

APL64: USING WSE

Summary.....	3
Synchronous Actions.....	4
Asynchronous Actions.....	4
 WSE Syntax	5
 WSE Actions.....	5
Actions of the  WSE Object.....	5
AsyncAnyTaskRunning.....	5
AsyncAvailableServerNames.....	5
AsyncCancelAll	6
AsyncFoFi	6
What is FOFI?	6
 WSE FoFi Operation	9
When to Use FoFi?.....	9
 WSE AsyncFoFi Action	9
 WSE AsyncFoFi Arguments	10
Number of Servers	10
Workspace to Load.....	10
Workspace Function to Run	10
Function Timeout	10
Processing Requests Argument Vector	10
OverallTimeout.....	10
FoFi Result	11
Processing Requests Outcome Vector	11
Processing Request Outcome Vector Element Structure	11
Outcome Data.....	11
Processing Requests Outcomes	11
AsyncFofi compared to AsyncCall, AsyncWait, CallNow and  Progress.....	11
 WSE 'AsyncFoFi' example	12
Setup of the example	12
Run the example	14
AsyncHandler	14

AsyncTasksRunning	16
AsyncWait	17
Clear	18
Count	18
Create	19
Help (?).....	20
Instances.....	21
New.....	22
☐ WSE Instance Actions	22
AsyncCall	23
AsyncCall Overview	23
Example: AsyncCall: Using One ☐ WSE Server.....	24
☐ WSE Client Instance Functions	24
☐ WSE Server Instance Function	26
Test the Example.....	27
Example: AsyncCall: Using Multiple ☐ WSE Servers	28
APL64 Client Instance Functions	28
APL64 Server Instance Function	32
Test the Example.....	34
Example: Stochastic Numerical Integration using Multiple ☐ WSE Instances	34
Math Details	34
☐ WSE Implementation using AsyncCall Action	35
APL64 ☐ WSE Client Instance Functions	35
APL64 ☐ WSE Server Instance Function	40
Test the Implementation	41
AsyncCancel	43
AsyncStatus	44
AsyncTaskIsRunning.....	45
Call	45
Niladic	46
Monadic.....	46
Dyadic	47
CallNow	48

Syntax: result←instanceName □wse 'CallNow' [fnName] [fnRightArg] [fnLeftArg]	48
Example: □WSE CallNow synchronous environment	48
Example: □WSE CallNow asynchronous parallel environment	50
Delete	57
Exec	57
GetSysVariable	58
GetVariable.....	58
LoadWs	58
SetSysVariable.....	58
SetVariable	59
SysCall	60
Niladic	60
Monadic.....	60
Dyadic	60
SysCommand	61
Variable	62
□WSE Action Restrictions	63
□WseSelf System Variable	63
□WseInfo System Variable	64
□WSE Comparision with Legacy Workspace Engine	65
□Progress.....	66
Using □Progress when an AsyncCall action is underway	67
□WseMc.....	67
Use □WseMc to send a message from an APL Server to the APL Client	67
Learn More	68

Summary

The □WSE system function can be used to create one or more independent instances of APL (engines). Each □WSE APL engine operates in a server role. The APL64 instance which uses □WSE to create APL engines operates in a client role. The client APL instance creates the □WSE APL servers, and requests processing services from the □WSE APL servers.

□WSE APL engines are in-process servers, contained in the memory allocation of the client APL instance. There is only one □WSE object for an APL64 client instance, and that □WSE contains all the □WSE server instances. A □WSE server APL instance of APL cannot use □WSE.

The □WSE system function may be used in an APL64 developer version instance, an APL64 Windows Runtime Executable (WRE) instance, or an APL64 Cross-platform Component (CPC) instance.

The □WSE system function is designed to be a cross-platform feature of APL64. When an APL64 Cross-platform Component (CPC) is used in a non-Windows environment, the Windows-only features of APL64 should not be used in a □WSE instance. The □WSE does not use or depend upon Windows-only ActiveX technology.

