APL64 Cross-platform Component

Contents

Rationale for APL64 Cross-platform Components

APL programmers have created many significant, unique and useful algorithms which can now be made available to non-APL, .Net programmers using the APL64 cross-platform component technology.

Examples:

An APL programmer may develop specialized algorithms which satisfy a strategic need of an enterprise. The design, implementation and test of these algorithms is greatly facilitated by the APL programming context. Using the APL64 cross-platform technology, these algorithmic assets can be made available across the enterprise in an industry-standard format. Enterprise .Net programmers can build web based, desktop, tablet or phone applications which consume an APL64 cross-platform component without any knowledge of APL.

A significant mainstream software development industry segment are vendors of component libraries, licensed on a subscription basis such as Component One, Syncfusion, Microsoft Marketplace, etc. An APL64 cross-platform component can be incorporated into such libraries and used by non-APL programmers.

What is an APL64 Cross-platform Component?

An APL64 cross-platform component is a Nuget package containing a .Net Core assembly which exposes one or more APL64 programmer-defined ‘public’ functions. This package may be used as part of a containing .Net application to support the business rules or algorithms of the application. Any .Net programming language may be used to access the APL64 cross-platform component.

An APL64 programmer may create an APL64 workspace which contains one or more programmer-defined functions. At least one of these functions must be a 'public' function so that it can be exposed as a .Net method for use by the containing application system. Other APL64 programmer-defined functions which are not 'public' can be included in the workspace to support the public functions.

A 'public' APL64-programmer defined function in an APL64 cross-platform component can be called as a .Net method from the containing .Net application which references the components. That function can also be run in APL64 like any other APL64 programmer-defined function.

Platforms Supported

An APL64 component can target the Windows 10/11 operating system, Android operating system, iOs/MacOs and Linux with .Net.

Support for an APL64 cross-platform is available for an operating system platform only if that platform supports the .Net Standard technology, for example many versions of the Linux operating system support .Net.

Elements of an APL64 Cross-Platform Component

An APL64 cross-platform component is designed to support APL-based processing of data resulting in arrays of calculated values and text, etc. A cross-platform component is an APL64-based calculation engine which can receive data from a .Net application, process that data and return the processing results to the .Net application.

Because an APL64 component is designed to be cross-platform, no APL function in the component can display a graphical user interface (GUI). Exceptions in an APL64 component must be passed to the containing .Net application as exception codes and/or exception texts in an application-specific protocol.

An APL64 cross-platform is based on an APL64 workspace containing one or more APL64 programmer-defined 'public' functions and, if necessary, non-public APL64 programmer-defined functions and variables.

Public Functions

Public Function Overview

APL64 public functions are those APL64 user-defined functions which will be exposed to a user of a cross-platform component. APL64 user-defined functions which are not specified by the APL64 programmer as public will not be exposed to the user of a cross-platform component.

An APL64 public function supports APL64--compatible, .Net data types as results or arguments:

APL64-compatible .Net Data Type	**Rank**	**.Net Array Examples**
Boolean or bool	0 – 3	bool bool[] bool[,] bool[,,]
Int32 or int	0 – 3	int int[] Int[,] int[,,]
Double or double	0 – 3	double double[] double[,] double[,,]

Char or char	0 – 3	char char[] char[,] char[,,]
String or string	0 – 3	string string[] string[,] string[,,]
Object	0 – 3	object object[] object[,] object[,,]
Object	More than 3	object using .Net boxing and APL ⊂ and ⊃

For all APL64 public function data types, except the Object type, the value of an APL64 public function result or argument must have a homogeneous data type.

For the Object data type, the result or argument may include elements of different APL64-compatible, .Net data types.

Public Function Header

An APL64 programmer-defined function is marked as public using the function header. The public function header has several components:

Public Function Header Prefix

A public function header must begin with the ':public' prefix.

Results

A public function result is optional. The result specification, if any, must be enclosed in parentheses, '(…)', except that the parentheses are optional if the public function has only one result. The result specification must be followed by the assignment glyph (←).

Multiple results are supported with each result separated by a semi-colon (;) and contained within the enclosed parentheses of the public function result specification.

A public function result contains:

- The APL64-compatible, .Net data type and rank
- An 'at-sign' (@) separator
- A valid APL64 variable name, excluding those containing 'Δ⍙'

Rank of the array is specified after the .Net data type with brackets ([…]) containing semi-colons(;) indicating the rank, e.g. for rank zero: no brackets, for rank one: double[], for rank two: double[;], for rank three: double[;;].

Left Argument

A left argument to an APL64 public function is not supported.

Function Name

The name of a public function can be any valid APL64 user-defined function name, excluding those containing 'Δ⍙'

Right Arguments

A public function right argument is optional. The right argument specification, if any, must be enclosed in parentheses, '(…)', except that the parentheses are optional if the public function has only one argument.

Multiple arguments are supported with each argument separated by a semi-colon (;) and contained within the enclosed parentheses of the public function argument specification.

A public function argument contains:

- The APL64-compatible, .Net data type and rank
- An ‘at-sign’ (@) separator
- A valid APL64 variable name, excluding those containing ‘Δ⍙’

Rank of the array is specified after the .Net data type with brackets ([…]) containing semi-colons(;) indicating the rank, e.g. for rank zero: no brackets, for rank one: double[], for rank two: double[;], for rank three: double[;;].

Local Variables and Functions

A local variable or local function in an APL64 public function is optional. Local objects, if any, must follow the function name, if the is no argument specification, or the argument specification, if present. Multiple local variables and functions are supported with each local variable separated by a space or semi-colon (;).

The name of a local variable or function may be any valid APL64 object name or a localizable APL64 system variable.

The value of a local variable may be any APL64 data type.

Local variables and functions are not exposed to a .Net application using an APL64 cross-platform component.

Such local functions cannot themselves be public functions.

⎕CSR – Public Function(s) Name, Header

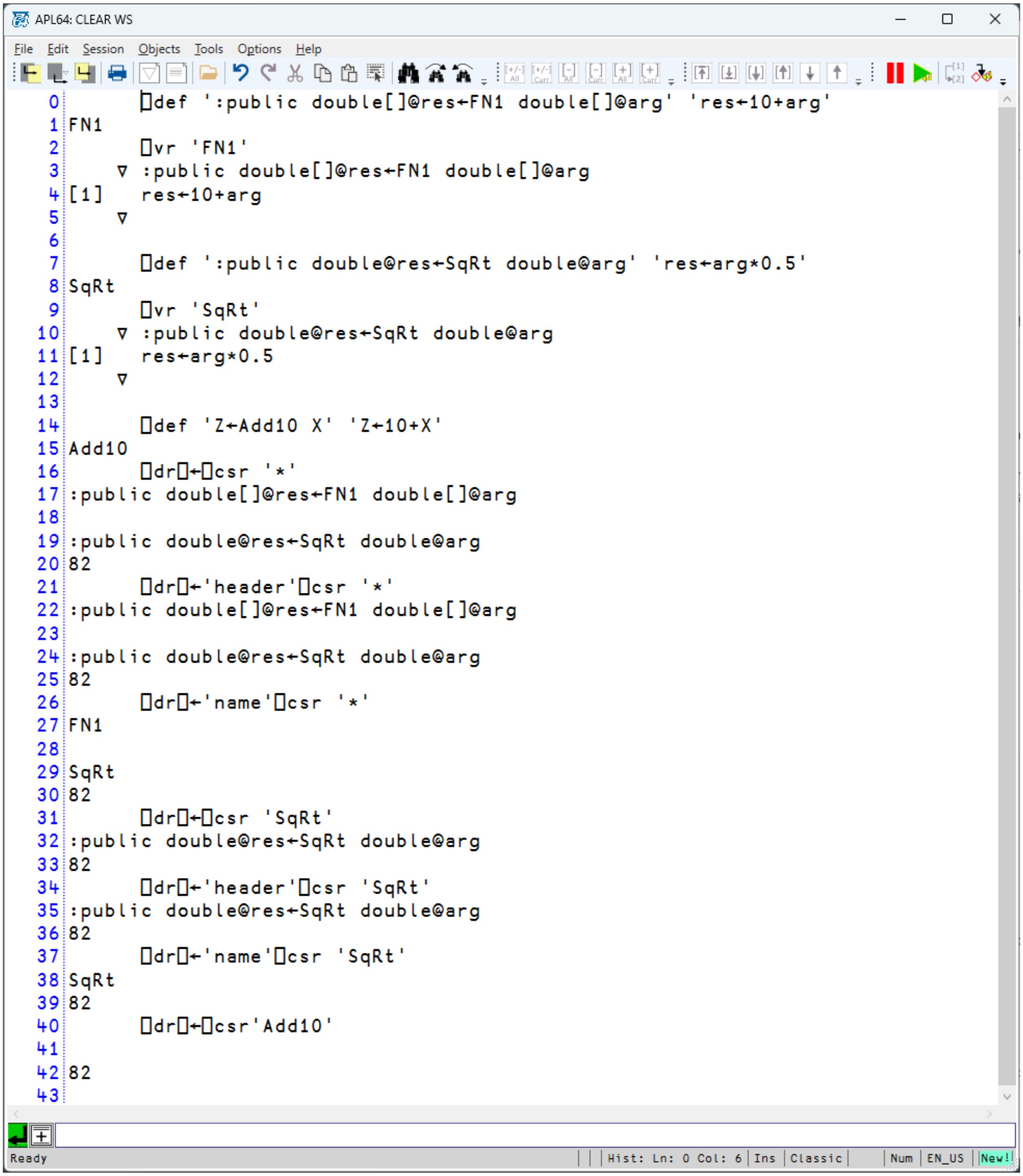
The ⎕CSR system function returns the name(s) or header(s) of the public functions in the current workspace as a character vector. If ⎕CSR returns information for more than one public function, each public function item is separated by a new line character.

Syntax: result ← larg ⎕CSR rarg

Arguments :

- larg: Optional: ‘header’ (default value), ‘name’
- rarg: ‘*’/All public functions in the current workspace, ‘publicFnName’/Specified public function

Examples:



```
      ⎕def ':public double[]@res←FN1 double[]@arg' 'res←10+arg'
FN1
      ⎕vr 'FN1'
     ∇ :public double[]@res←FN1 double[]@arg
[1]   res←10+arg
     ∇

      ⎕def ':public double@res←SqRt double@arg' 'res←arg*0.5'
SqRt
      ⎕vr 'SqRt'
     ∇ :public double@res←SqRt double@arg
[1]   res←arg*0.5
     ∇

      ⎕def 'Z←Add10 X' 'Z←10+X'
Add10
      ⎕dr⎕←⎕csr '*'
:public double[]@res←FN1 double[]@arg

:public double@res←SqRt double@arg
82
      ⎕dr⎕←'header'⎕csr '*'
:public double[]@res←FN1 double[]@arg

:public double@res←SqRt double@arg
82
      ⎕dr⎕←'name'⎕csr '*'
FN1

SqRt
82
      ⎕dr⎕←⎕csr 'SqRt'
:public double@res←SqRt double@arg
82
      ⎕dr⎕←'header'⎕csr 'SqRt'
:public double@res←SqRt double@arg
82
      ⎕dr⎕←'name'⎕csr 'SqRt'
SqRt
82
      ⎕dr⎕←⎕csr'Add10'

82

```

Public Function with Result or Argument of Rank > 3

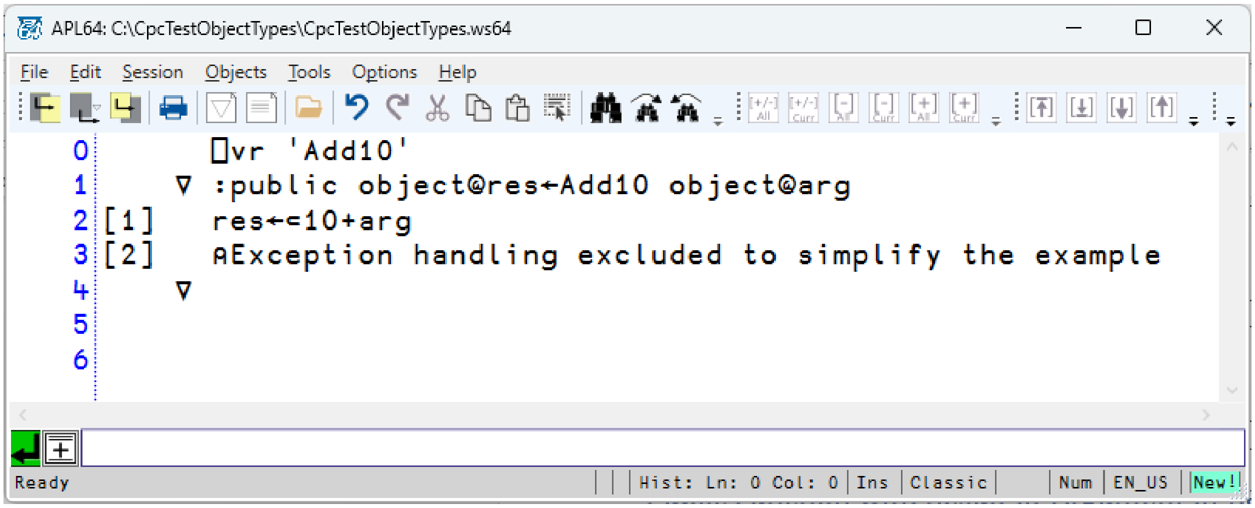
Syntactically, public function headers with high-rank results or arguments can be unwieldly, e.g.:

```
:public char[;;;;;;;;]@res←MyFn (Int32[;;;;;;;;;;;]@arg1;Double[;;;;;;]@arg2)
```

In APL64, such semi-colon-delimited syntax is limited to arrays of rank three or less. When an APL64 public function must receive an argument or return a result of rank greater than three, use:

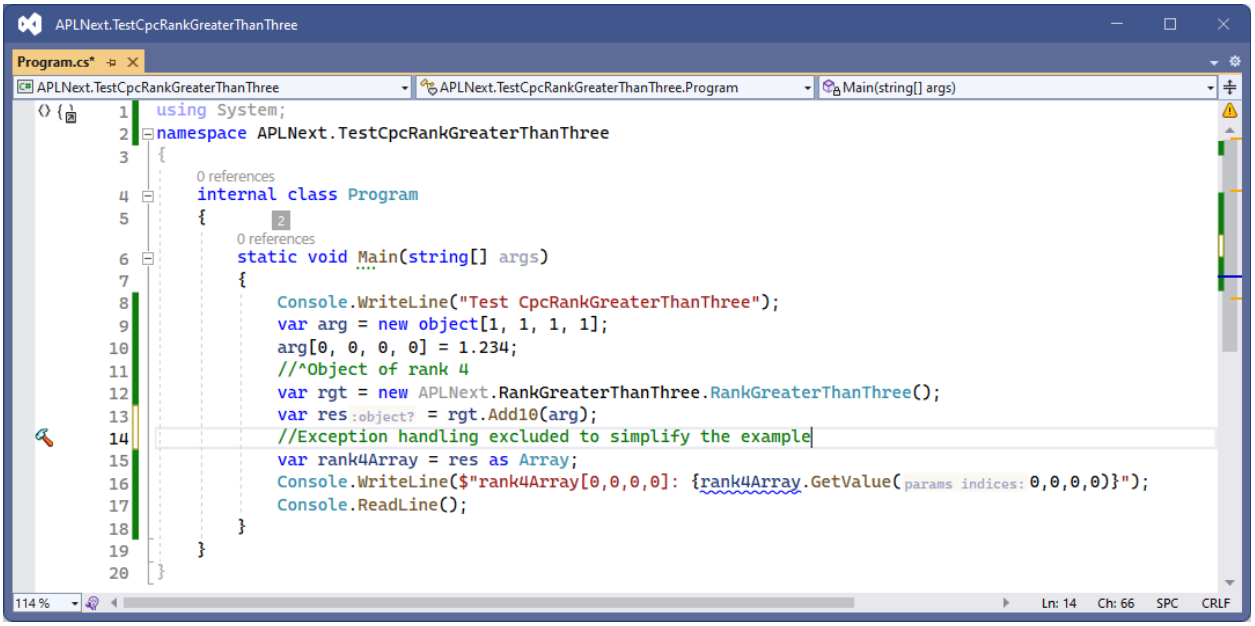
- The .Net object type
- The boxing and unboxing capabilities of the .Net object and array types
- The APL64 enclose and disclose functions

Consider the 'Add10' public function. Since APL64 can add 10 to an array of any rank or depth, the argument received from .Net does not need to be disclosed. Since the result type expected by .Net is object, the result of the APL64 function must be enclosed so .Net will receive it as a .Net object.



