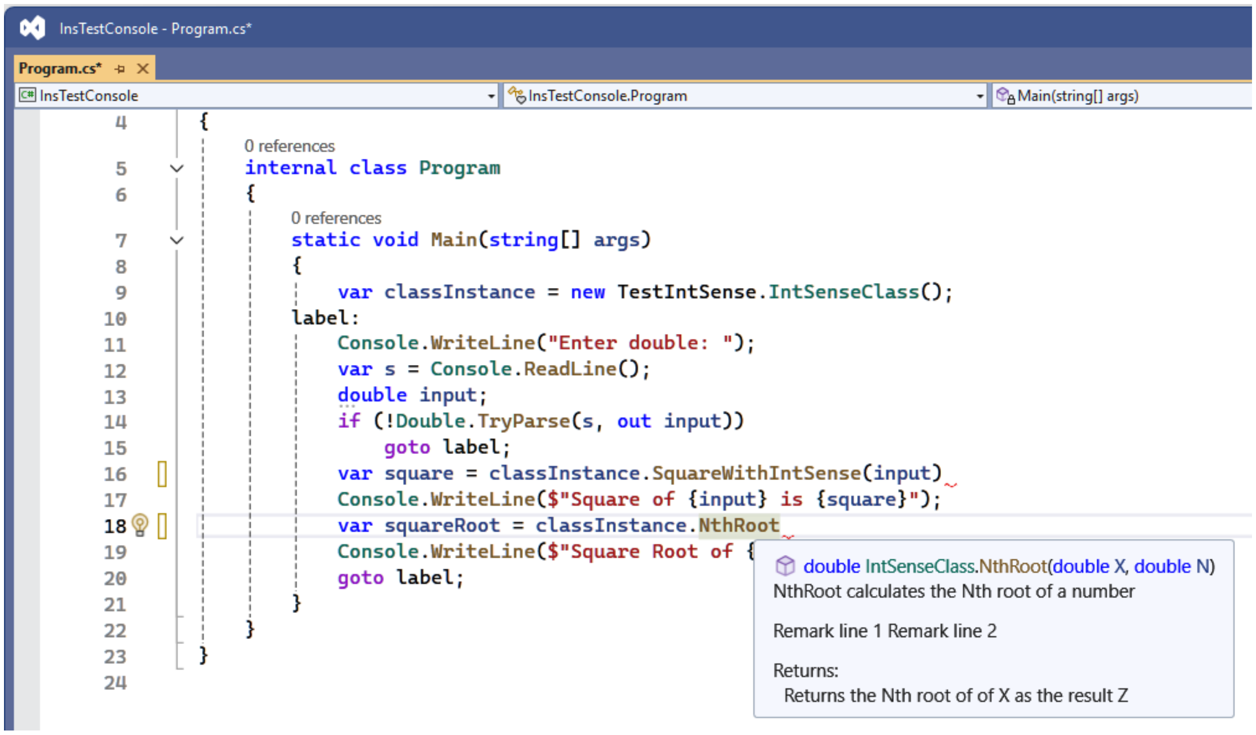

APL64: C:\TestIntSense\Source\TestIntSense.ws64

```
      'full' ⎕CSR 'NthRoot'
/// <summary>
/// NthRoot calculates
/// the Nth root of a number
/// </summary>
/// <param name="X">double input value for which
/// the Nth root is to be obtained
/// </param>
/// <param name="N">power double to calculate Nth root
/// </param>
/// <returns>
/// Returns the Nth root of
/// of X as the result Z
/// </returns>
/// <remarks>
/// Remark line 1
/// Remark line 2
/// </remarks>
public double @NthRoot(double @X, double @N)
{
try
{
    var @arg = this._a_p_l_e_n_g_i_n_e_.NewAval(2);
    @arg[0] = this._a_p_l_e_n_g_i_n_e_.FromCs(@X);
    @arg[1] = this._a_p_l_e_n_g_i_n_e_.FromCs(@N);
    var @res = this._a_p_l_e_n_g_i_n_e_.CallMethod("NthRoot", @arg);
    this._a_p_l_e_n_g_i_n_e_.ToCs(@res, out double @Z);
    return @Z;
}
catch (Exception @e)
{
throw new Exception(@e.Message);
}
}
```

Ready | Hist: Ln: 17 Col: 14 | Ins | Classic | EN_US

CPC Public Function IntelliSense in Visual Studio

When the APL64 CPC Nuget package containing an APL64 public function is used in a .Net C# application, the public comments provided by the APL64 programmer, are visible in Visual Studio programmer in IntelliSense format:



InsTestConsole - Program.cs*

Program.cs*

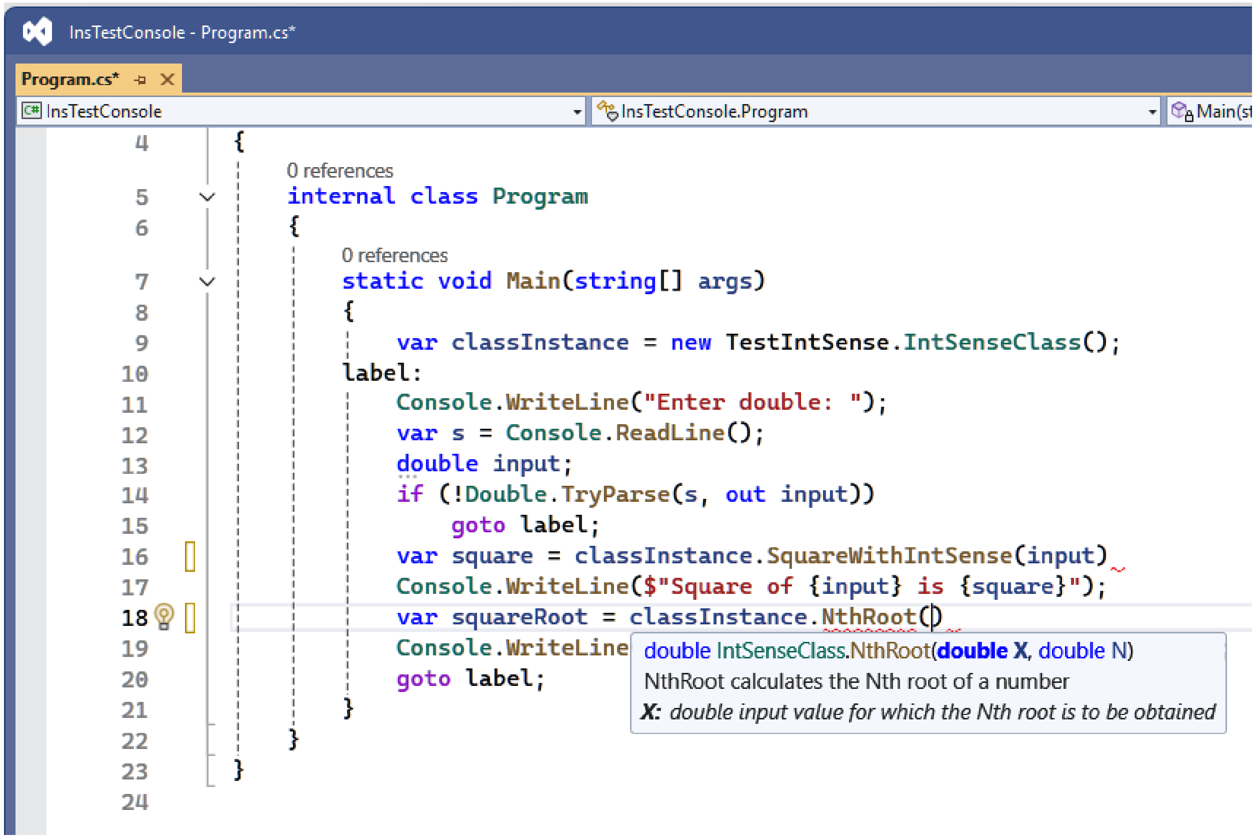
InsTestConsole | InsTestConsole.Program | Main(string[] args)

```
{
    0 references
    internal class Program
    {
        0 references
        static void Main(string[] args)
        {
            var classInstance = new TestIntSense.IntSenseClass();
        label:
            Console.WriteLine("Enter double: ");
            var s = Console.ReadLine();
            double input;
            if (!Double.TryParse(s, out input))
                goto label;
            var square = classInstance.SquareWithIntSense(input)
            Console.WriteLine($"Square of {input} is {square}");
            var squareRoot = classInstance.NthRoot
            Console.WriteLine($"Square Root of {
            goto label;
        }
    }
}
```

double IntSenseClass.NthRoot(double X, double N)
NthRoot calculates the Nth root of a number