When the □WSE system function is used in an APL instance targeting the Windows environment all features of APL64, including □WI may be used by a □WSE server APL instance, except for □WSE. A □WSE server APL instance of APL cannot use □WSE.

The □WSE system function exposes actions. Since all □WSE server instances are contained within one □WSE object in the client APL instance, there are □WSE actions which apply to the □WSE object, and □WSE actions which apply to a specific □WSE server instance.

Most □WSE actions operate synchronously, i.e. while the action is processed, subsequent processing by the client instance of APL is suspended until the specified action processing is completed. □WSE supports several asynchronous actions, including AsyncCall and AsyncFoFi which may be used for parallel processing in APL64.

Synchronous Actions

Synchronous □WSE actions include Call, Load, SysCall, SysCommand, and Exec. While a □WSE synchronous action is running in a □WSE server instance only that □WSE synchronous action can be running.

Asynchronous Actions

Asynchronous □WSE actions include AsyncCall and AsyncWait. The □WSE asynchronous actions can be used for parallel processing of data using a pool of □WSE server instances.

When a client instance of APL uses the AsyncCall action on a specified □WSE server instance of APL, the results are not immediately provided to the □WSE client APL instance. Instead, the results of the AsyncCall action are provided via an event. This event can be subscribed by the client APL instance with an APL64 programmer-defined function.

While a □WSE AsyncCall action is running:

- □Progress in the □WSE instance can provide information to the Client APL instance
- Other □WSE AsyncCall actions can be running
- One □WSE synchronous action can be running
- If no □WSE synchronous action is running, execution in the client APL instance can continue

WSE Syntax

[result]←[leftArg] WSE action [rightArg]

The left argument is '#' for actions of the WSE object, which apply to all WSE instances. The left argument for actions of a WSE instance is the instance name or WseSelf, if it has been set to a valid WSE instance name.

WSE Actions

Action names are case insensitive.

Each WSE action has specific results and arguments.

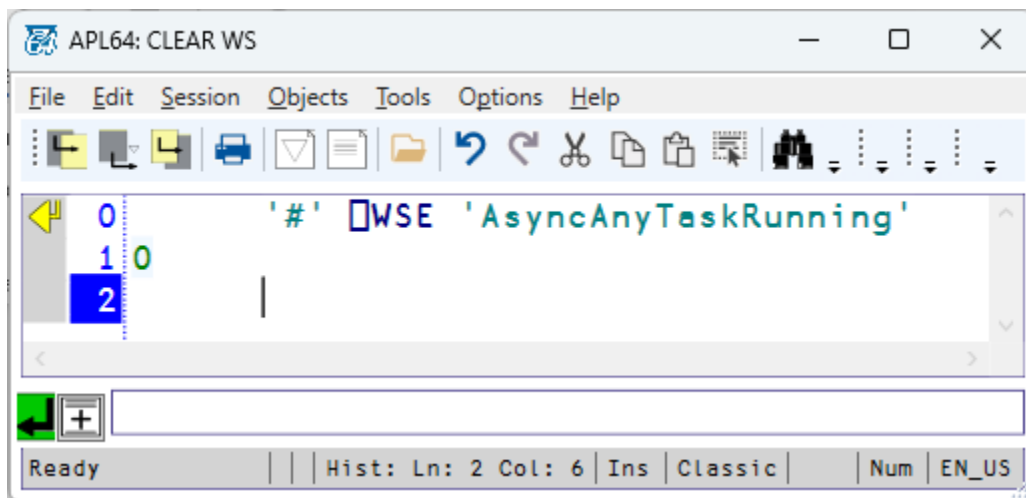
All Action examples use a new instance of the APL64 Developer version. Action examples may use multi-line APL64 executable statements.