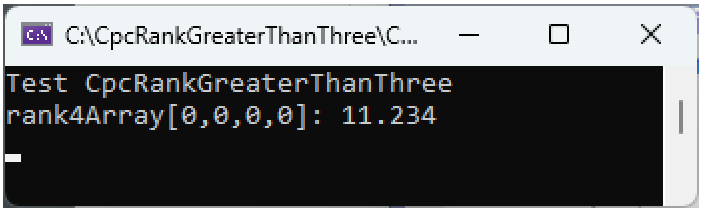
```
        ⎕vr 'Add10'
      ∇ :public object@res←Add10 object@arg
[1]     res←⊂10+arg
[2]     ⍝Exception handling excluded to simplify the example
      ∇
```

Consider the .Net console project which uses the APL64 public function Add10 to add 10 to a numeric array of rank four. The rank four numeric array is passed directly to the APL64 public function Add10 which returns a .Net object type result. By casting the object result to a .Net array, the underlying rank four result is unboxed.


APLNext.TestCpcRankGreaterThanThree
Program.cs*
APLNext.TestCpcRankGreaterThanThree
APLNext.TestCpcRankGreaterThanThree.Program
Main(string[] args)
114 %
Ln: 14
Ch: 66
SPC
CRLF

```
using System;
namespace APLNext.TestCpcRankGreaterThanThree
{
    internal class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Test CpcRankGreaterThanThree");
            var arg = new object[1, 1, 1, 1];
            arg[0, 0, 0, 0] = 1.234;
            //^Object of rank 4
            var rgt = new APLNext.RankGreaterThanThree.RankGreaterThanThree();
            var res = rgt.Add10(arg);
            //Exception handling excluded to simplify the example
            var rank4Array = res as Array;
            Console.WriteLine($"rank4Array[0,0,0,0]: {rank4Array.GetValue(0,0,0,0)}");
            Console.ReadLine();
        }
    }
}
```

The resulting output of the .Net console project:



```
Test CpcRankGreaterThanThree
rank4Array[0,0,0,0]: 11.234
```

In this example, because the underlying datatype of the argument and result are not included in the APL64 public function header, it is up to the APL64 programmer to provide the end user of the cross-platform component with sufficient documentation about the arguments and results.

Public Function with Non-homogenous Result or Argument

When an APL64 public function must receive an argument or return a result which does not have homogeneous element types, the .Net object data type is used.

For objects of rank 0 – 3, the APL64 public function header formats are used:

- object@varName
- object[]@varName
- object[,]@varName
- object[,,]@varName

For objects of rank greater than three use the object@varName format.

When an APL64 public function has a result or argument of .Net object type, it is up to the APL64 programmer to provide the end user of the cross-platform component with sufficient documentation about the arguments and results.

As an alternative to using the object data type for an argument or result of an APL64 public function, consider defining the APL64 public function with multiple explicit results and arguments, each with a homogeneous (non-object) data type. For example:

Suppose the argument types of the MyFn APL64 public function are Int32[], Double[,] and Char[,] and the result type is Char[,], the more explicit function header would be:

```
:public char[,]@res←MyFn Int32[]@arg1;Double[,]@arg2;Char[,]arg3
```

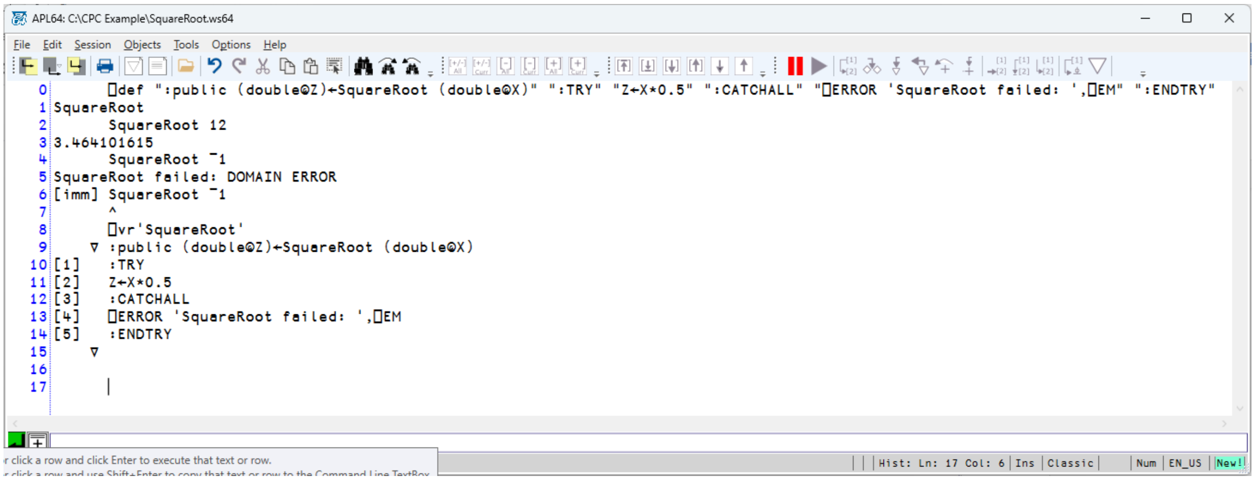
Instead of the non-specific form combining the argument into an object:

```
:public char[,]@res←MyFn object@arg
```

APL64 Programmer-defined Public Function Examples

One result (rank 0) and one argument (rank 0)

```
⎕def ":public (double@Z)←SquareRoot (double@X)" ":TRY" "Z←X*0.5" ":CATCHALL" "⎕ERROR
'SquareRoot failed: ',⎕EM" ":ENDTRY"
```


APL64: C:\CPC Example\SquareRoot.ws64
File Edit Session Objects Tools Options Help
Hist: Ln: 17 Col: 6 | Ins | Classic | Num | EN_US | New!

```
      ⎕def ":public (double@Z)←SquareRoot (double@X)" ":TRY" "Z←X*0.5" ":CATCHALL" "⎕ERROR 'SquareRoot failed: ',⎕EM" ":ENDTRY"
SquareRoot
      SquareRoot 12
3.464101615
      SquareRoot ¯1
SquareRoot failed: DOMAIN ERROR
[imm] SquareRoot ¯1
      ^
      ⎕vr'SquareRoot'
    ∇ :public (double@Z)←SquareRoot (double@X)
[1]   :TRY
[2]   Z←X*0.5
[3]   :CATCHALL
[4]   ⎕ERROR 'SquareRoot failed: ',⎕EM
[5]   :ENDTRY
    ∇

      |
```

r click a row and click Enter to execute that text or row.
r click a row and use Shift+Enter to copy that text or row to the Command Line TextBox

Three results (rank 0, 1, 2) and two arguments (rank1, 3)

```
:public (string@res1;string[]@res2;double[;]@res3)←FNEX1(int[]@arg1;char[;;]@arg2);local1;local2
```

One result (rank 1) and no arguments

```
:public double[]@res1←FNEX2
```

No result and two arguments (rank 1 and 3)

```
:public FNEX3 (int[]@arg1;char[;;]@arg2);local1;local2
```

One result (rank 0) and one argument (rank 0), with parenthesis optional

```
:public int@Z←FNEX4 int@X
```

Two results (rank 0, 1) and one argument (rank 0)

```
:public (int@Z1;char[]@Z2)←FNEX4 int@X
```

One result (rank 1) and two arguments (rank 0, rank 0)

```
:public char[]@Z←FNEX4 (int@X1;int@X2)
```

One result (rank 1) and two arguments (rank 0, rank 0)

```
:public object[]@Z←FNEX4 (int@X1;int@X2)
```

Processing of Public Function Arguments and Results

Within the body of an APL64 public function, all objects are APL64 objects, and all processing is done by the APL64 interpreter.

When an APL64 cross-platform component is used within a .Net application, conversion of the arguments and results, if any, of an APL64 public function must be performed. These conversions are automatically performed by the APL64 interpreter outside of the scope of the APL64 public function.

When an APL64 public function is not used within an APL64 cross-platform component, no conversions are necessary as function is used within an APL64 application.

Within the context of an APL64 cross-platform component:

- The APL64 interpreter converts the .Net value of the function arguments, if any, to APL64 variables for use within that function.
- When the APL64 public function with a result ends execution, the APL64 interpreter converts the APL64 value of the result to a .Net variable.
- Argument and result conversions are performed whenever possible according to the APL64 programmer-specified public function header and the availability of an appropriate APL64 data type.

Conversion of APL64 Public Function Arguments to APL64 Data Types

From .Net Data Type	To APL64 Data Type	Interpreter Conversion of Argument from .Net Data Type to APL64 Data Type
Boolean	Boolean	Yes
Char	Char	Yes
Double	Double	Yes
Int32	Int32	Yes
String	String	Yes
Object	Based on source object element data types	Yes, if .Net object element type is APL64-compatible, otherwise, an exception is thrown
All other .Net data types	N/A	Exception thrown

Conversion of APL64 Public Function Results to .Net Data Types

From APL64 Data Type	To .Net Data Type	Interpreter Conversion Of Result from APL64 Data Type To .Net Data Type
Boolean	Boolean	Yes
Boolean	All other .Net data types	Exception thrown
Char	Char	Yes
Char	All other .Net data types	Exception thrown
Double	Double	Yes
Double	All other .Net data types	Exception thrown
Int32	Double	Yes
Int32	Int32	Yes
Int32	All other .Net data types	Exception thrown
APL64 Object	.Net Object	Yes: .Net object element type based on APL64 object element type
String	String	Yes
String	All other .Net data types	Exception thrown

Exceptions In APL64 Public Functions

Exceptions due to Conversions between .Net and APL64

Variables in .Net are strongly typed and APL64 supports a well-documented subset of those types (String, Char, Int32, Double, Bool).

When an APL64 public function is executed within an APL64 cross-platform component, the APL64 converts the .Net arguments to APL64 variable values and converts the APL64 function results to .Net variable values. If such a conversion is not possible, the APL64 interpreter throws a .Net exception which can be captured by the .Net application using the APL64 cross-platform component.

The APL64 public function does not throw these .Net data conversion exceptions and these exceptions cannot be caught within the APL64 public function.

It is the responsibility of the APL64 programmer to assure that:

- An APL64 public function header incorporates .Net argument types which are supported in APL64
- An APL64 public function header incorporates .Net result types which are supported in APL64. In addition, the APL64 result values must be compatible with the function header result types.

Exceptions due to APL64 public function operations

After the conversion of the .Net input data to the APL64 format, an exception may occur within the APL64 public function.

Exceptions which may occur while the APL64 public function is running, are thrown as APL64 exceptions. They can be caught by the APL64 programmer-developed public function using APL64 features such as ⎕ELX, ⎕DM, ⎕EM, and :TRY :CATCHALL :FINALLY. Once an exception is caught it can be handled within the APL64 component or thrown as a .Net exception using ⎕ERROR or ⎕THROW. The text of an exception message should be provided by the APL64 programmer so that is appropriate for the .Net environment.

⎕CpcInfo System Variable

Since an APL64 public function can be run, as a traditional APL programmer-defined function within a purely APL64-based application or run as an APL64 public function in an APL64 cross-platform component used by a .Net application, the 'calling environment' may differ. The APL64 programmer-provided exception message provided when an exception occurs within an APL64 public function should reflect the calling environment.

The APL64 ⎕CpcInfo system variable is available to determine the calling environment of an APL64 public function:

⎕CpcInfo Element#	⎕CpcInfo Element Value
1	Boolean: Running in CPC

The first element of the ⎕CpcInfo system variable may be used to determine if an APL64 public function is running in a cross-platform component. In future versions of APL64 additional elements may be added to the result of the ⎕CpcInfo system variable.

Example:

```
:public double@res←MyPublicFn int@arg
:TRY
 …
 res←                                    …
:CATCHALL
 :IF                              1⊃⎕CpcInfo
  ⎕ERROR   ".Net-style   exception   message"
 :ELSE
  ⎕ERROR   "APL64-style   exception   message"
 :ENDIF
:ENDTRY
```

Global Variables In APL64 Public Functions

Global variables, i.e., those existing outside of the scope of an APL64 public function, may be created, modified, or deleted within an APL64 public function, the same as for any APL64 user-defined function. The values of such global variables are not exposed to a .Net application using an APL64 cross-platform, unless the APL64 programmer creates a public function with the value of a global variable among its results.

Creating an APL64 Cross-platform Component

A Nuget package containing a .Net assembly which exposes a .Net instance class containing APL64 public functions may be created by the APL64 programmer using a utility in the APL64 Developer version. The use of this utility is described in this section by example.

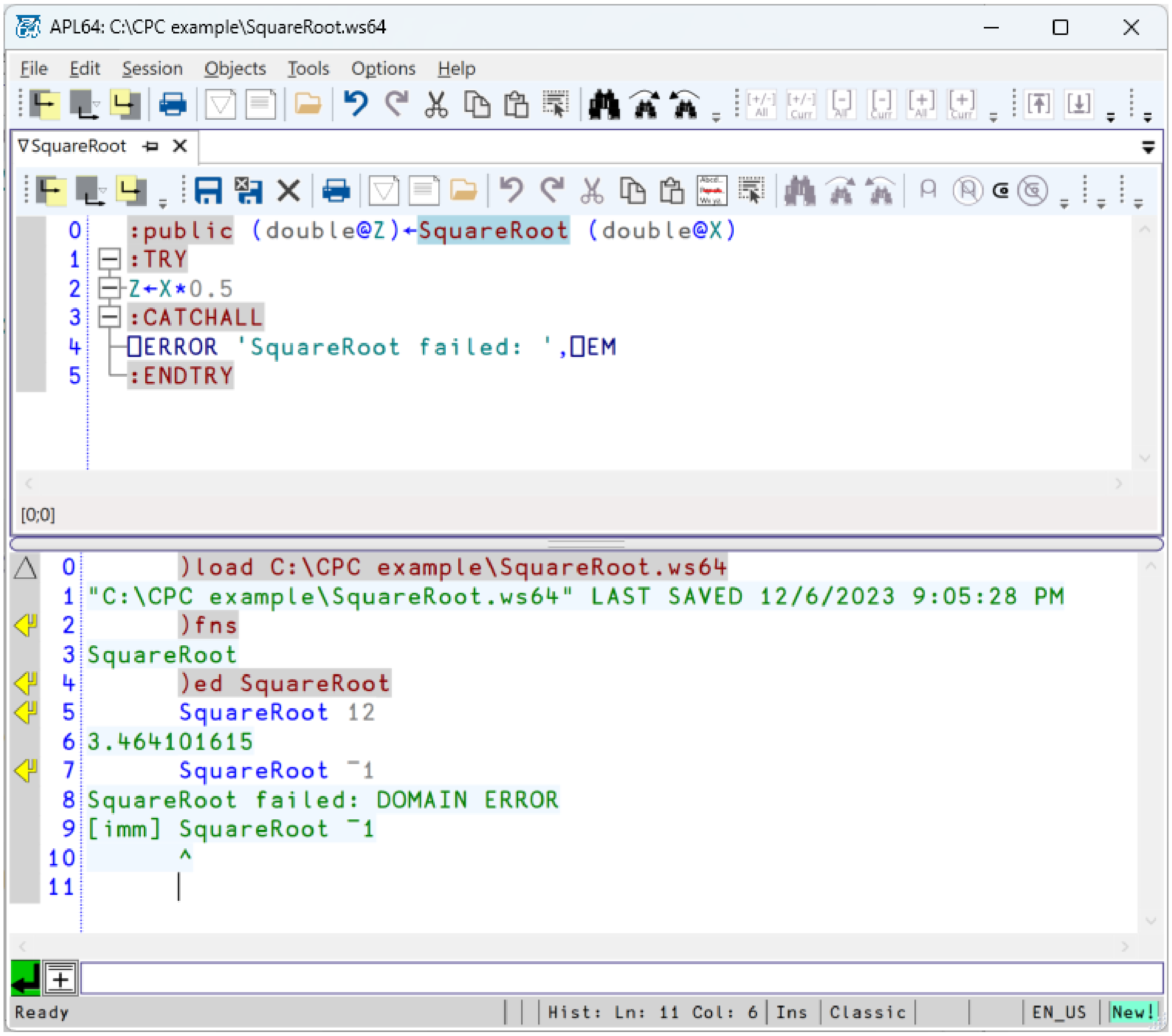
Start an APL64 Developer Version Instance

An APL64 cross-platform component is a ‘runtime’ component which must be created using the APL64 developer version.