Remark line 1 Remark line 2

Returns:
Returns the Nth root of of X as the result Z



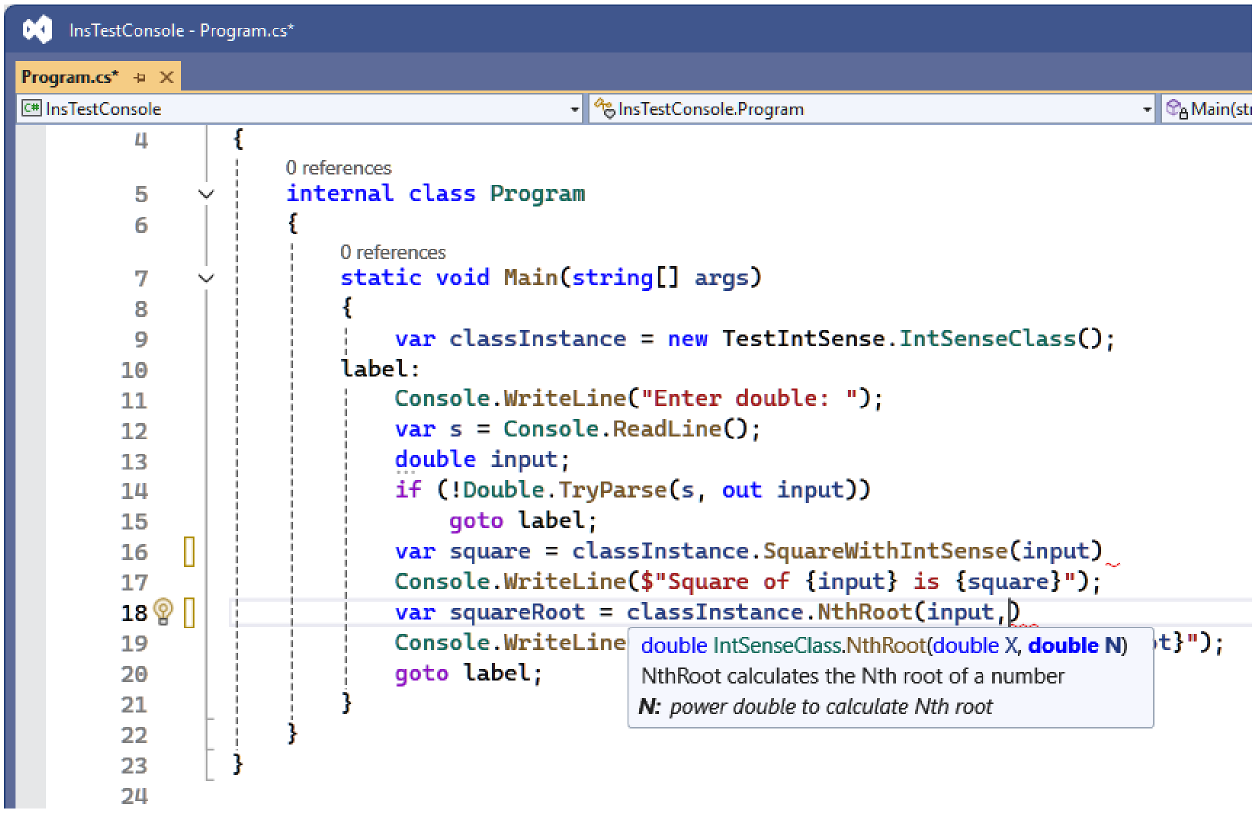
InsTestConsole - Program.cs*

Program.cs*

InsTestConsole | InsTestConsole.Program | Main(st

```
{
    0 references
    internal class Program
    {
        0 references
        static void Main(string[] args)
        {
            var classInstance = new TestIntSense.IntSenseClass();
        label:
            Console.WriteLine("Enter double: ");
            var s = Console.ReadLine();
            double input;
            if (!Double.TryParse(s, out input))
                goto label;
            var square = classInstance.SquareWithIntSense(input)
            Console.WriteLine($"Square of {input} is {square}");
            var squareRoot = classInstance.NthRoot()
            Console.WriteLine
            goto label;
        }
    }
}
```

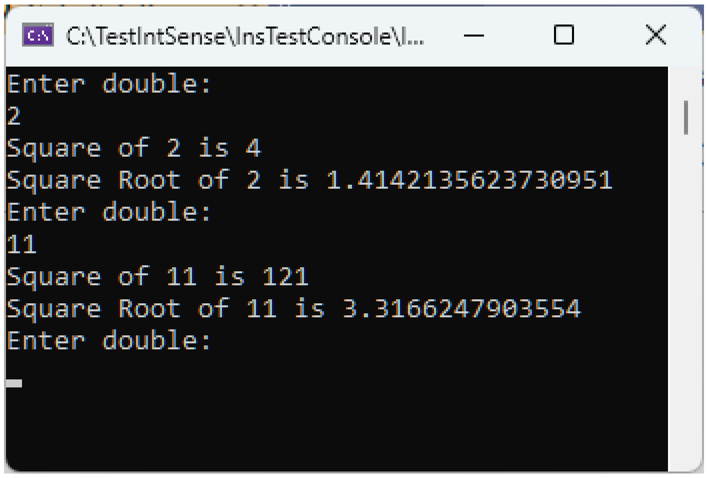
double IntSenseClass.NthRoot(**double X**, double N)
NthRoot calculates the Nth root of a number
X: *double input value for which the Nth root is to be obtained*


InsTestConsole - Program.cs*

```
{
    0 references
    internal class Program
    {
        0 references
        static void Main(string[] args)
        {
            var classInstance = new TestIntSense.IntSenseClass();
        label:
            Console.WriteLine("Enter double: ");
            var s = Console.ReadLine();
            double input;
            if (!Double.TryParse(s, out input))
                goto label;
            var square = classInstance.SquareWithIntSense(input)
            Console.WriteLine($"Square of {input} is {square}");
            var squareRoot = classInstance.NthRoot(input,)
            Console.WriteLine                                    t}");
            goto label;
        }
    }
}
```

```
double IntSenseClass.NthRoot(double X, double N)
NthRoot calculates the Nth root of a number
N: power double to calculate Nth root
```