Actions of the WSE Object

AsyncAnyTaskRunning

Indicates if any WSE asynchronous operation is in progress. This information is transient, as requesting this information does not modify the running state of any asynchronous operation in a WSE server APL instance.

```
bool ← '#' WSE 'AsyncAnyTaskRunning'
```

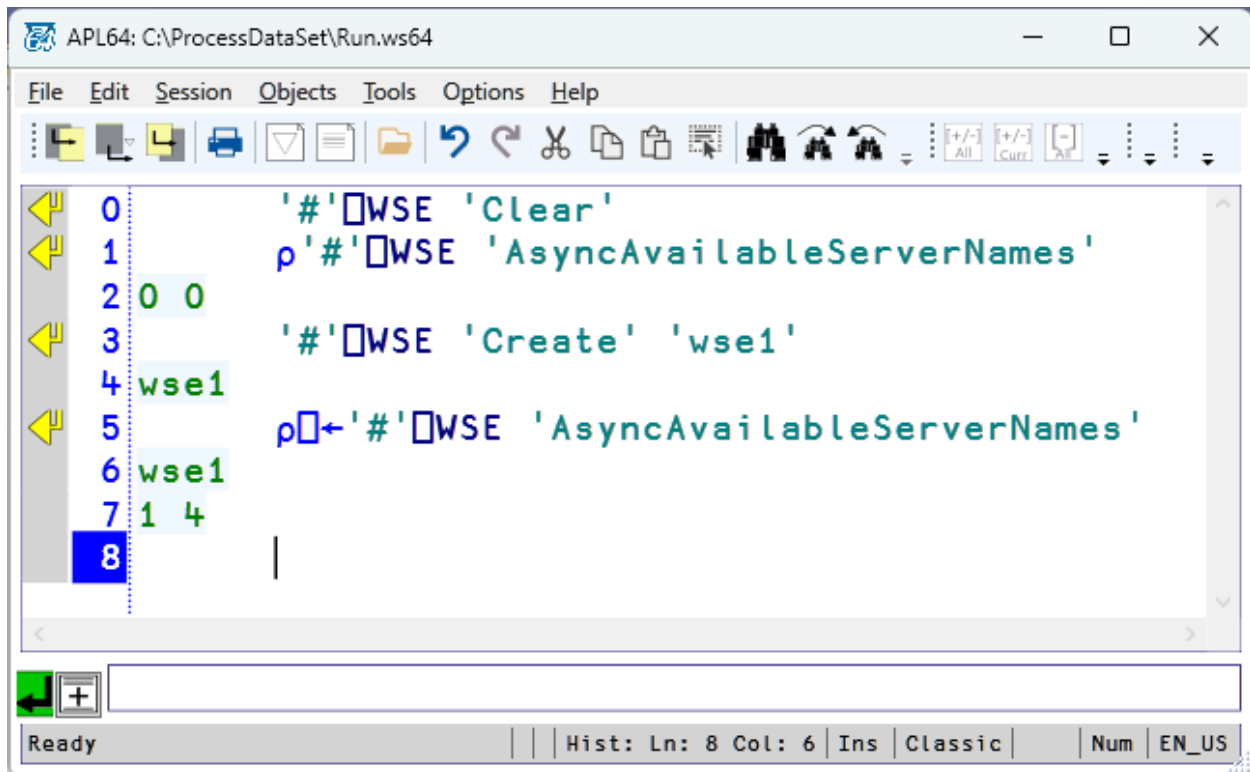


AsyncAvailableServerNames

Returns the WSE instance names of the WSE servers which have no asynchronous operations in progress. This information is transient, as requesting this information does not modify the running state of any asynchronous operation in a WSE server APL instance.

```
Rank2TextArray←'#' WSE 'AsyncAvailableServerNames'
```

```
'# WSE 'Clear'
ρ'# WSE 'AsyncAvailableServerNames'
'# WSE 'Create' 'wse1'
ρ←'# WSE 'AsyncAvailableServerNames'
```



AsyncCancelAll

Cancels all running ☐WSE asynchronous tasks running in a ☐WSE server instance. This action, if possible, is not instantaneous, but instead occurs when the APL64 interpreter determines it is appropriate to end the execution of the APL64 programmer-defined functions running in ☐WSE server APL instances.