Load the APL64 Workspace containing Public Functions

Load an APL64 workspace containing at least one public function and all other functions and variables necessary to support the functionality of the public functions in the workspace.

Verify that the public function is operating properly in APL64.


APL64: C:\CPC example\SquareRoot.ws64
File Edit Session Objects Tools Options Help
∇SquareRoot
[0;0]
Ready
Hist: Ln: 11 Col: 6
Ins
Classic
EN_US
New!

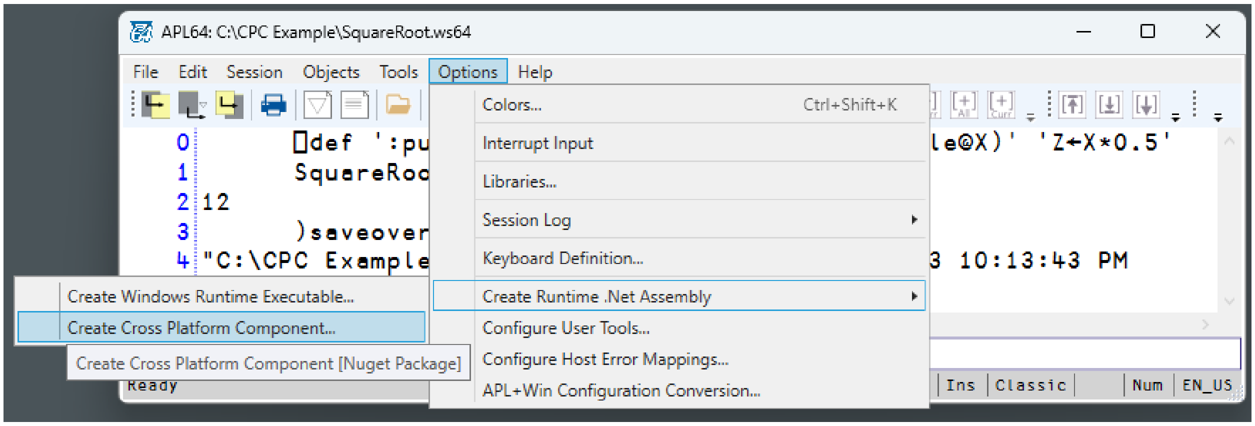
```
:public (double@Z)←SquareRoot (double@X)
:TRY
Z←X*0.5
:CATCHALL
⎕ERROR 'SquareRoot failed: ',⎕EM
:ENDTRY
```

```
      )load C:\CPC example\SquareRoot.ws64
"C:\CPC example\SquareRoot.ws64" LAST SAVED 12/6/2023 9:05:28 PM
      )fns
SquareRoot
      )ed SquareRoot
      SquareRoot 12
3.464101615
      SquareRoot ¯1
SquareRoot failed: DOMAIN ERROR
[imm] SquareRoot ¯1
      ^
```

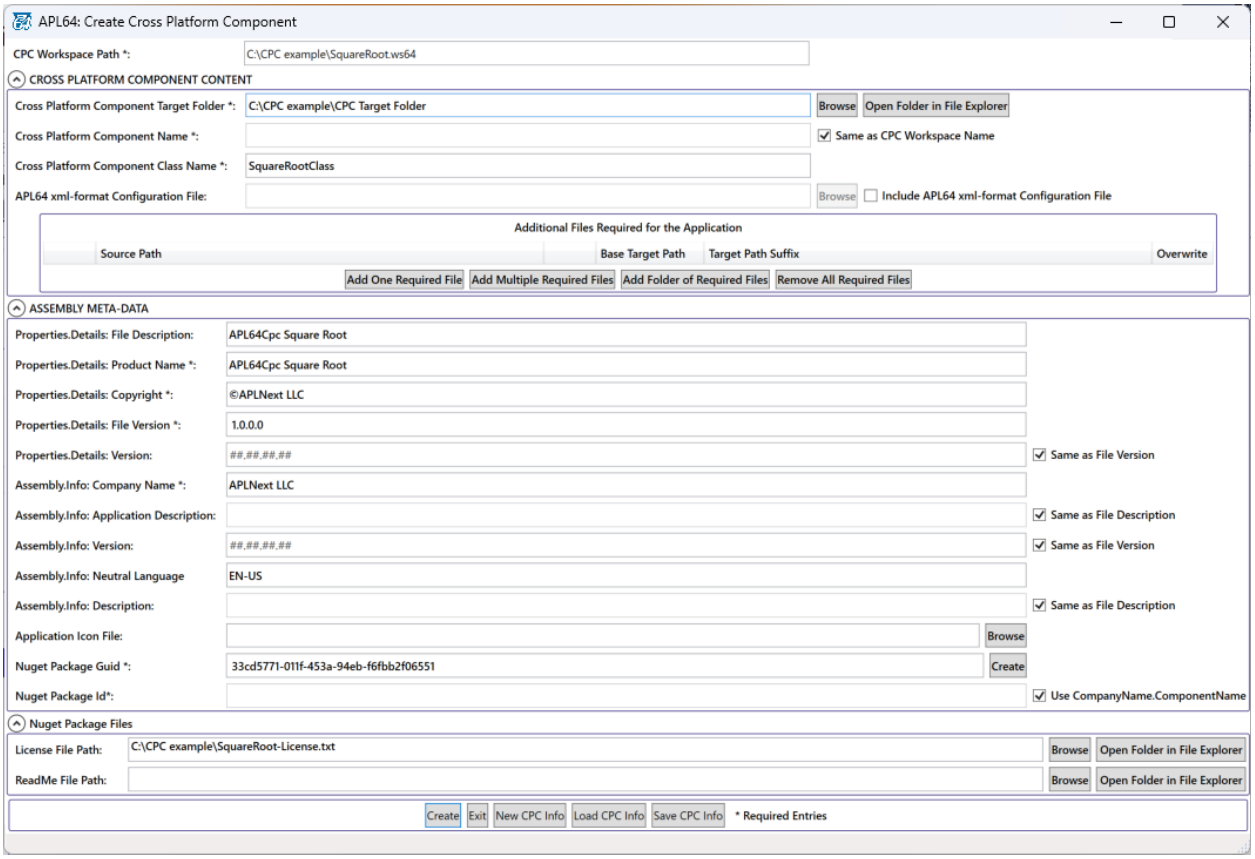
Prepare Auxiliary Files Required for the Component

All additional files necessary to support the public functions should be accessible to the workstation on which the cross-platform component will be created. These additional files can include APL64 workspaces. These workspaces, if any, cannot be loaded from within the component at runtime, but variables and functions can be copied from them at runtime. Public functions copied from a workspace at runtime will not be considered public functions.

Use Options | Create Runtime .Net Assembly | Create Cross-platform Component…


APL64: C:\CPC Example\SquareRoot.ws64
File
Edit
Session
Objects
Tools
Options
Help
Colors... Ctrl+Shift+K
Interrupt Input
Libraries...
Session Log
Keyboard Definition...
Create Runtime .Net Assembly
Configure User Tools...
Configure Host Error Mappings...
APL+Win Configuration Conversion...
Create Windows Runtime Executable...
Create Cross Platform Component...
Create Cross Platform Component [Nuget Package]
0 ⎕def ':pu
1 SquareRoo
2 12
3)saveover
4 "C:\CPC Example
le⍎X)' 'Z←X*0.5'
3 10:13:43 PM
Ready
Ins
Classic
Num
EN_US

Complete the Required Entries in the Dialog:


APL64: Create Cross Platform Component
CROSS PLATFORM COMPONENT CONTENT
Browse
Open Folder in File Explorer
Additional Files Required for the Application
Source Path
Base Target Path
Target Path Suffix
Overwrite
Add One Required File
Add Multiple Required Files
Add Folder of Required Files
Remove All Required Files
ASSEMBLY META-DATA
Nuget Package Files
Create
Exit
New CPC Info
Load CPC Info
Save CPC Info
* Required Entries

CPC Workspace Path *: C:\CPC example\SquareRoot.ws64

Cross Platform Component Target Folder *: C:\CPC example\CPC Target Folder

Cross Platform Component Name *: ☑ Same as CPC Workspace Name

Cross Platform Component Class Name *: SquareRootClass

APL64 xml-format Configuration File: Browse ☐ Include APL64 xml-format Configuration File

Properties.Details: File Description: APL64Cpc Square Root

Properties.Details: Product Name *: APL64Cpc Square Root

Properties.Details: Copyright *: ©APLNext LLC

Properties.Details: File Version *: 1.0.0.0

Properties.Details: Version: ##.##.##.## ☑ Same as File Version

Assembly.Info: Company Name *: APLNext LLC

Assembly.Info: Application Description: ☑ Same as File Description

Assembly.Info: Version: ##.##.##.## ☑ Same as File Version

Assembly.Info: Neutral Language EN-US

Assembly.Info: Description: ☑ Same as File Description

Application Icon File: Browse

Nuget Package Guid *: 33cd5771-011f-453a-94eb-f6fbb2f06551 Create

Nuget Package Id*: ☑ Use CompanyName.ComponentName

License File Path: C:\CPC example\SquareRoot-License.txt

ReadMe File Path:

CPC Workspace Path

This is the full path of the current workspace. This workspace file will be embedded in the CPC. This workspace contains one or more public APL64 programmer-developed functions which will be exposed by the CPC. This field is not editable.

Cross-platform Component Target Folder

This is the folder which will contain the CPC. This folder cannot be the same folder as the folder containing the CPC workspace folder, but it can be a sub-folder of the folder containing the CPC workspace. Each CPC should have separate source and target folders. Using the same source or target folders when creating different CPC can result in a CPC with components which are unnecessary or incorrect. The target folder and its sub-folders should be empty prior to running the CPC creation utility, except for prior versions of the same CPC Nuget package.

Cross-platform Component Name

Enter the desired name of the cross-platform component. A unique name should be selected to distinguish the component. Check the 'Same as CPC Workspace Name' checkbox, if desired, in which case the workspace name without extension will be used as the cross-platform component name. **Note**: .Net does not permit a hyphen in the 'Runtime Executable Name' field. An underscore may be used instead of the hyphen.

The cross-platform component name is case-sensitive. The cross-platform component name will be trimmed of prefixed or suffixed spaces and interior spaces will be replaced by underscore (_).

- The cross-platform component name is used as the name of the .Net assembly (dll) for the Nuget package, so its value must be suitable for a .Net assembly name.
- The cross-platform component name is used in the names of the info and error log files which are created by the APL64 CPC utility.
- The cross-platform component name may be used as the second part (suffix) of the Nuget package id, e.g. 'MyCompany.MyComponentName'.

Cross-platform Class Name

In .Net methods, including APL programmer-defined, public functions exposed as .Net method, must be contained in a .Net class.

Enter the class name for the .Net class which will expose the APL public functions in the resulting Cross-platform component. The class name should be different than the cross-platform component name. The class name must begin with a letter, _ or @ and the remainder of the class name must contain characters, digits or _, but no spaces. The cross-platform component name is case-sensitive. The .Net SDK, which is used by the CPC creation utility to create the Nuget package, restricts the Cross-platform Component Class Name.

APL64 xml-format Configuration File

This is an optional file in the CPC. There is no restriction on the file name of the APL64 xml-format configuration file, but if needed, the file must be accessible to the workstation when the CPC is created. Enter the filename of, or browse to, an APL64 xml-format configuration file on the APL64 programmer's workstation in the 'APL64 xml-format Configuration File field in the CPC creation utility.

- The file will be embedded in the CPC.
- The file will not be materialized at runtime on the end user's workstation, it cannot be examined by the .Net application which consumes the CPC.
- Each time the application is run the file will be read as a memory stream into the APL64 instance.

Because the CPC is designed to be a cross-format component, it is not possible to have command line arguments for the APL64 interpreter included in the component.

Additional Files Required for the Application

Optional components which the APL64 programmer may consider for inclusion in the ‘Additional Files Required for the Application’ section of the CPC:

- APL64 component files required by the application system
- Windows native files required by the application system
- Application-specific documentation files
- Digital signature for the application file
- Activation and licensing wrapper for the component
- Folders of required files

If application-specific files are required for the CPC application, click the ‘Add Required File’ button to add one additional file to the CPC or click the ‘Add Multiple Required Files’ button to add several additional files to the CPC or click the ‘Add Folder of Required Files’ button to add files in a folder and all sub-folders.

APL64 programmer entries required for each ‘Additional File’:

- The ‘Remove’ button will remove the ‘Additional File’ entry in the selected row. This action does not delete the physical file referenced by the CPC entry which is removed.
- The ‘Source Path’ field should contain the full path of the existing file on the APL64 programmer’s workstation to be included in the CPC.
- The ‘Browse’ button facilitates completing the ‘Source Path’ field.
- The file will be materialized to the end user’s workstation with file name specified by the APL64 programmer. The ‘Source Path’ does not have to match the desired file name of the end user’s workstation.
    - If the ‘Base Target Path’ is selected as ‘Custom Path’ by the APL64 programmer, the ‘Target Path Suffix’ field must be the full file name of the file on the end user’s workstation.
- If the ‘Base Target Path’ is selected as ‘Executable Path’ by the APL64 programmer, the target file name on the end user’s workstation will be the concatenation of the selected ‘Base Target Path’ and the APL64 programmer entry for the ‘Target Path Suffix’ field.
- The ‘Overwrite’ check box
    - If checked, the additional file is materialized to the target location and overwrites a pre-existing file in that location with the same name.
    - If not checked, the additional file is materialized to the target location only if there is no pre-existing file in that location with the same name.
- Target location on the end user’s workstation

    An APL64 programmer-specified file is embedded into the CPC. At runtime, the file is materialized on the end user’s workstation at the APL64 programmer-specified target location.

    The target location is specified using the ‘Base Target Path’ field of the CPC creation utility.