Example C# Console Project in Operation


C:\TestIntSense\InsTestConsole\I...

```
Enter double:
2
Square of 2 is 4
Square Root of 2 is 1.4142135623730951
Enter double:
11
Square of 11 is 121
Square Root of 11 is 3.3166247903554
Enter double:
```

Missing IntelliSense Documentation

If IntelliSense documentation is not present in an APL64 programmer-developed public function, a warning will be issued when the APL64 CPC containing that public function is created in APL64. The resulting CPC will be usable by a .Net programmer without the benefit of context-sensitive documentation for the class and methods exposed by that CPC. Visual

Studio may warn of the missing documentation when the .Net solution containing the CPC Nuget package without IntelliSense documentation is run in debug mode.

⎕CPCIS CPCreate IntelliSense Scaffold Tool

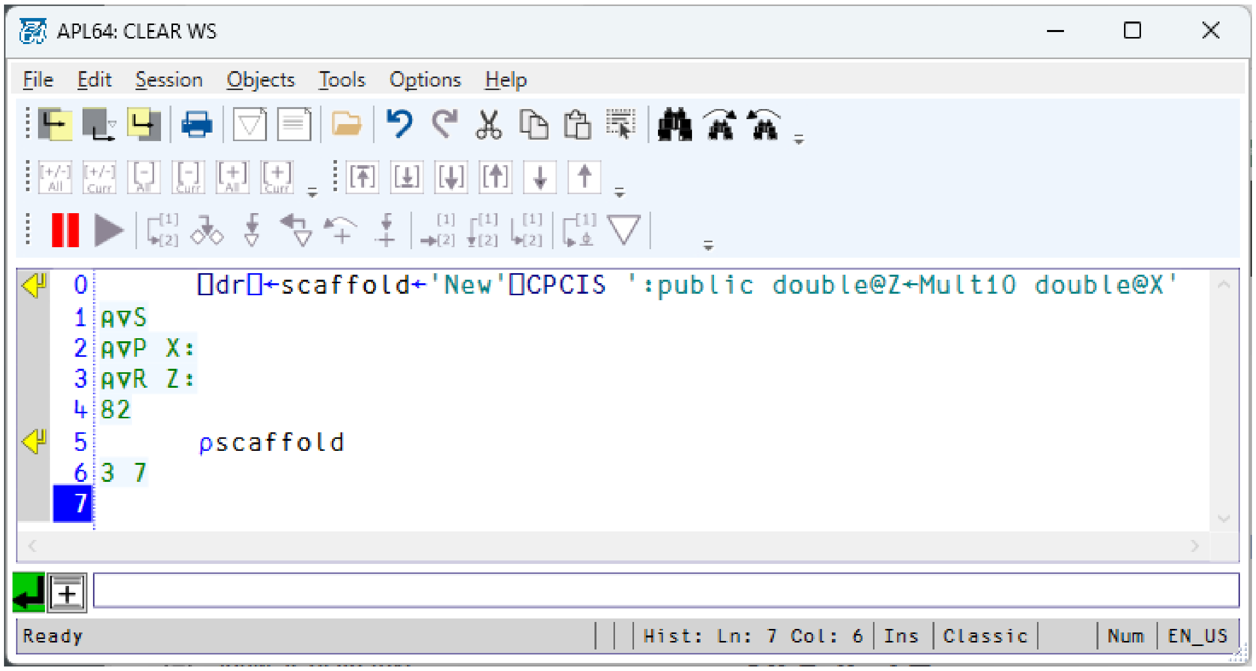
The ⎕CPCIS system function facilitates creating and maintaining IntelliSense remarks for public functions in an the APL64 cross-platform component.

Syntax: res←lArg ⎕CPCIS rArg

Create new IntelliSense scaffold from public function header

scaffoldCharMat ← 'New' ⎕CPCIS functionLine[0]

```
⎕dr⎕←scaffold←'New'⎕CPCIS ':public double@Z←Mult10 double@X'
⍴scaffold
```



```
      ⎕dr⎕←scaffold←'New'⎕CPCIS ':public double@Z←Mult10 double@X'
⍝∇S
⍝∇P X:
⍝∇R Z:
82
      ⍴scaffold
3 7
```