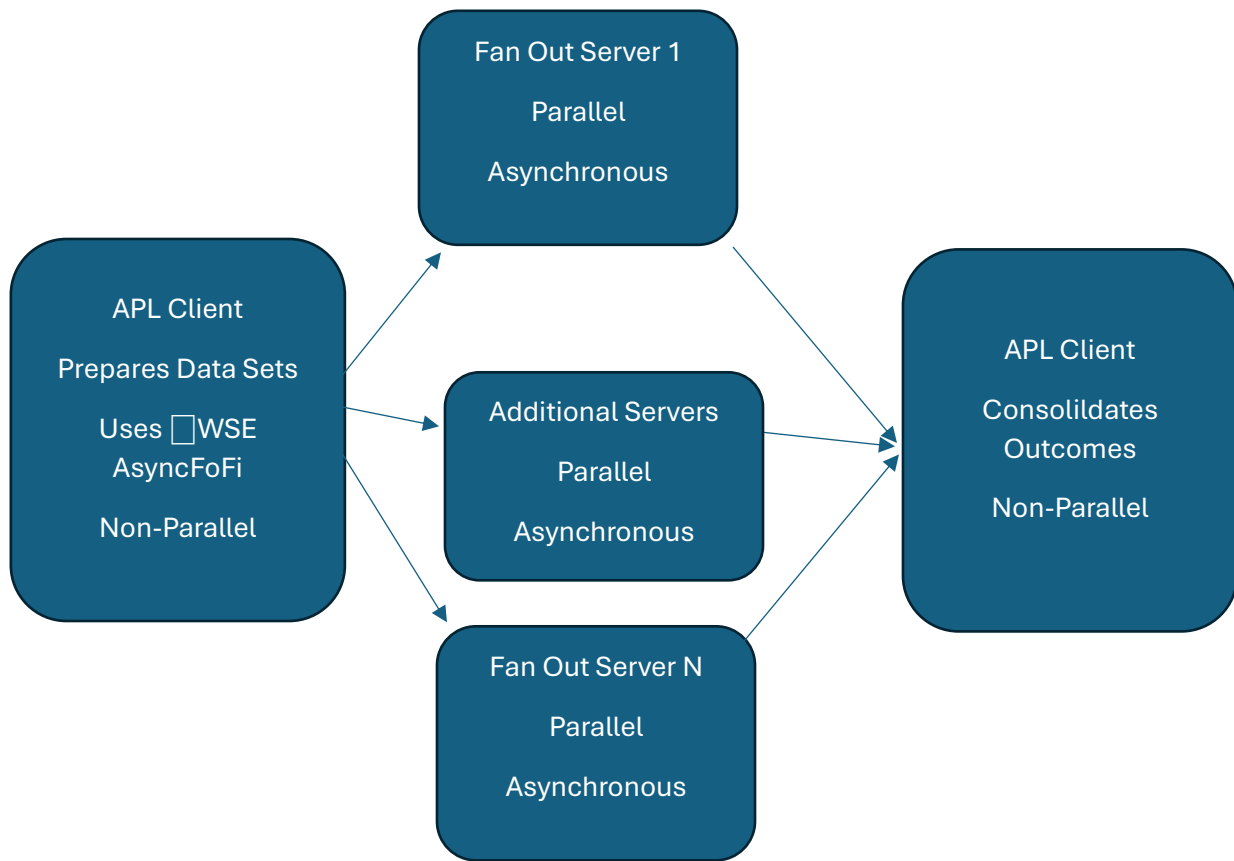
```
'# ☐WSE 'AsyncHandler'
```

AsyncFoFi

What is FOFI?

‘FOFI’ is an acronym for the ‘Fan Out Fan In’ program design structure. The ‘Fan Out’ program design structure uses multiple, independent, asynchronous servers each of which processes one individual member of a set of independent application tasks. The ‘Fan In’ program design structure consolidates the results of multiple, independent, asynchronous processes to a single result structure and makes that result available to the client which initiated the FoFi action.