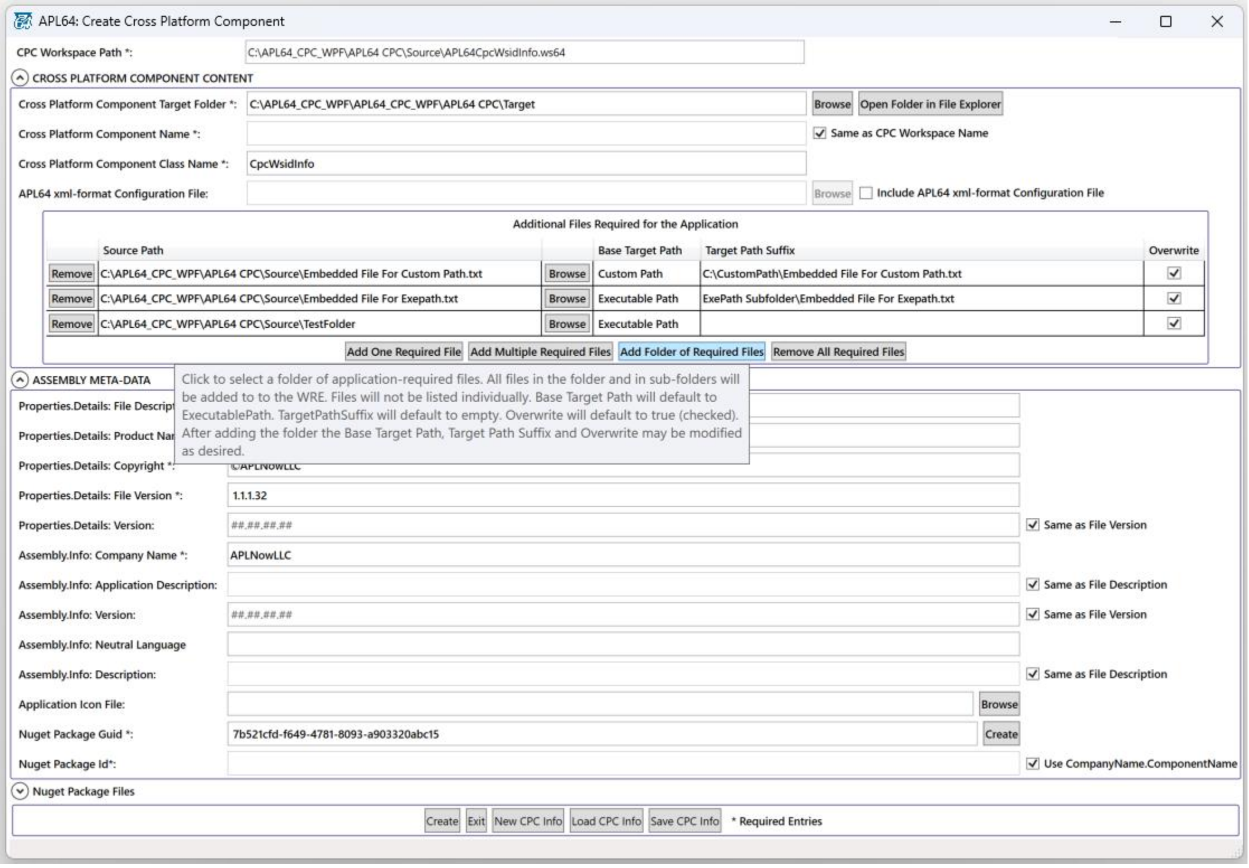
Executable Path

The ‘ExecutablePath’ is operating-system-determined and provides read/write access for activities of the end user of the workstation. This path may be obtained from within the APL64-based application using the ⎕EXEPATH system function. This path is determined by the .Net application which consumes the CPC.

Custom Path

This location option provides complete flexibility for the APL64 programmer.

Clicking the 'Add Folder of Required Files' will present the Folder Browser. Navigate to the selected folder and select it. The folder, all sub-folders and all contained files in the folders. The Base Target Path, Target Path Suffix, and Overwrite fields may be modified after the folder is added to the CPC.


APL64: Create Cross Platform Component
Browse
Open Folder in File Explorer
Browse
Browse
Add One Required File
Add Multiple Required Files
Add Folder of Required Files
Remove All Required Files
Browse
Create
Create
Exit
New CPC Info
Load CPC Info
Save CPC Info

CPC Workspace Path *: C:\APL64_CPC_WPF\APL64 CPC\Source\APL64CpcWsidInfo.ws64

CROSS PLATFORM COMPONENT CONTENT

Cross Platform Component Target Folder *: C:\APL64_CPC_WPF\APL64_CPC_WPF\APL64 CPC\Target

Cross Platform Component Name *: ☑ Same as CPC Workspace Name

Cross Platform Component Class Name *: CpcWsidInfo

APL64 xml-format Configuration File: □ Include APL64 xml-format Configuration File

Additional Files Required for the Application

	Source Path		Base Target Path	Target Path Suffix	Overwrite
Remove	C:\APL64_CPC_WPF\APL64 CPC\Source\Embedded File For Custom Path.txt	Browse	Custom Path	C:\CustomPath\Embedded File For Custom Path.txt	☑
Remove	C:\APL64_CPC_WPF\APL64 CPC\Source\Embedded File For Exepath.txt	Browse	Executable Path	ExePath Subfolder\Embedded File For Exepath.txt	☑
Remove	C:\APL64_CPC_WPF\APL64 CPC\Source\TestFolder	Browse	Executable Path		☑

ASSEMBLY META-DATA

Click to select a folder of application-required files. All files in the folder and in sub-folders will be added to to the WRE. Files will not be listed individually. Base Target Path will default to ExecutablePath. TargetPathSuffix will default to empty. Overwrite will default to true (checked). After adding the folder the Base Target Path, Target Path Suffix and Overwrite may be modified as desired.

Properties.Details: File Descript

Properties.Details: Product Nar

Properties.Details: Copyright *: ©APLNowLLC

Properties.Details: File Version *: 1.1.1.32

Properties.Details: Version: ##.##.##.## ☑ Same as File Version

Assembly.Info: Company Name *: APLNowLLC

Assembly.Info: Application Description: ☑ Same as File Description

Assembly.Info: Version: ##.##.##.## ☑ Same as File Version

Assembly.Info: Neutral Language

Assembly.Info: Description: ☑ Same as File Description

Application Icon File:

Nuget Package Guid *: 7b521cfd-f649-4781-8093-a903320abc15

Nuget Package Id*: ☑ Use CompanyName.ComponentName

Nuget Package Files

* Required Entries

Materializing Files Embedded in a CPC to the End User Machine

Application-required files embedded in a CPC are not automatically materialized to the end user's machine. When an APL64 CPC contains embedded application-required files, the MaterializeAdditionalFiles() method in the APL64 CPC class may be used to materialize these files to the target machine at runtime. It is up to the designer/programmer of the application which uses the CPC to use (or not use) the MaterializeAdditionalFIles() method in the application.

An APL64 CPC exposes APL64 public functions as .Net methods which can be used in many types of application systems including:

- Windows desktop applications
- Server-side web services for multiple client users
- Phone and Tablet applications using .Net-compatible, non-Windows operating systems

To use an APL64 public function in an APL64 CPC, a class instance must be created. In applications, such as web services, many instances of this APL64 CPC class may be simultaneously created and used by different users.

The developer of the application which uses an APL64 CPC must carefully consider if or when to use the MaterializeAdditionalFiles() method of the CPC class to avoid conflicts when multiple users are accessing the materialized files on the target machine.

There are two overloads of the MaterializeAdditionalFiles() method:

MaterializeAdditionalFiles() method

Result: bool: True indicates successful materialization of all files and folders which could be materialized

Arguments:

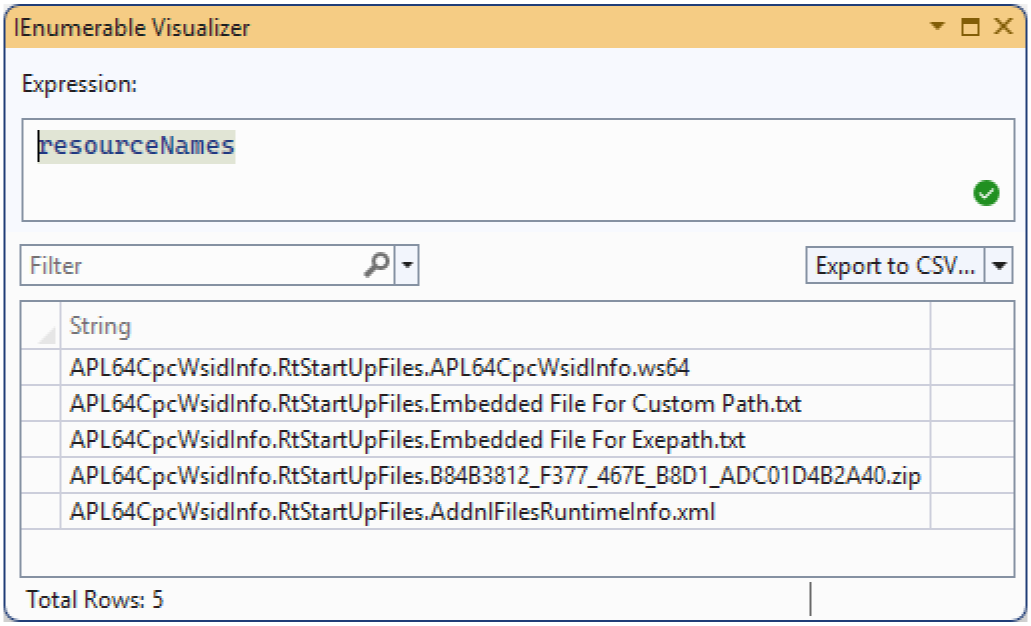
- out string[] materializedFiles
  The 'materializedFiles' out parameter result is an array of strings containing the full paths of the materialized files on the target machine.
- out string errMsg
  The 'errMsg' out parameter is an empty string if the materialization is successful, otherwise an exception message is returned.

Extended MaterializeAdditionalFiles() method

Result: bool: True indicates successful materialization of all selected files and folders which could be materialized

Arguments:

- out string[] materializedFiles
  The 'materializedFiles' out parameter result is an array of strings containing the full paths of the materialized files on the target machine.
- out string errMsg
  The 'errMsg' out parameter is an empty string if the materialization is successful, otherwise an exception message is returned.
- out string[] resourceNames
  An array of resource names which can be materialized

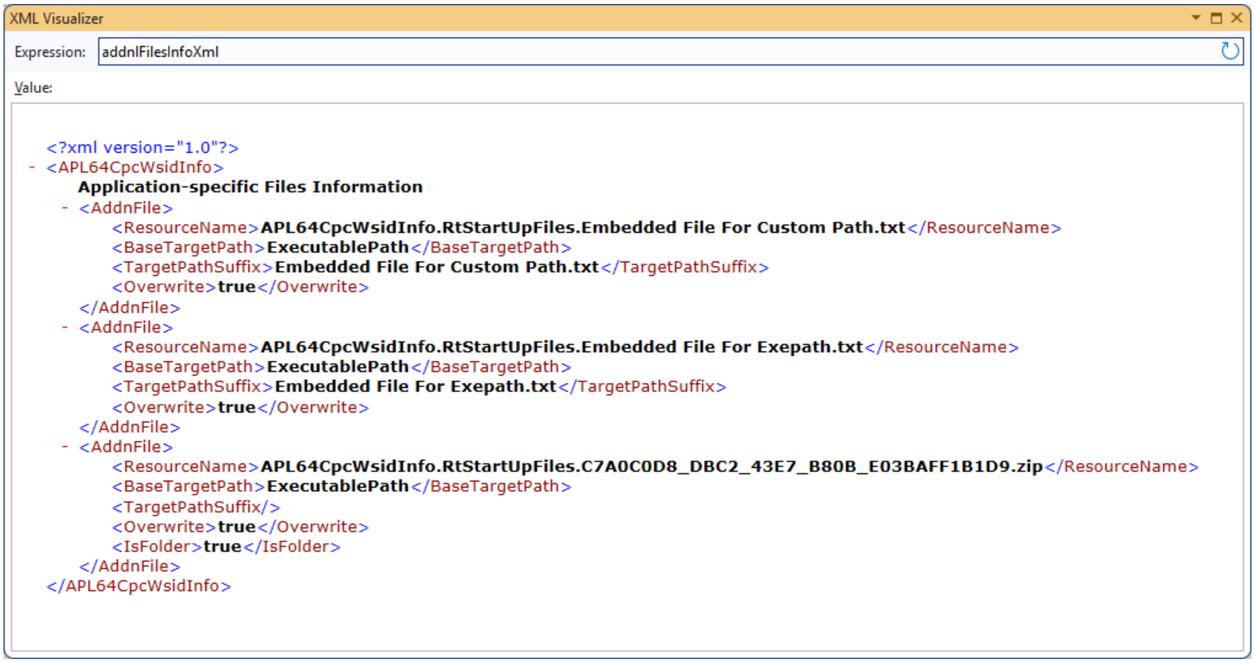


```
resourceNames
```

String
APL64CpcWsidInfo.RtStartUpFiles.APL64CpcWsidInfo.ws64
APL64CpcWsidInfo.RtStartUpFiles.Embedded File For Custom Path.txt
APL64CpcWsidInfo.RtStartUpFiles.Embedded File For Exepath.txt
APL64CpcWsidInfo.RtStartUpFiles.B84B3812_F377_467E_B8D1_ADC01D4B2A40.zip
APL64CpcWsidInfo.RtStartUpFiles.AddnlFilesRuntimeInfo.xml

Total Rows: 5

- out string resourceInfoXml
  An xml-format string indicating the materialization information for each of the available resources



addnlFilesInfoXml

```
  <?xml version="1.0"?>
- <APL64CpcWsidInfo>
     Application-specific Files Information
   - <AddnFile>
        <ResourceName>APL64CpcWsidInfo.RtStartUpFiles.Embedded File For Custom Path.txt</ResourceName>
        <BaseTargetPath>ExecutablePath</BaseTargetPath>
        <TargetPathSuffix>Embedded File For Custom Path.txt</TargetPathSuffix>
        <Overwrite>true</Overwrite>
     </AddnFile>
   - <AddnFile>
        <ResourceName>APL64CpcWsidInfo.RtStartUpFiles.Embedded File For Exepath.txt</ResourceName>
        <BaseTargetPath>ExecutablePath</BaseTargetPath>
        <TargetPathSuffix>Embedded File For Exepath.txt</TargetPathSuffix>
        <Overwrite>true</Overwrite>
     </AddnFile>
   - <AddnFile>
        <ResourceName>APL64CpcWsidInfo.RtStartUpFiles.C7A0C0D8_DBC2_43E7_B80B_E03BAFF1B1D9.zip</ResourceName>
        <BaseTargetPath>ExecutablePath</BaseTargetPath>
        <TargetPathSuffix/>
        <Overwrite>true</Overwrite>
        <IsFolder>true</IsFolder>
     </AddnFile>
  </APL64CpcWsidInfo>
```

To obtain the resourceNames and addnlFilesInfoXml information for a CPC without materializing any resources of the CPC, set the selectedFiles argument to new List<string>(){“*”}. The materializedFiles result will be empty.

Assembly Meta-data

The 'ASSEMBLY META-DATA' information will be visible to the end user of the APL64 cross-platform component. This data includes two types of entries:

- The 'Properties.Details' entries affect the information exposed by the operating system file system.

    - File Description: Optional text value.
    - Product Name: Required text value.
    - Copyright: Required text value indicating the entity holding the copyright. This text will be suffixed by: - `Portions © APLNow, LLC & APLNext, LLC`.
    - File Version: Required value in the format '##.##.##.##, where '#' indicates a digit. Leading zeros are not required, e.g. 1.2.3.4 and 01.02.03.04 are both acceptable. The file version field should be updated by the APL64 programmer before creating a new version of a cross-platform component.
    - Version: Optional value if 'Same as File Version' is checked, otherwise in the format '##.##.##.##, where '#' indicates a digit.
- The 'Assembly.Info' entries affect the information exposed when the CPC package is examined by .Net methods.
    - Company Name: This required value is included in the .Net assembly information. This value may be used as the first part (prefix) of the Nuget package id created by the APL64 CPC utility, e.g. 'MyCompanyName.MyComponentName'.
    - Application Description: Optional text value
    - Version: Optional Assembly Version value. if 'Same as File Version' is checked the File Version values will be used, otherwise enter the desired value in the format '##.##.##.##, where '#' indicates a digit.
    - Neutral Language: The language for the CPC, e.g. or 'Neutral Language' or 'English (United States)'.
    - Description: Optional application description text.
    - Icon File: Optional path to the application icon file provided by the APL64 programmer. This icon will apply to the CPC packaged created by this utility, but not to the end user application which uses the CPC package.Save the Component Information
    - Nuget Package Guid: A required, globally-unique Guid. If necessary, use the Create button to fill this field. This value is established once for a new cross-platform component. This value should not be changed when subsequent versions of the same component are created.
    - Nuget Package Id: A required text value used as the Nuget package id. The Nuget package guid uniquely distinguishes a Nuget package, but unique Nuget package id should be selected. Often the Nuget package id is composed of a company name prefix and package name suffix. A Nuget package id prefix can be reserved. Nuget requirements for a package id must be considered.