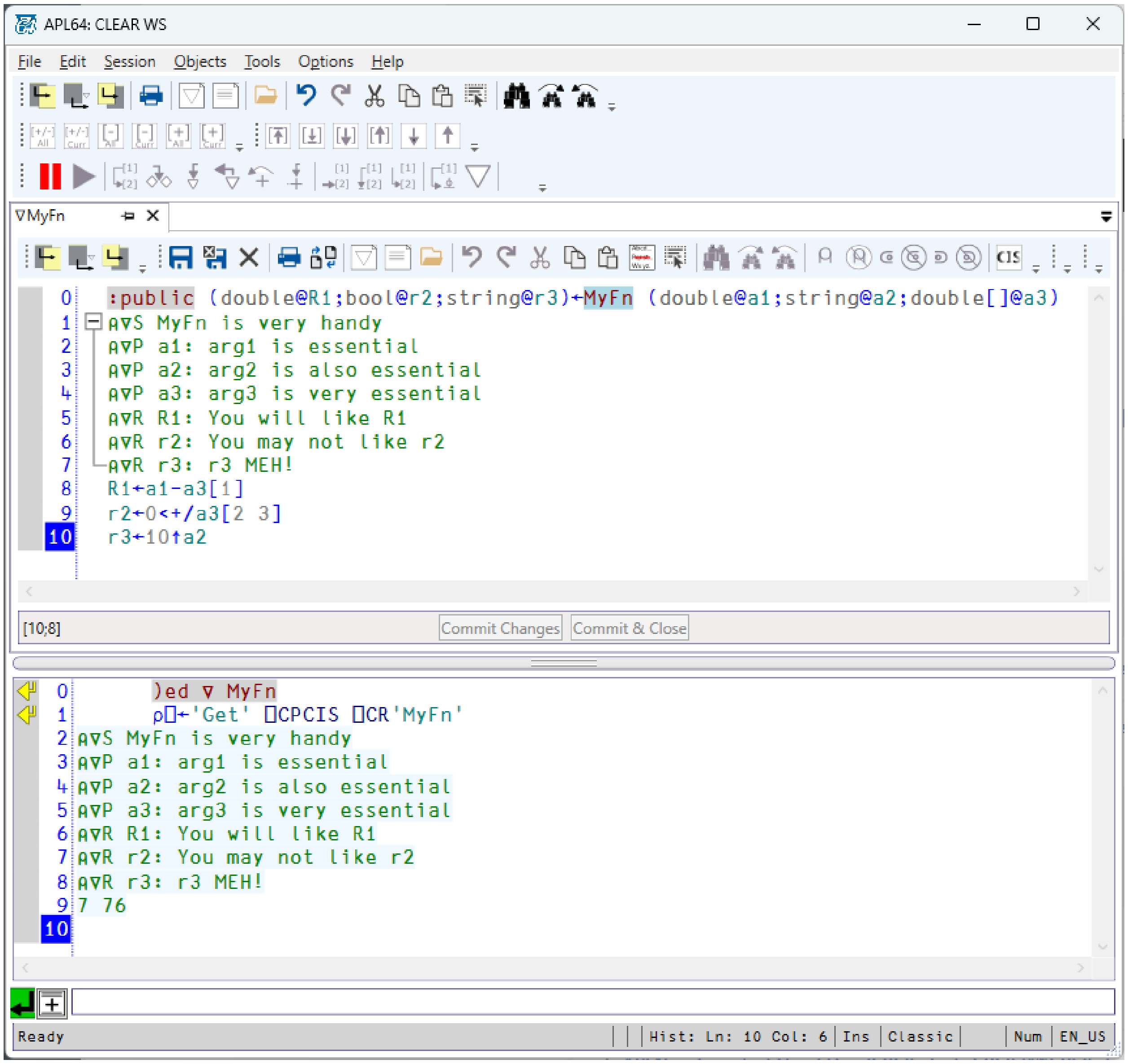
Get existing IntelliSense scaffold, if any, from public function ⎕CR

scaffoldCharMAt ← 'Get' ⎕CPCIS functionCR

```
:public (double@R1;bool@r2;string@r3)←MyFn (double@a1;string@a2;double[]@a3)
⍝∇S MyFn is very handy
⍝∇P a1: arg1 is essential
⍝∇P a2: arg2 is also essential
⍝∇P a3: arg3 is very essential
```

```
⍝∇R R1: You will like R1
⍝∇R r2: You may not like r2
⍝∇R r3: r3 MEH!
R1←a1-a3[1]
r2←0<+/a3[2 3]
r3←10↑a2
```

```
ρ⎕←'Get' ⎕CPCIS ⎕CR'MyFn'
```