The `WSE AsyncFoFi` action simplifies asynchronous, parallel running of a specified APL64 programmer-defined function for a set of function arguments. Each function argument represents a processing request made to a pool of `WSE` servers.



WSE FoFi Operation

The  WSE AsyncFoFi action is used from an APL64 client instance. The APL64 client specifies the number of  WSE servers, the APL workspace, the APL function in the workspace, the vector of function arguments and starts the FoFi action. The APL client then waits for the processing requests to be satisfied, and upon that time gets the results as a vector of processing outcomes. The AsyncFoFi action:

- Sets up a specified number of  WSE server instances to create a server pool.
- Loads the specified workspace in each instance containing a specified function.
- Runs, in parallel and asynchronously, the specified function in the workspaces loaded in the  WSE servers. Each  WSE server is provided with one element of a specified vector of function arguments which represent processing requests.  WSE servers in the pool are reused as necessary to complete the processing of all function arguments. This is the 'Fan Out' part of the FOFI design.
- Suspends processing in the APL client which created the pool of  WSE servers. The APL64 client waits until the processing of each function argument has reached an outcome, and all processing requests are satisfied.
- Consolidates the results of the processing requests into a single vector result, makes that result vector available to the APL client, and allows the continuation of processing by the APL client. This is the 'Fan In' part of the FOFI design.

One FOFI operation at a time is supported in an APL64 client instance. FOFI servers are in-process  WSE servers.

When to Use FoFi?

FoFi can improve the performance of an application system if:

- The application system needs to process records of the same type and structure using a specified APL64 programmer-defined function.
- The processing of each of those records is independent of the processing of all of the other records.
- There are many such records.
- The processing of a record takes considerable time to complete.
- The workstation has multiple physical or virtual processors
- The overhead of using a pool of  WSE servers is insignificant compared to the alternative synchronous, sequential processing of the many data sets.

WSE AsyncFoFi Action

The  WSE AsyncFoFi action is used to configure, start and complete a FOFI operation.

Syntax: foFiResult← wse'AsyncFoFi' arguments

□ WSE AsyncFoFi Arguments

□ WSE Async FoFi Arguments			
Arg#	Column Description	Data Type	Required
1	nServers	Int32	Y
2	wsPath	Server path	Y
3	fnName	Text	Y
4	fnTimeout	Int32 milliseconds	Y
5	fnArgs	APL variable	Y
6	overallTimeout	Int32 milliseconds	Y

Number of Servers

The nServers value should generally be no more than two less than the number of logical processors available on the workstation. Refer to 7▷ □ sysinit for the number of logical processors available on the workstation. This value will be used to create a pool of □ WSE server instances.

For example, if nServers is 5, when the □ WSE AsyncFOFI action is used, five □ WSE server instances will be created. The names of these □ WSE servers will be 'FOFI_0', ..., 'FOFI_4'.

After the completion of an AsyncFOFI action, these □ WSE instances are deleted.

Workspace to Load

The wsPath should point to an APL64 workspace which is accessible to the client APL instance before the FOFI operation begins. This workspace must contain the function to run.

Workspace Function to Run

The fnName is the name of an APL monadic function in the workspace to load. The function to run may have an optional result. The right argument to the function to run will be set from the applicable element of the processing requests argument vector.

Function Timeout

The fnTimeout value limits the running time of the function to run. If the timeout expires the processing request will be cancelled.

Processing Requests Argument Vector

The fnArgs is an APL64 rank-1 variable of arguments for the function to run. Each element of the Processing Requests Argument Vector will be used initiate an APL client request to run an independent, asynchronous processing request on one of the APL servers in the server pool.

OverallTimeout

The overallTimeout is an Int32 milliseconds value which limits the entire time allocated to the FOFI operation. If processing requests have not resulted in an outcome when the overallTimeout expires, they will be cancelled. When the overallTimeout expires, the FOFI operation complete event fires in the scope of the client APL instance so that normal client APL operation is restored, including access to the FOFI Processing Requests Outcome Vector.