      Click the 'Use CompanyName.ComponentName' checkbox in the CPC utility dialog to use the APL64 programmer-provided company name and cross-platform component name field values as the Nuget package id.

      If invalid characters are included in the Nuget package id by the APL64 programmer, the CPC utility output log file will contain a message analogous to:

      ```
      The package ID ' My Company, LLC.SquareRoot' contains invalid characters. Examples of valid package IDs include 'MyPackage' and 'MyCompany.MyPackage'
      ```

- ○ Click the 'Save CPC Info' button to save the cross-platform component configuration information for future use.

Nuget Package Files

The APL64 cross-platform component is packaged in Nuget format. A Nuget package should contain several files prepared by the APL64 programmer. Complete this section by entering, or browsing to, the full path of the files for: License, ReadMe.

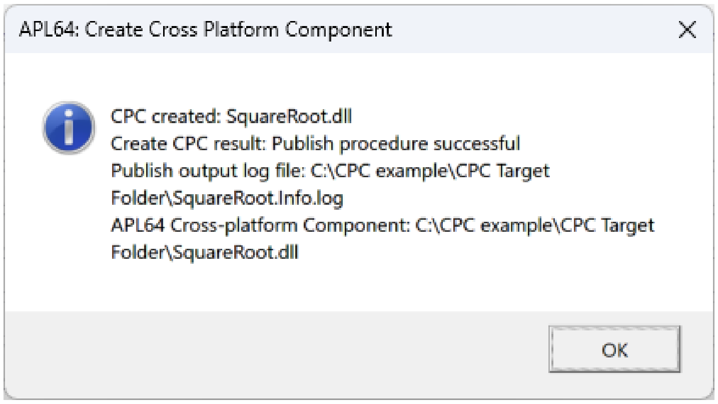
Create the Component

Click the 'Create' button to start the cross-platform component creation process. The CPC creation method uses the .Net Core 'Publish' method to prepare the CPC component in the specified folder on the APL64 programmer's workstation.

If the CPC is successfully created, a message box will indicate:

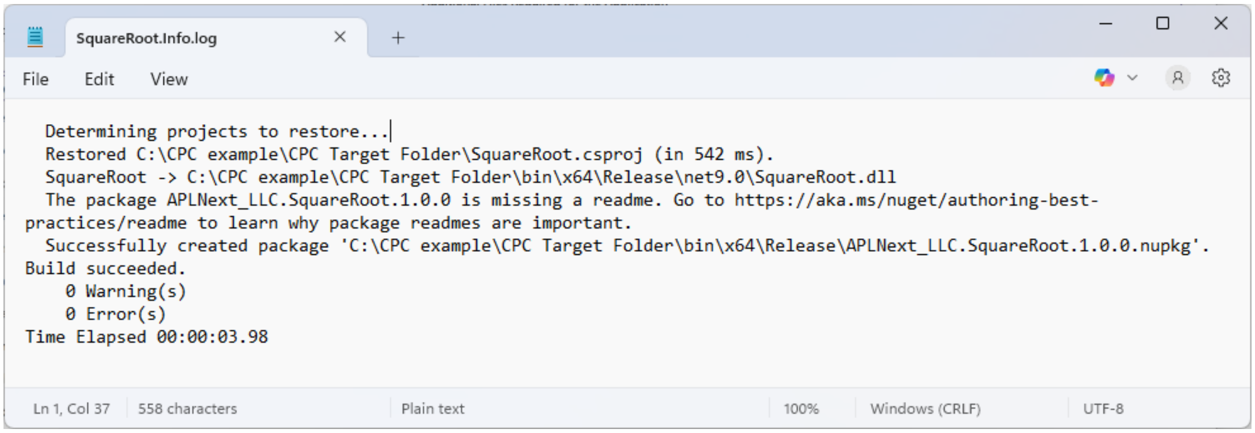
- Success of the process
- Publish log file name in the CPC output folder
- CPC Nuget package name in the CPC output folder

The CPC output folder will contain the '…Info.log' file. This file may contain warnings even if the CPC is successfully created and will create exceptions if the CPC is not successfully created.



CPC created: SquareRoot.dll
Create CPC result: Publish procedure successful
Publish output log file: C:\CPC example\CPC Target Folder\SquareRoot.Info.log
APL64 Cross-platform Component: C:\CPC example\CPC Target Folder\SquareRoot.dll

In this example, the CPC creation was successful as indicated by the log file in the target folder:



```
  Determining projects to restore...
  Restored C:\CPC example\CPC Target Folder\SquareRoot.csproj (in 542 ms).
  SquareRoot -> C:\CPC example\CPC Target Folder\bin\x64\Release\net9.0\SquareRoot.dll
  The package APLNext_LLC.SquareRoot.1.0.0 is missing a readme. Go to https://aka.ms/nuget/authoring-best-
practices/readme to learn why package readmes are important.
  Successfully created package 'C:\CPC example\CPC Target Folder\bin\x64\Release\APLNext_LLC.SquareRoot.1.0.0.nupkg'.
Build succeeded.
    0 Warning(s)
    0 Error(s)
Time Elapsed 00:00:03.98
```

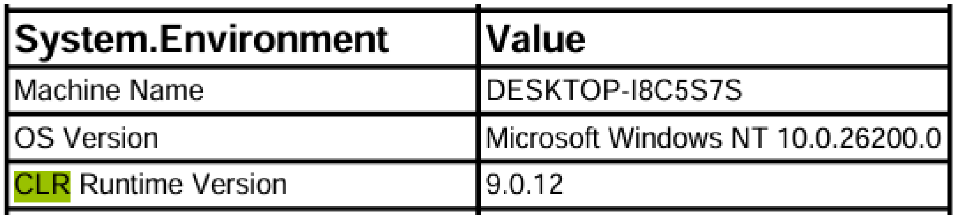
Test the Component from a .Net Application

Using Microsoft Visual Studio, Microsoft Visual Studio Code (free) or Microsoft Visual Studio Community Edition (free), create a simple .Net project which references the APL64 component Nuget package and call the public functions in the APL64 component as .Net methods.

In a production environment, the APL64 component Nuget package mayl be published on-line so consumers can reference it in their .Net projects. In a test environment, the same Nuget package can be located on the test workstation and used in a Visual Studio test project.

Determine the .Net Version

Determine the .Net version used to create the APL64 Nuget package. Open an instance of the APL64 developer version, click on the **Help | About APL64 | System Information** button and review the System Information.pdf document for the CLR Runtime Version.

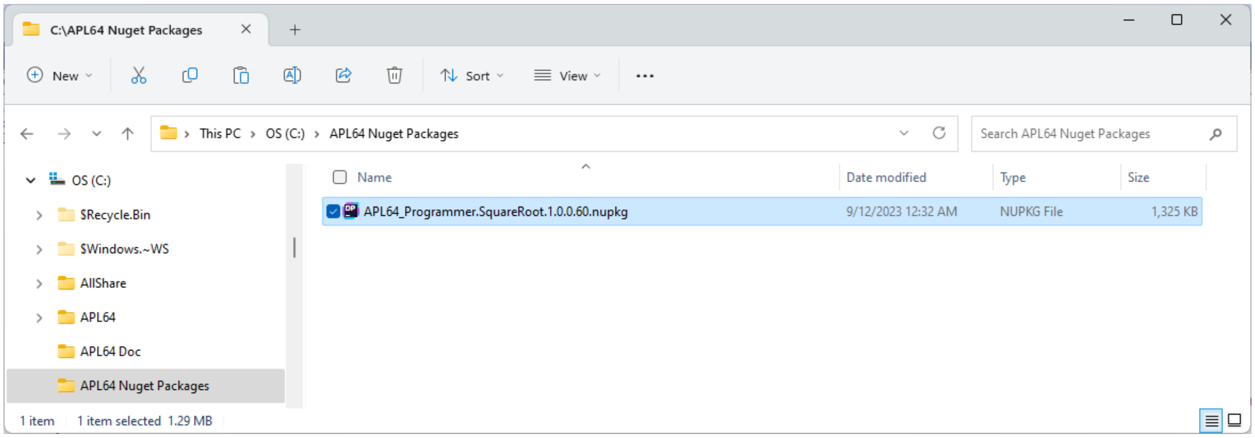


System.Environment	Value
Machine Name	DESKTOP-I8C5S7S
OS Version	Microsoft Windows NT 10.0.26200.0
CLR Runtime Version	9.0.12

Publish the CPC Nuget Package Locally

Create a workstation folder to contain the APL64 Nuget package, e.g. c:\APL64 Nuget Packages\.

Copy the APL64 Nuget package file to the workstation folder:

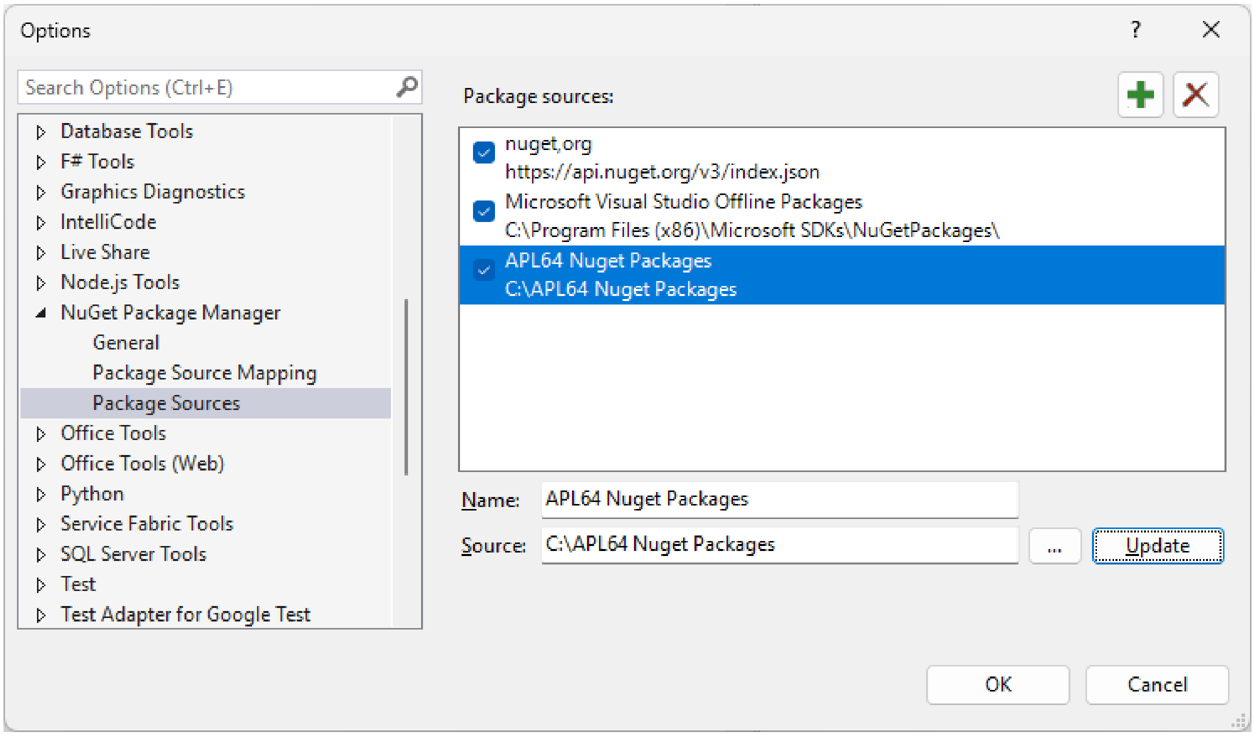


Name	Date modified	Type	Size
APL64_Programmer.SquareRoot.1.0.0.60.nupkg	9/12/2023 12:32 AM	NUPKG File	1,325 KB

1 item 1 item selected 1.29 MB

Open an instance of Microsoft Visual Studio that is compatible with the version of .Net used to create the APL64 Nuget package.

In Visual Studio use **Tools | Nuget Package Manager | Package Manager Settings | Package Sources | + (Add package source)** to provide the folder of the APL64 Nuget packages:



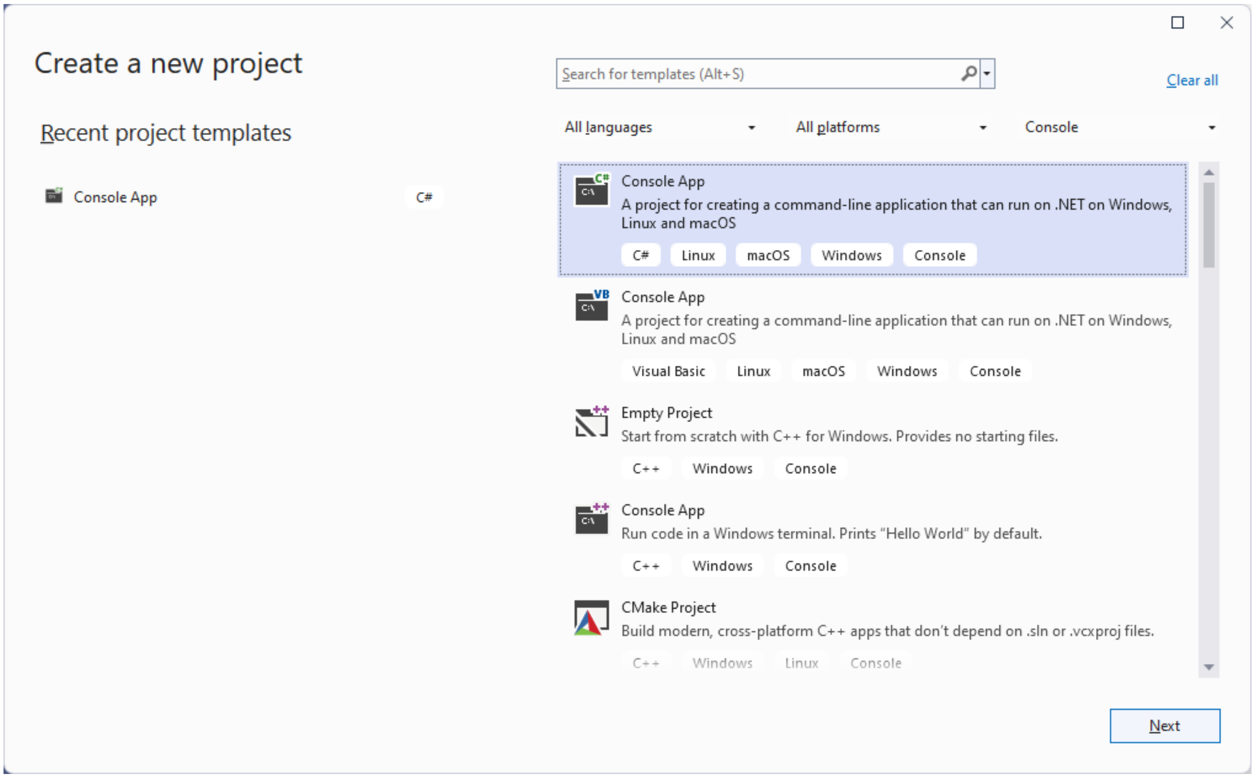
☑ nuget.org
https://api.nuget.org/v3/index.json
☑ Microsoft Visual Studio Offline Packages
C:\Program Files (x86)\Microsoft SDKs\NuGetPackages\
☑ APL64 Nuget Packages
C:\APL64 Nuget Packages

Name: APL64 Nuget Packages

Source: C:\APL64 Nuget Packages

Create a C# .Net Project to Consume the CPC Nuget Package

Create a Visual Studio project targeting the .Net version identified above which will consume the APL64 Nuget package.