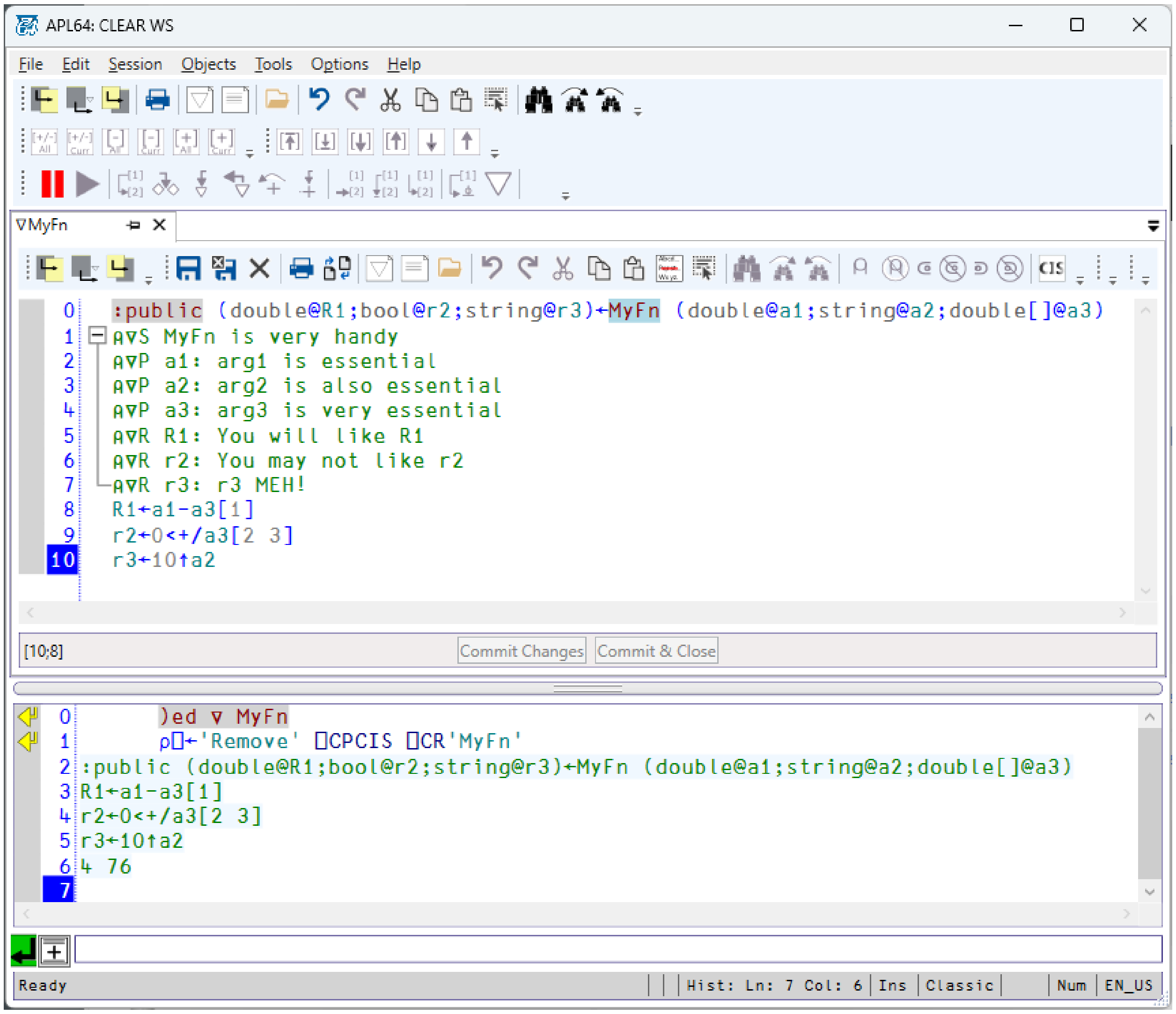
```
:public (double@R1;bool@r2;string@r3)←MyFn (double@a1;string@a2;double[]@a3)
⍝∇S MyFn is very handy
⍝∇P a1: arg1 is essential
⍝∇P a2: arg2 is also essential
⍝∇P a3: arg3 is very essential
⍝∇R R1: You will like R1
⍝∇R r2: You may not like r2
⍝∇R r3: r3 MEH!
R1←a1-a3[1]
r2←0<+/a3[2 3]
r3←10↑a2
```

```
      )ed ∇ MyFn
      ρ⎕←'Get' ⎕CPCIS ⎕CR'MyFn'
⍝∇S MyFn is very handy
⍝∇P a1: arg1 is essential
⍝∇P a2: arg2 is also essential
⍝∇P a3: arg3 is very essential
⍝∇R R1: You will like R1
⍝∇R r2: You may not like r2
⍝∇R r3: r3 MEH!
7 76
```

Remove existing IntelliSense scaffold, if any, from public function ⎕CR

```
updatedFnCR ← 'Remove' ⎕CPCIS  sourceFunctionCR
```

```
ρ⎕←'Remove' ⎕CPCIS ⎕CR'MyFn'
```



```
:public (double@R1;bool@r2;string@r3)←MyFn (double@a1;string@a2;double[]@a3)
⍝∇S MyFn is very handy
⍝∇P a1: arg1 is essential
⍝∇P a2: arg2 is also essential
⍝∇P a3: arg3 is very essential
⍝∇R R1: You will like R1
⍝∇R r2: You may not like r2
⍝∇R r3: r3 MEH!
R1←a1-a3[1]
r2←0<+/a3[2 3]
r3←10↑a2
```

```
      )ed ∇ MyFn
      ρ⎕←'Remove' ⎕CPCIS ⎕CR'MyFn'
:public (double@R1;bool@r2;string@r3)←MyFn (double@a1;string@a2;double[]@a3)
R1←a1-a3[1]
r2←0<+/a3[2 3]
r3←10↑a2
4 76
```