FoFi Result

The foFiResult is vector of FoFi processing outcomes. A successful FoFi operation means that a valid 'Processing Request Outcome Vector' has been returned to the APL client. Each processing request generates a processing request outcome. The type of a processing request outcome can be completed (with or without a result), cancelled, exception and information from server to client.

Processing Requests Outcome Vector

All server processing outcomes are made available to the client when all server processing requests have determined their outcomes. The Processing Requests Outcome Vector will contain an outcome element for each of the processing requests in the 'Processing Requests Argument Vector'.

Processing Request Outcome Vector Element Structure

Processing Request Outcome Vector Element Structure		
Element#	Element Description	Data Type
1	WSE server id	text
2	Request Id	Consecutive integer
3	Outcome Type	Text
4	Outcome Data	APL variable

Outcome Data

Outcome Data		
Outcome Type	Outcome Data Description	Outcome Data Type
Completed without Result	No result available	"
Complete with Result	Result of function in server workspace	APL variable
Cancelled	Reason for cancellation	APL character vector
Exception Occurred	Exception message text	APL character variable
Info. From Server to Client	Information from server	APL variable

The outcome data of a processing request is

Processing Requests Outcomes

Processing Request Outcomes		
Description	Outcome Type Text	Processing Request Ended
Completed without Result	CompleteWithoutResult	Yes
Complete with Result	CompleteWithResult	Yes
Cancelled	Cancel	Yes
Exception Occurred	Error	Yes
Info. From Server to Client	Progress	No

AsyncFofi compared to AsyncCall, AsyncWait, CallNow and Progress

Combination of the WSE AsyncCall, AsyncWait, CallNow actions and Progress system function supports:

- Asynchronous processing of multiple processing requests using `WSE` server instances
- Communication between the server and the client is available while asynchronous processing is under way. `Progress` system function can provide information (typically feedback) from a `WSE` server to the APL client. The `WSE CallNow` action can provide information from the APL client to a `WSE` server.
- Support for different workspaces and functions for the `WSE` server pool.
- Requires an APL64 programmer-created 'control' function to:
 - Set up the `WSE` server pool
 - Send the processing requests to the pool using the `WSE AsyncCall` action
 - Request a wait while all servers in the pool are busy using the `WSE AsyncWait` action
 - Create additional processing requests as `WSE` server pool members become available
 - Request a wait until all processing requests are satisfied
- Requires an APL64 programmer-created 'handler' function to respond to the processing request satisfied event
- `WSE` server(s) processing can be stopped or suspended using the `WSE CallNow` action with coordination of the `WSE` server instance

The `WSE AsyncFoFi` action

- Is easier to use than the combined use of the `WSE AsyncCall`, `AsyncWait`, `CallNow` actions and `Progress`.
- Under the covers of the `WSE AsyncFoFi` action the 'control' and 'handler' functions are created and used by the APL64 interpreter.
- Only one workspace and one function apply to all `WSE` server instances.
- Feedback to the APL client is not available for the `WSE FoFi` action. The `Progress` system function and the `WSE CallNow` action are not considered during a `WSE AsyncFoFi` operation.
- `WSE` server processing, once started cannot be stopped or suspended by the APL client.
- All processing requests end when the `WSE` execution of the specified function ends, or the `overallTimeout` expire.