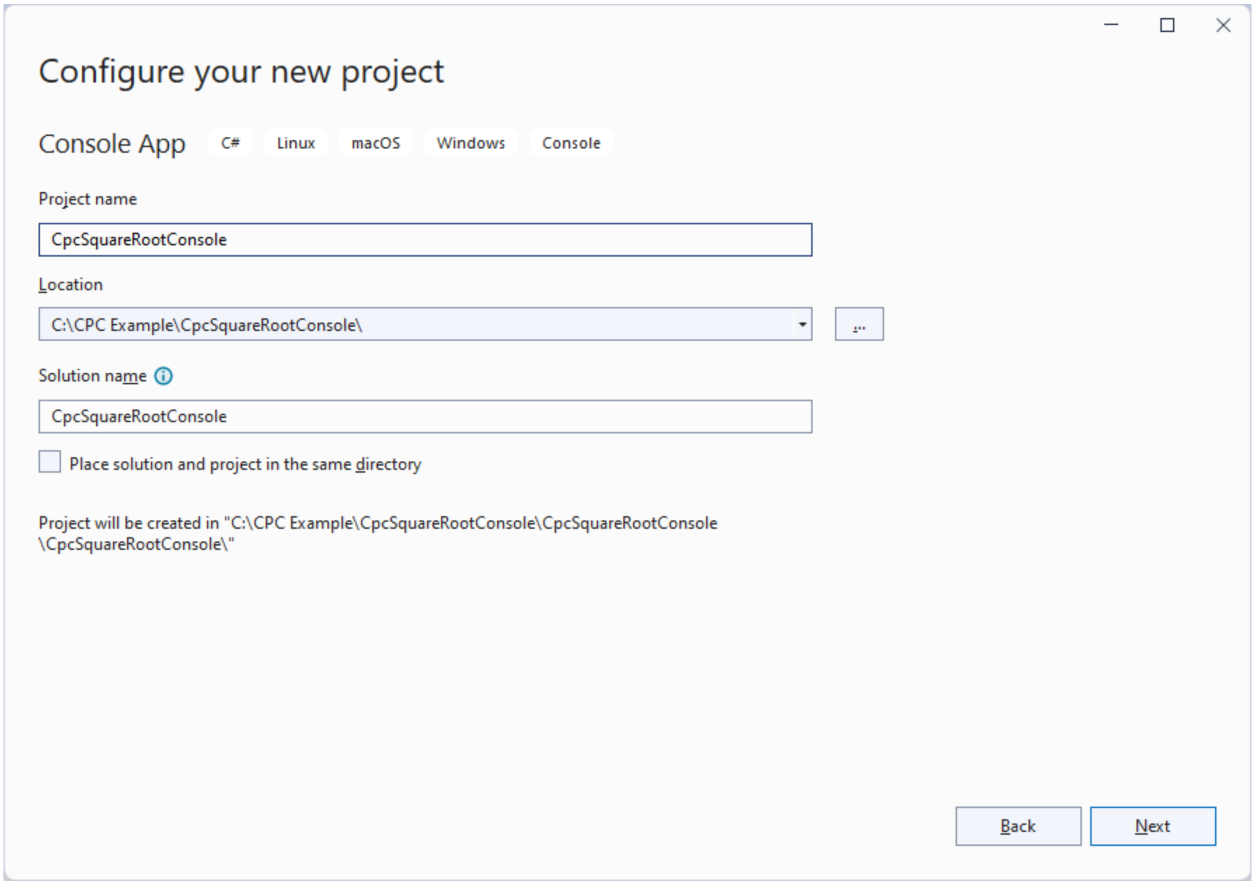
Console App
A project for creating a command-line application that can run on .NET on Windows, Linux and macOS
C# Linux macOS Windows Console

Console App
A project for creating a command-line application that can run on .NET on Windows, Linux and macOS
Visual Basic Linux macOS Windows Console

Empty Project
Start from scratch with C++ for Windows. Provides no starting files.
C++ Windows Console

Console App
Run code in a Windows terminal. Prints "Hello World" by default.
C++ Windows Console

CMake Project
Build modern, cross-platform C++ apps that don't depend on .sln or .vcxproj files.
C++ Windows Linux Console



Configure your new project

Console App

Project name

CpcSquareRootConsole

Location

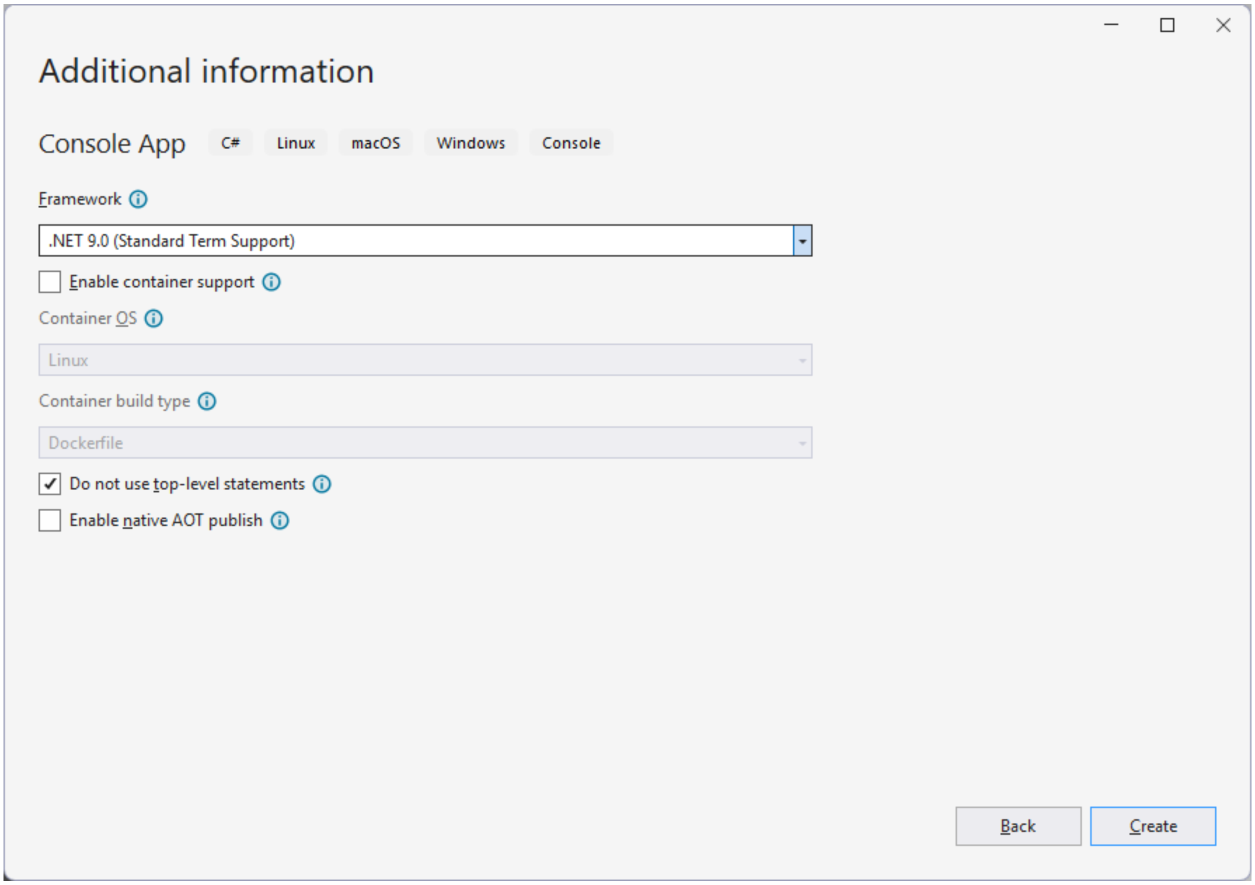
C:\CPC Example\CpcSquareRootConsole\

Solution name

CpcSquareRootConsole

□ Place solution and project in the same directory

Project will be created in "C:\CPC Example\CpcSquareRootConsole\CpcSquareRootConsole\CpcSquareRootConsole\"



Additional information

Console App

Framework

.NET 9.0 (Standard Term Support)

☐ Enable container support

Container OS

Linux

Container build type

Dockerfile

☑ Do not use top-level statements

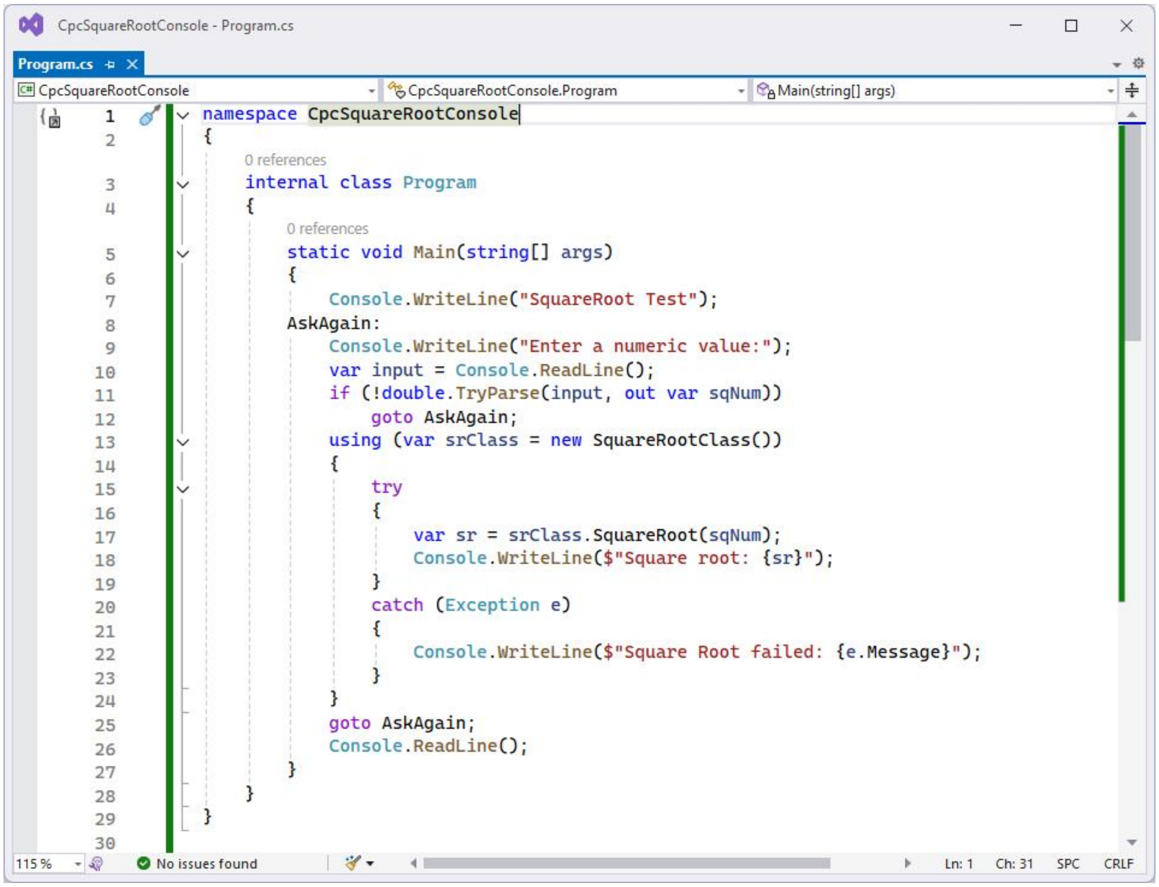
☐ Enable native AOT publish

Add the C# source code to the project, being sure to provide the required input as variable values or via an input mechanism.

In this example:

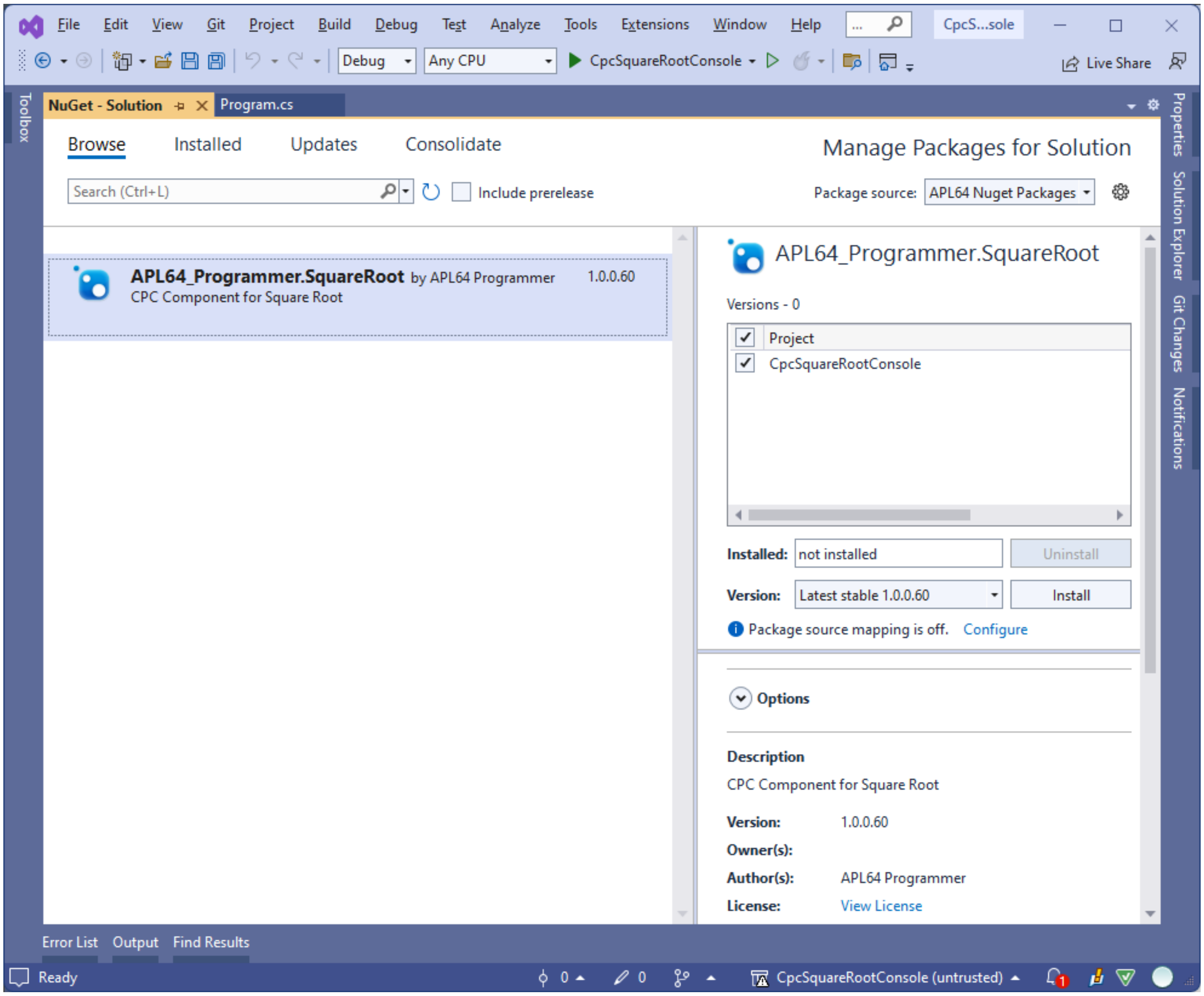
- The source code line var input = Console.ReadLine(); obtains the required input from the user of the C# program.
- The source code line var srClass = new SquareRoot.SquareRootClass(); creates an instance of the SquareRoot class contained in the CPC.
- The source code line var sr = srClass.SquareRoot(sqNum); calls the SquareRoot method, an APL64 public function, with the argument obtained by the C# program.
- The source code lines Console.WriteLine($"Square root… display the successful or failed results to the user of the CPC program.
- The source code try/catch/finally structure enables the C# program to distinguish successful output from failed (exception) output. Exception output may occur if the APL64 programmer's APL64 public function validates improper input or if an unanticipated exception occurs in that APL64 public function.

```
namespace CpcSquareRootConsole
{
   internal class Program
   {
        static void Main(string[] args)
        {
            Console.WriteLine("SquareRoot Test");
        AskAgain:
            Console.WriteLine("Enter a numeric value:");
            var input = Console.ReadLine();
            if (!double.TryParse(input, out var sqNum))
                goto AskAgain;
            using (var srClass = new SquareRootClass())
            {
                try
                {
                    var sr = srClass.SquareRoot(sqNum);
                    Console.WriteLine($"Square root: {sr}");
                }
                catch (Exception e)
                {
                    Console.WriteLine($"Square Root failed: {e.Message}");
                }
            }
            goto AskAgain;
            Console.ReadLine();
        }
    }
}
```



```
namespace CpcSquareRootConsole
{
    internal class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("SquareRoot Test");
        AskAgain:
            Console.WriteLine("Enter a numeric value:");
            var input = Console.ReadLine();
            if (!double.TryParse(input, out var sqNum))
                goto AskAgain;
            using (var srClass = new SquareRootClass())
            {
                try
                {
                    var sr = srClass.SquareRoot(sqNum);
                    Console.WriteLine($"Square root: {sr}");
                }
                catch (Exception e)
                {
                    Console.WriteLine($"Square Root failed: {e.Message}");
                }
            }
            goto AskAgain;
            Console.ReadLine();
        }
    }
}
```