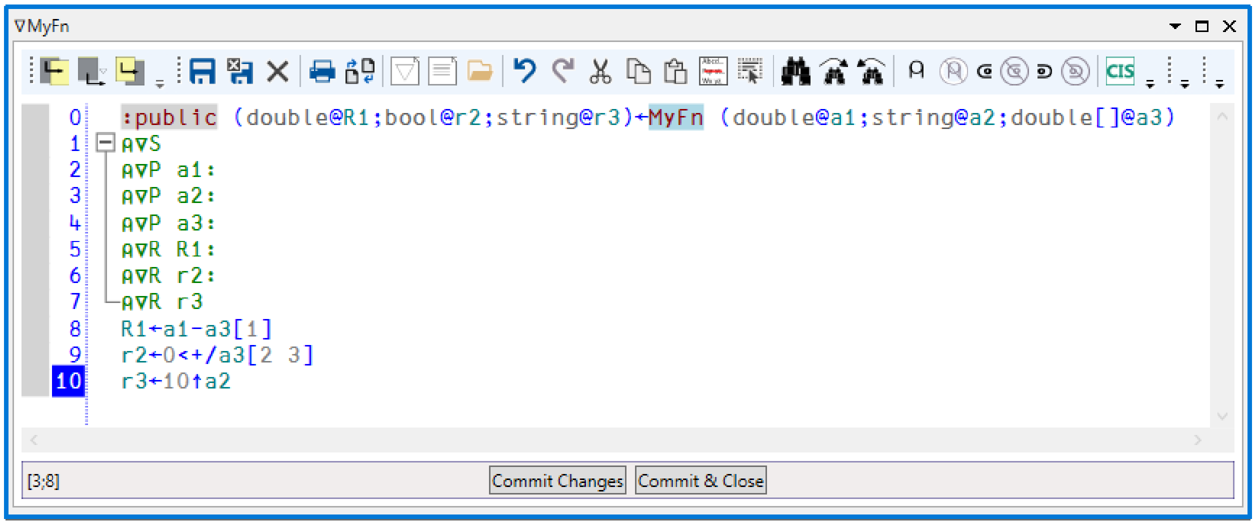
```
Ready        Hist: Ln: 7 Col: 6 | Ins | Classic |   Num | EN_US
```

Replace existing IntelliSense scaffold in a public function ⎕CR

updatedFunctionCR ← 'Replace' ⎕CPCIS srcFnCR newScaffoldCharMat

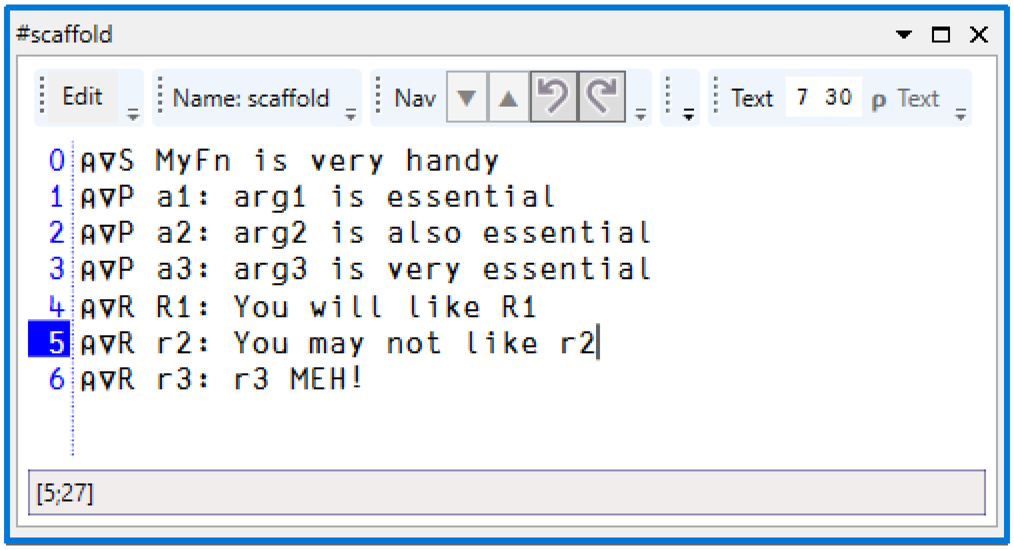
```
:public (double@R1;bool@r2;string@r3)←MyFn (double@a1;string@a2;double[]@a3)
⍝∇S
⍝∇P a1:
⍝∇P a2:
⍝∇P a3:
⍝∇R R1:
⍝∇R r2:
⍝∇R r3
R1←a1-a3[1]
```

```
r2←0<+/a3[2 3]
r3←10↑a2
```



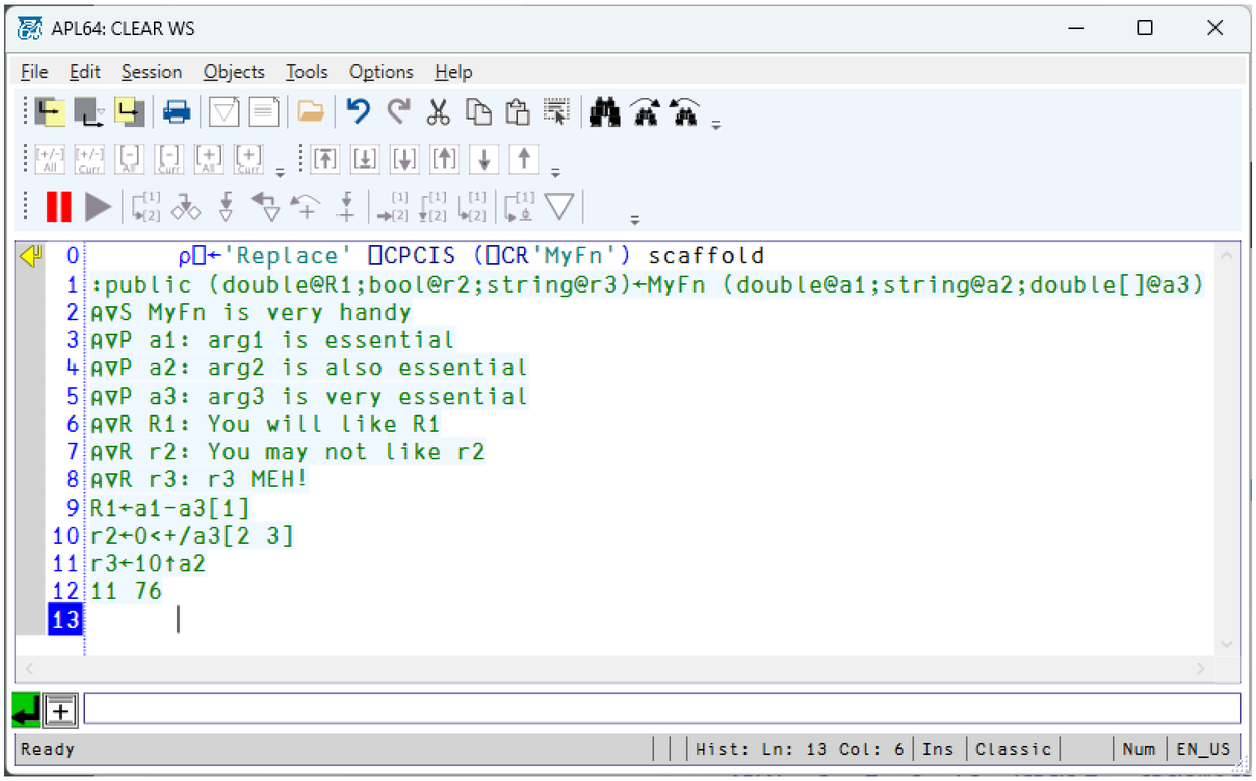
```
:public (double@R1;bool@r2;string@r3)←MyFn (double@a1;string@a2;double[]@a3)
⍝∇S
⍝∇P a1:
⍝∇P a2:
⍝∇P a3:
⍝∇R R1:
⍝∇R r2:
⍝∇R r3
R1←a1-a3[1]
r2←0<+/a3[2 3]
r3←10↑a2
```

```
⍝∇S MyFn is very handy
⍝∇P a1: arg1 is essential
⍝∇P a2: arg2 is also essential
⍝∇P a3: arg3 is very essential
⍝∇R R1: You will like R1
⍝∇R r2: You may not like r2
⍝∇R r3: r3 MEH!
```



```
⍝∇S MyFn is very handy
⍝∇P a1: arg1 is essential
⍝∇P a2: arg2 is also essential
⍝∇P a3: arg3 is very essential
⍝∇R R1: You will like R1
⍝∇R r2: You may not like r2
⍝∇R r3: r3 MEH!
```

```
ρ⎕←'Replace' ⎕CPCIS (⎕CR'MyFn') scaffold
```



```
      ρ⎕←'Replace' ⎕CPCIS (⎕CR'MyFn') scaffold
:public (double@R1;bool@r2;string@r3)←MyFn (double@a1;string@a2;double[]@a3)
⍝∇S MyFn is very handy
⍝∇P a1: arg1 is essential
⍝∇P a2: arg2 is also essential
⍝∇P a3: arg3 is very essential
⍝∇R R1: You will like R1
⍝∇R r2: You may not like r2
⍝∇R r3: r3 MEH!
R1←a1-a3[1]
r2←0<+/a3[2 3]
r3←10↑a2
11 76
```