Add the CPC Nuget Package to the Project References using Tools | Nuget Package Manager | Manage Nuget Packages for Solution… and click Install:



Manage Packages for Solution

Package source: APL64 Nuget Packages

APL64_Programmer.SquareRoot by APL64 Programmer 1.0.0.60
CPC Component for Square Root

APL64_Programmer.SquareRoot

Versions - 0

☑ Project
☑ CpcSquareRootConsole

Installed: not installed

Version: Latest stable 1.0.0.60

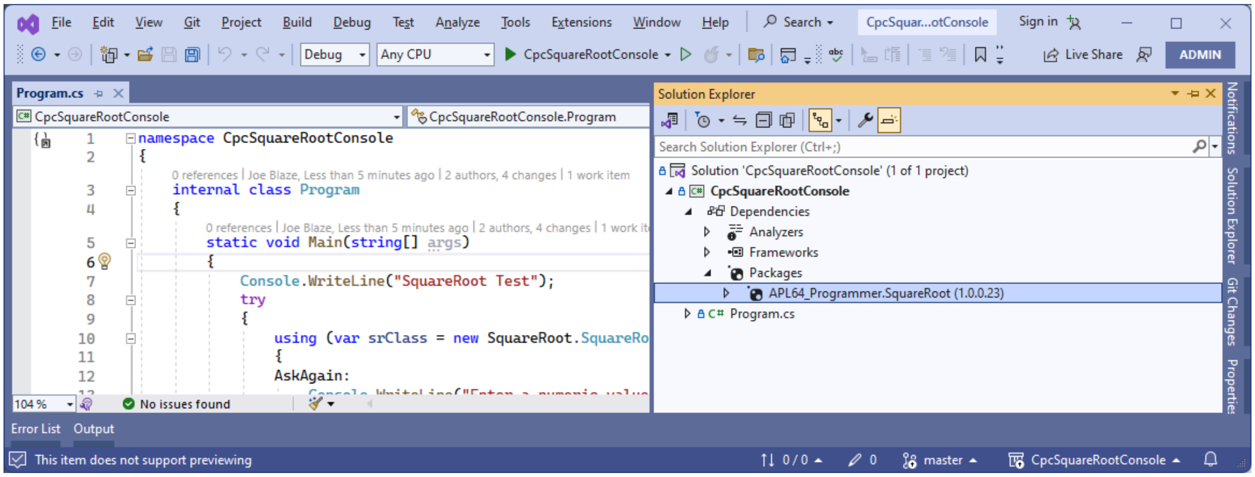
Package source mapping is off.

Description
CPC Component for Square Root

Version: 1.0.0.60
Owner(s):
Author(s): APL64 Programmer
License: View License

CpcSquareRootConsole (untrusted)

Check the project dependencies to see the APL64 Nuget package is included:



```
namespace CpcSquareRootConsole
{
    0 references | Joe Blaze, Less than 5 minutes ago | 2 authors, 4 changes | 1 work item
    internal class Program
    {
        0 references | Joe Blaze, Less than 5 minutes ago | 2 authors, 4 changes | 1 work it
        static void Main(string[] args)
        {
            Console.WriteLine("SquareRoot Test");
            try
            {
                using (var srClass = new SquareRoot.SquareRo
                {
                AskAgain:
```

No issues found

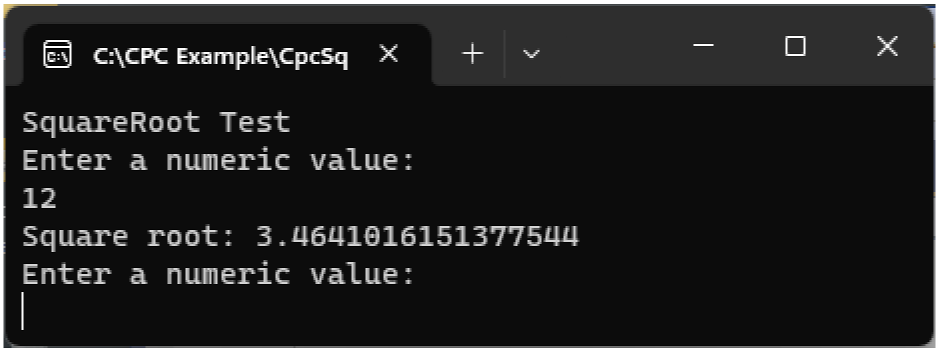
Solution 'CpcSquareRootConsole' (1 of 1 project)
- CpcSquareRootConsole
  - Dependencies
    - Analyzers
    - Frameworks
    - Packages
      - APL64_Programmer.SquareRoot (1.0.0.23)
  - Program.cs

This item does not support previewing

0 / 0 0 master CpcSquareRootConsole

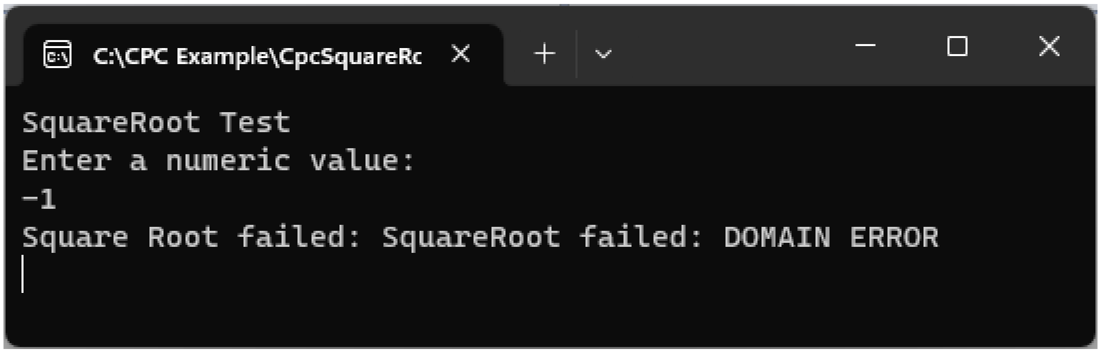
Run the Test Project

Enter the required input and verify the results:



```
SquareRoot Test
Enter a numeric value:
12
Square root: 3.4641016151377544
Enter a numeric value:
```

If the APL64 public function throws an exception because it validates the user entered arguments or the public function unexpectedly fails, the exception provided by that APL64 public function can be captured using a .Net Try/Catch block and displayed to the user:



```
SquareRoot Test
Enter a numeric value:
-1
Square Root failed: SquareRoot failed: DOMAIN ERROR
```

Deploying the Component to a Consumer

The APL64 component is a Nuget package which can be provided as a file to a .Net consumer. Alternatively, the Nuget package can be privately or publicly published to a Nuget repository.

Special Cross-Platform Considerations

Public Functions are Recognized only when the Component is Created

Public functions cannot be created or re-defined within a component after it has been created. Public functions copied to the workspace at runtime, i.e. after the component has been created, will be recognized and can be used as APL programmer-defined functions, but they will not be recognized as public functions.

Auxiliary Workspaces can be copied but not loaded from within a Component

If necessary, a component can include auxiliary workspaces. At runtime, variables and functions can be copied from such included workspaces from within the component.

File Access from within the Component must consider the Operating System Platform

File support, such as naming conventions, volume and disk identification as well as case sensitivity vary with the operating system platform and must be properly handled by the APL64 programmer creating the component. Check the documentation of the APL64 system variable ⎕Pathcase for additional details.

Windows-only Legacy Features cannot be included in a Component

Legacy features of APL64, such as ⎕WI, ⎕WCALL and ActiveX, cannot be used in an APL64 cross-platform component because they are supported only in the Windows environment.

Updating an APL64 Cross-platform Component

When the APL64 source code for a component must be updated, the APL64 CPC must be recreated using the APL64 Developer version. Update the version number of the CPC when it is created.

Nuget Package Best Practices

See here for more information.

⎕CPC Cross-platform Component System Function

Purpose:
Specified actions of the Cross-platform Component (CPC) creation utility in the APL64 developer version may be performed under program control using the ⎕CPC system function. These actions are performed using the memory-based, CPC configuration data set in the current APL64 developer version instance.

The CPC dialogue and the ⎕CPC system function share the same current CPC data set. The current CPC data set remains available during the current APL64 developer version instance but is not automatically saved when the current APL64 developer version instance ends. If appropriate, use the ⎕CPC system function or the CPC dialogue to save the current CPC data set for future APL64 developer instances with a programmer-selected file name.

To use an APL64 Cross-platform component, the consumer of the CPC must be a .Net application which references the CPC Nuget package and utilizes the APL64 programmer-defined public function(s) exposed by the CPC.

The ⎕CPC system function is available only in the APL64 Developer version.

Syntax: result ← action ⎕cpc argument

Arguments:
action is the action to be performed. Action argument text is case-insensitive.

argument is the required argument to the action.

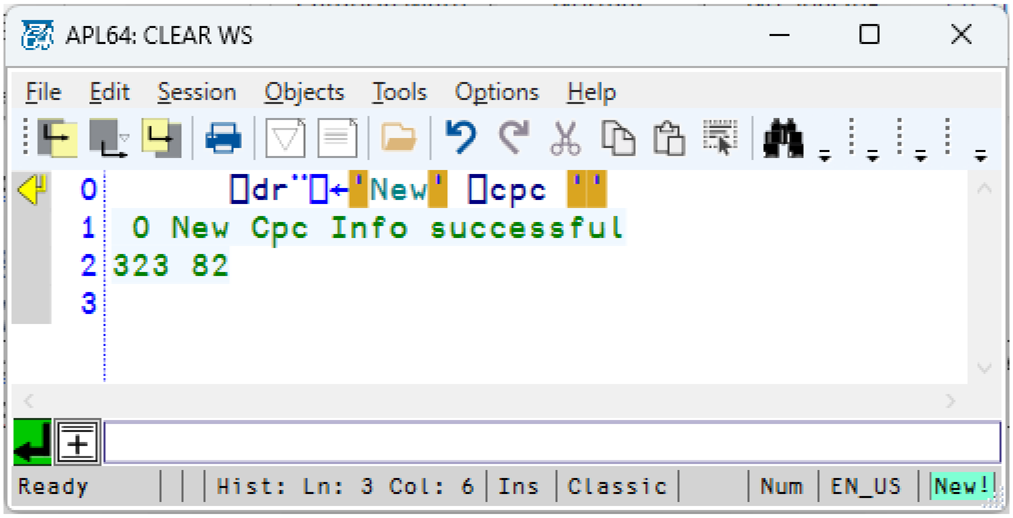
Result:
result is a two-element variable where the first element is an integer: 0 = success, 1 = failure. The second element is a character vector describing the action result.

⎕CPC Actions

⎕CPC New Action

Syntax: 'New' ⎕CPC ''

The New action will initialize the CPC data set in the current APL64 developer version instance. The New action does not modify the CPC xml-format configuration file in the current APL64 instance.

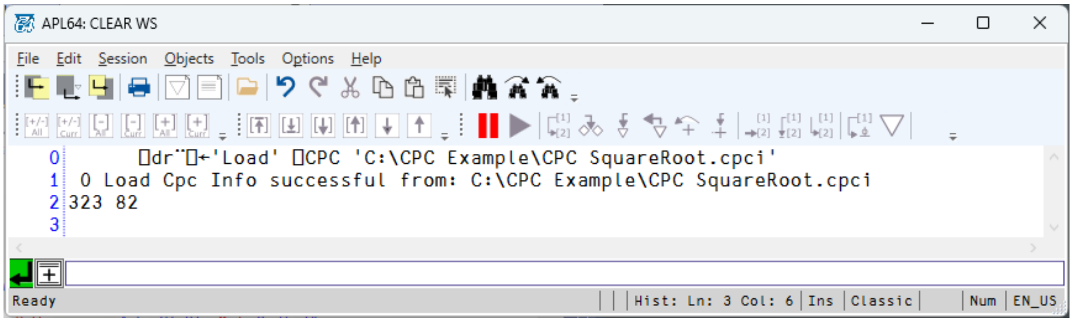


```
      ⎕dr¨⎕←'New' ⎕cpc ''
 0 New Cpc Info successful
323 82
```

⎕CPC Load Action

Syntax: ‘Load’ ⎕CPC pathToCpcConfigurationFile

The Load action will load a pre-prepared CPC xml-format configuration file into the CPC data set in the current APL64 developer version instance. The Load action does not modify the CPC xml-format configuration file in the current APL64 instance.



```
      ⎕dr¨⎕←'Load' ⎕CPC 'C:\CPC Example\CPC SquareRoot.cpci'
 0 Load Cpc Info successful from: C:\CPC Example\CPC SquareRoot.cpci
323 82
```

⎕CPC PropValue Action

Syntax: priorPropValue ← 'PropValue' ⎕CPC propertyName newPropertyValue

The PropValue action will get, and optionally set, the properties in the CPC xml-format configuration file in the current APL64 instance. These properties correspond to the **Options | Create Runtime .Net Assembly | Create Cross-platform Component** dialog. The property names are not case sensitive. The PropValue action may be used to modify the CPC xml-format configuration file in the current APL64 instance.

Property Names		
Description	**Property Name**	**Data Type**
Cross Platform Component Target Folder	TgtFolder	Text
Cross Platform Component Name	CompName	Text
Cross Platform Component Name Same As Workspace Name	CNameSameAsWsName	Bool
Cross Platform Component Class Name	CompClassName	Text
Apl64 Xml-Format Configuration File	ConfigFile	Text

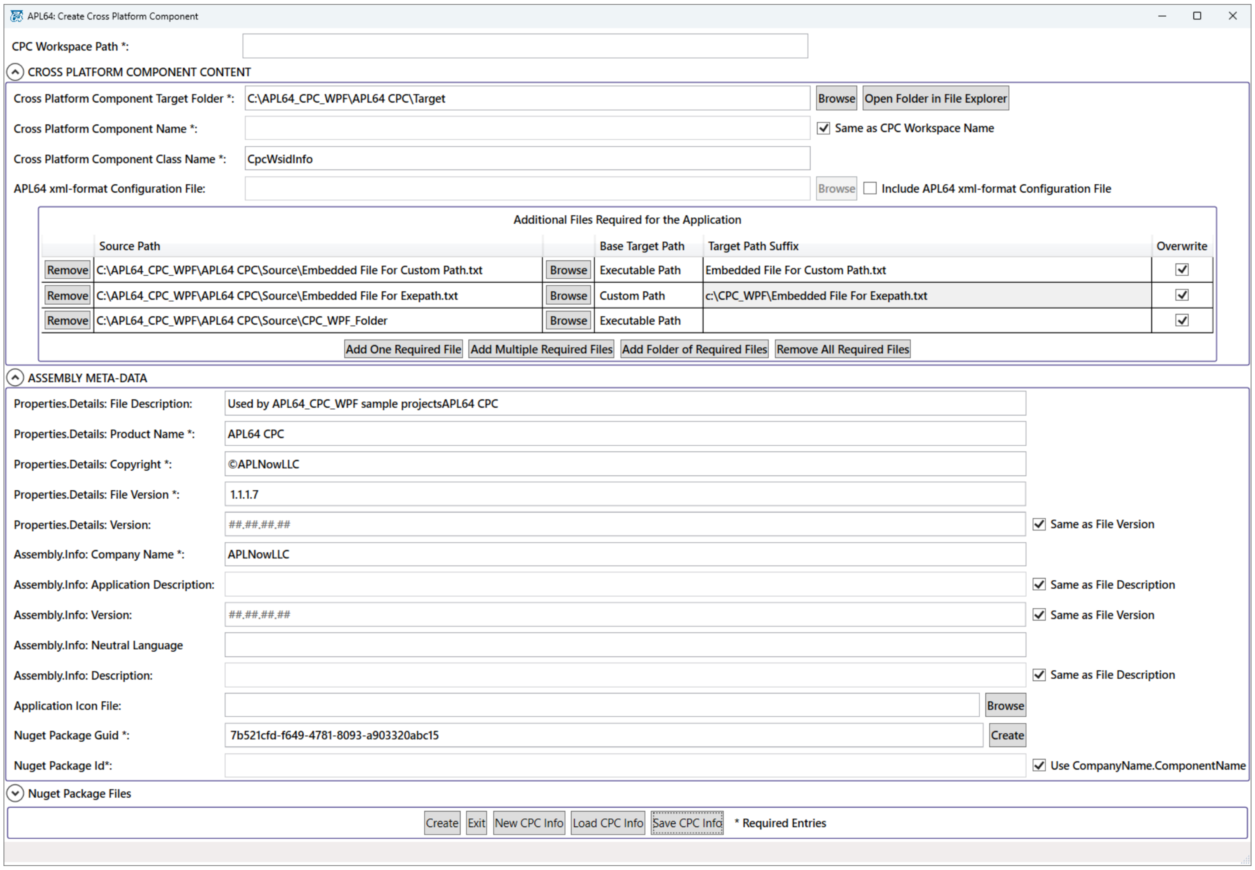
Include Apl64 Xml-Format Configuration File	InclConfigFile	Bool
Addnlappfiles	AddnlAppFiles	AAF-Matrix
Properties.Details: File Description	FileDesc	Text
Properties.Details: Product Name	ProdName	Text
Properties.Details: Copyright	Copyright	Text
Properties.Details: File Version	FileVersion	Text(#.#.#.#)
Properties.Details: Version	Version	Text(#.#.#.#)
Properties.Details: Version: Same As File Version	VersionSameAsFileVersion	Bool
Assembly.Info: Company Name	CoName	Text
Assembly.Info: Application Description	AppDesc	Text
Assembly.Info: Application Description: Same As File Description	AppDescSameAsFileDesc	bool
Assembly.Info: Assembly Version	AsmVer	Text(#.#.#.#)
Assembly.Info: Assembly Version Same As File Version	AsmVerSameAsFileVer	bool
Assembly.Info: Neutral Language"	NeutLang	text
Assembly.Info: Description	Desc	text
Assembly.Info: Description Same As File Description	DescSameAsFileDesc	bool
Application Icon File	AppIconFile	text
Nuget Package Guid	NugetPkgGuid	text
Nuget Package Id	NugetPkgId	text
Nuget Package Id: Use Companyname.Componentname	NugetPkgIdUseCoName,CName	bool
Cpc License File Path	LicFilePath	text
Cpc Readme File Path	ReadMeFilePath	text

Text: Character vector or string scalar

AAF-Matrix: One row for each additional application file	
Column Description	**Data Type**
Source Path	Text
Source Path is Folder	bool
Base Target Path	'ExecutablePath' or 'CustomPath'
Target Path	Text
Overwrite	bool

Examples:

CPC xml-format configuration file in the current APL64 instance:



APL64: Create Cross Platform Component

CPC Workspace Path *:		

CROSS PLATFORM COMPONENT CONTENT

Cross Platform Component Target Folder *:	C:\APL64_CPC_WPF\APL64 CPC\Target	Browse Open Folder in File Explorer
Cross Platform Component Name *:		☑ Same as CPC Workspace Name
Cross Platform Component Class Name *:	CpcWsidInfo	
APL64 xml-format Configuration File:		Browse ☐ Include APL64 xml-format Configuration File

Additional Files Required for the Application

	Source Path		Base Target Path	Target Path Suffix	Overwrite
Remove	C:\APL64_CPC_WPF\APL64 CPC\Source\Embedded File For Custom Path.txt	Browse	Executable Path	Embedded File For Custom Path.txt	☑
Remove	C:\APL64_CPC_WPF\APL64 CPC\Source\Embedded File For Exepath.txt	Browse	Custom Path	c:\CPC_WPF\Embedded File For Exepath.txt	☑
Remove	C:\APL64_CPC_WPF\APL64 CPC\Source\CPC_WPF_Folder	Browse	Executable Path		☑

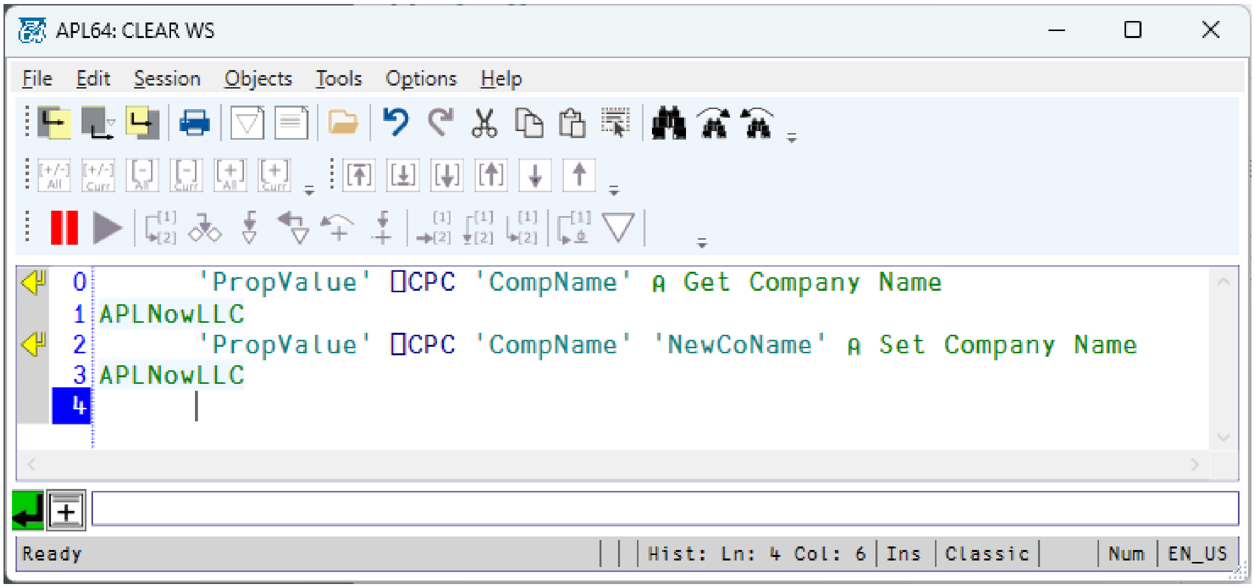
Add One Required File
Add Multiple Required Files
Add Folder of Required Files
Remove All Required Files

ASSEMBLY META-DATA

Properties.Details: File Description:	Used by APL64_CPC_WPF sample projectsAPL64 CPC	
Properties.Details: Product Name *:	APL64 CPC	
Properties.Details: Copyright *:	©APLNowLLC	
Properties.Details: File Version *:	1.1.1.7	
Properties.Details: Version:	##.##.##.##	☑ Same as File Version
Assembly.Info: Company Name *:	APLNowLLC	
Assembly.Info: Application Description:		☑ Same as File Description
Assembly.Info: Version:	##.##.##.##	☑ Same as File Version
Assembly.Info: Neutral Language		
Assembly.Info: Description:		☑ Same as File Description
Application Icon File:		Browse
Nuget Package Guid *:	7b521cfd-f649-4781-8093-a903320abc15	Create
Nuget Package Id*:		☑ Use CompanyName.ComponentName

Nuget Package Files

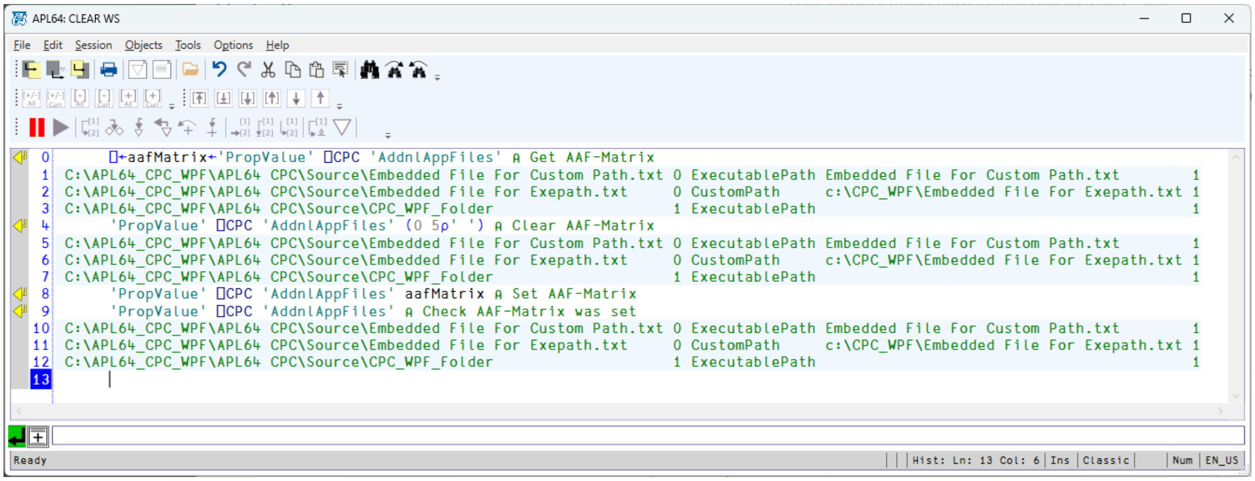
* Required Entries



APL64: CLEAR WS

```
0        'PropValue' ⎕CPC 'CompName' ⍝ Get Company Name
1 APLNowLLC
2        'PropValue' ⎕CPC 'CompName' 'NewCoName' ⍝ Set Company Name
3 APLNowLLC
4
```

Ready | Hist: Ln: 4 Col: 6 | Ins | Classic | Num | EN_US


APL64: CLEAR WS
File Edit Session Objects Tools Options Help
Ready
Hist: Ln: 13 Col: 6 Ins Classic Num EN_US

```
      ⎕←aafMatrix←'PropValue' ⎕CPC 'AddnlAppFiles' ⍝ Get AAF-Matrix
C:\APL64_CPC_WPF\APL64 CPC\Source\Embedded File For Custom Path.txt 0 ExecutablePath Embedded File For Custom Path.txt        1
C:\APL64_CPC_WPF\APL64 CPC\Source\Embedded File For Exepath.txt     0 CustomPath     c:\CPC_WPF\Embedded File For Exepath.txt 1
C:\APL64_CPC_WPF\APL64 CPC\Source\CPC_WPF_Folder                    1 ExecutablePath                                          1
      'PropValue' ⎕CPC 'AddnlAppFiles' (0 5⍴' ') ⍝ Clear AAF-Matrix
C:\APL64_CPC_WPF\APL64 CPC\Source\Embedded File For Custom Path.txt 0 ExecutablePath Embedded File For Custom Path.txt        1
C:\APL64_CPC_WPF\APL64 CPC\Source\Embedded File For Exepath.txt     0 CustomPath     c:\CPC_WPF\Embedded File For Exepath.txt 1
C:\APL64_CPC_WPF\APL64 CPC\Source\CPC_WPF_Folder                    1 ExecutablePath                                          1
      'PropValue' ⎕CPC 'AddnlAppFiles' aafMatrix ⍝ Set AAF-Matrix
      'PropValue' ⎕CPC 'AddnlAppFiles' ⍝ Check AAF-Matrix was set
C:\APL64_CPC_WPF\APL64 CPC\Source\Embedded File For Custom Path.txt 0 ExecutablePath Embedded File For Custom Path.txt        1
C:\APL64_CPC_WPF\APL64 CPC\Source\Embedded File For Exepath.txt     0 CustomPath     c:\CPC_WPF\Embedded File For Exepath.txt 1
C:\APL64_CPC_WPF\APL64 CPC\Source\CPC_WPF_Folder                    1 ExecutablePath                                          1
```

⎕CPC Create Action

Syntax: 'Create' ⎕CPC ''

The Create action will create a CPC executable using the CPC configuration data set in the current APL64 developer version instance at the location specified by the right argument. The output of the ⎕CPC Create method may indicate exceptions or warnings occurred during the process, in which case the resulting log file should be reviewed by the APL programmer.

The Create action does not modify the CPC xml-format configuration file in the current APL64 instance.

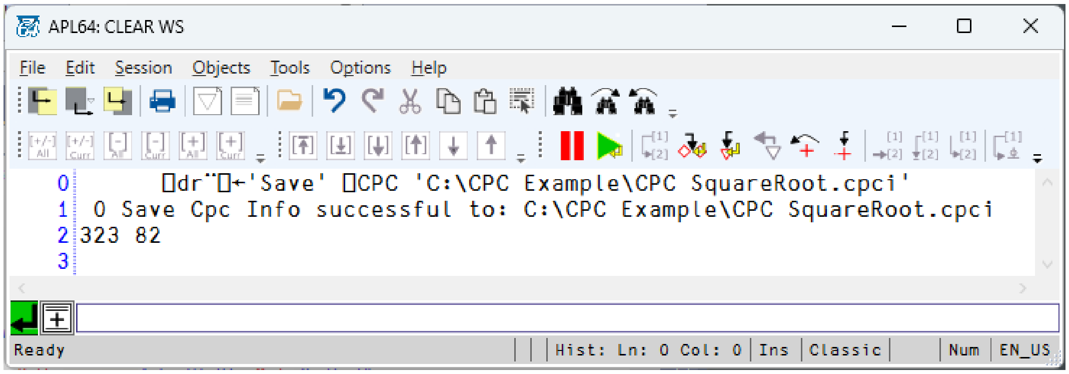
The pre-prepared CPC xml-format configuration file must be loaded prior to using the ⎕CPC Create action.

The pre-prepared workspace specified in the CPC configuration data set must be loaded prior to using the ⎕CPC Create action.

⎕CPC Save Action

Syntax: 'Save' ⎕CPC targetFileName

The Save action will save the CPC xml-format configuration data set in the current APL64 developer version instance to the file specified in the right argument.


APL64: CLEAR WS
File Edit Session Objects Tools Options Help
Ready
Hist: Ln: 0 Col: 0 Ins Classic Num EN_US

```
      ⎕dr¨⎕←'Save' ⎕CPC 'C:\CPC Example\CPC SquareRoot.cpci'
 0 Save Cpc Info successful to: C:\CPC Example\CPC SquareRoot.cpci
323 82
```

How the CPC Creation Utility Works

The APL64 programmer input to the CPC creation utility is used by the utility to define a .Net, C# language project in the target folder. This project contains:

- A .Net library assembly which is the basis of a Nuget package
- The APL64 programmer-specified APL64 workspace containing the APL64 programmer-defined, public functions
- The APL64 programmer-specified files required for the application system
- References to the .Net Nuget packages and assemblies required to run the APL64 interpreter

The CPC creation utility uses the .Net SDK installed on the APL64 programmer's workstation to publish the project into a Nuget package in the CPC target folder.

The resulting Nuget package can be included in a larger .Net project using any .Net language(s).

To Learn More

Contact support@apl2000.com for assistance with the APL64 CPC features of APL64.