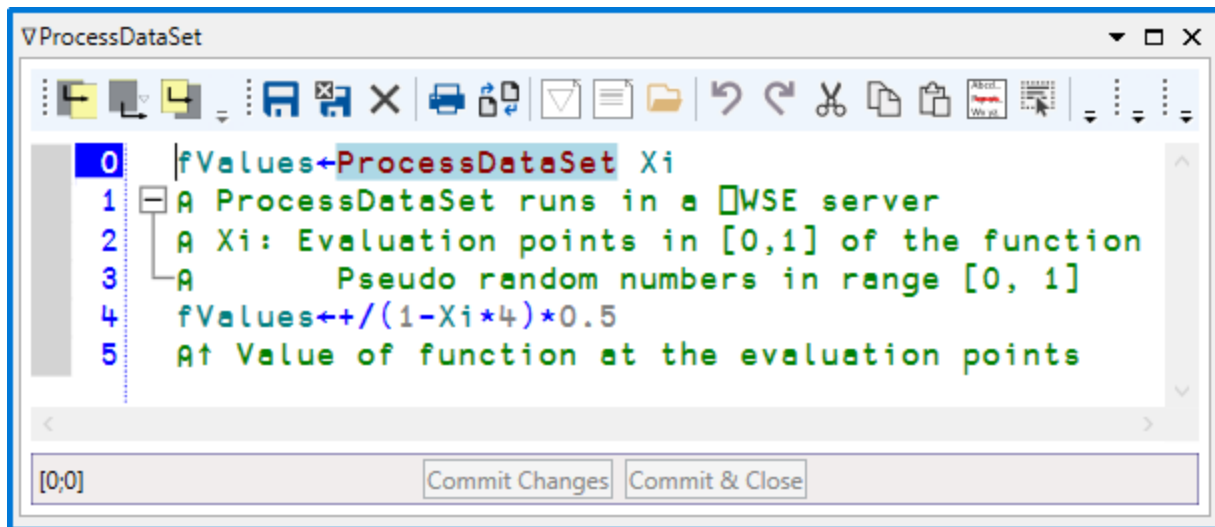
`WSE 'AsyncFoFi' example`

This example uses the same mathematics the [Stochastic Numerical Integration using Multiple WSE Instances](#) for the `WSE 'AsyncCall'` action.

Setup of the example

Create the APL64 workspace 'c:\AsyncFoFi.ws64'. In this workspace create the `ProcessDataSet` function, and save the workspace.

```
fValues ← ProcessDataSet Xi
⌈ ProcessDataSet runs in a WSE server
⌈ Xi: Evaluation points in [0,1] of the function
⌈ Pseudo random numbers in range [0, 1]
fValues ← +/(1 - Xi*4)*0.5
⌈ ↑ Value of function at the evaluation points
```

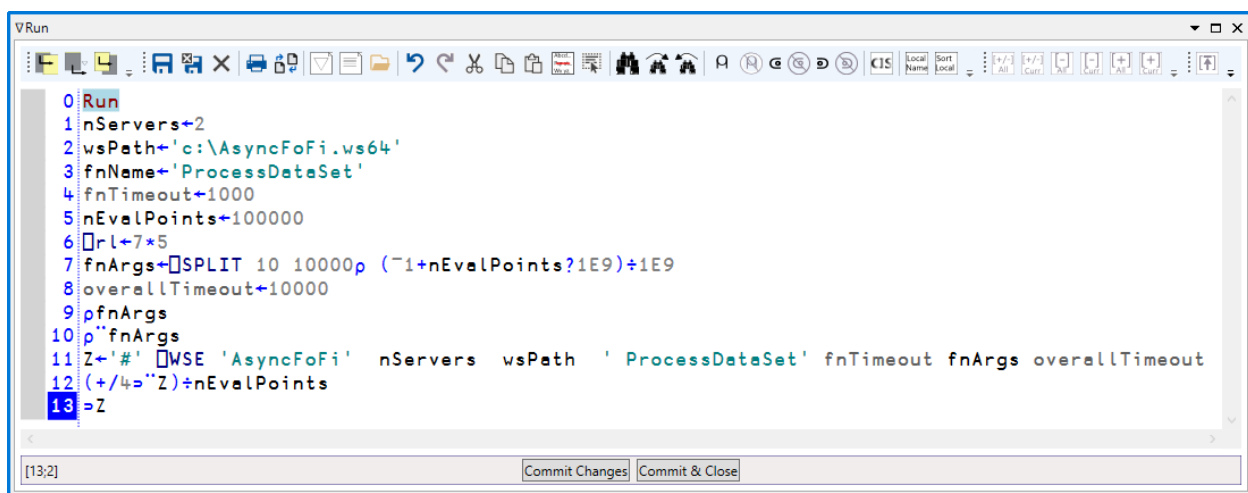


Create the Run function:

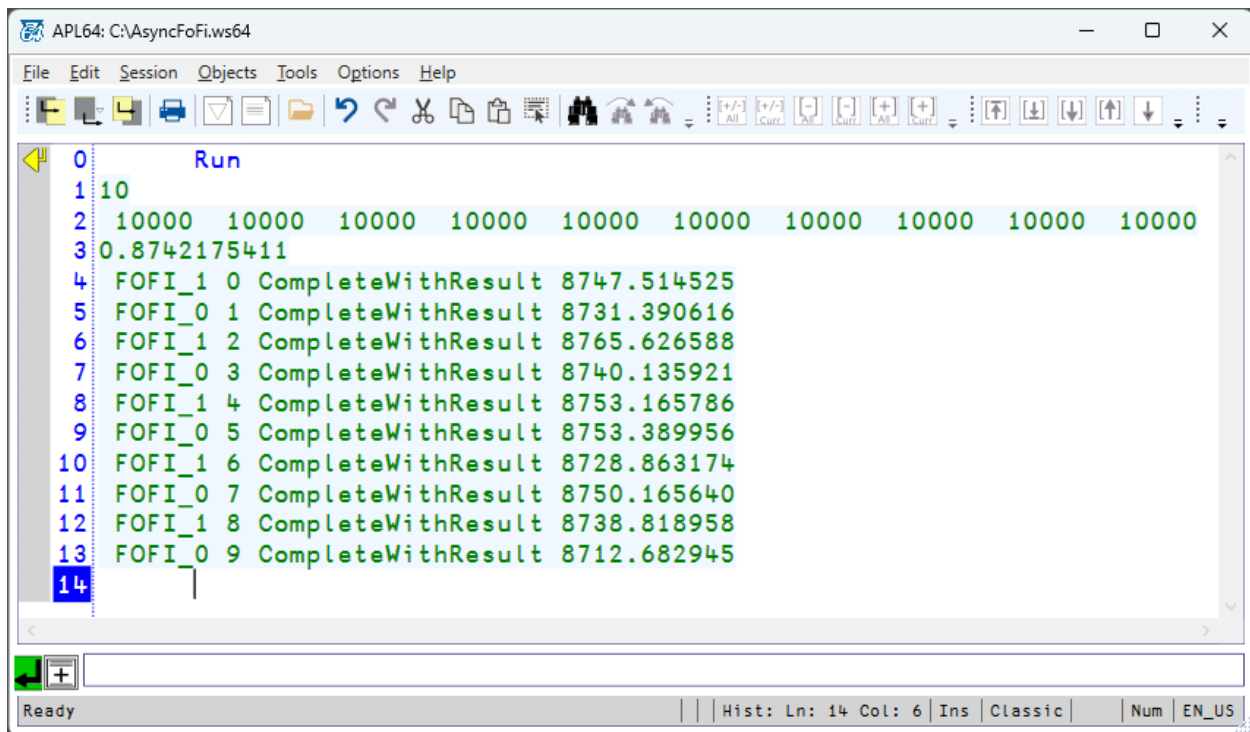
```

Run
nServers←2
wsPath←'c:\AsyncFoFi.ws64'
fnName←'ProcessDataSet'
fnTimeout←1000
nEvalPoints←100000
rl←7*5
fnArgs←SPLIT 10 10000p (~1+nEvalPoints?1E9)÷1E9
overallTimeout←10000
pfnArgs
p`fnArgs
Z←'#' WSE 'AsyncFoFi' nServers wsPath ' ProcessDataSet' fnTimeout fnArgs overallTimeout
(+/4>"Z)÷nEvalPoints
>Z

```



Run the example



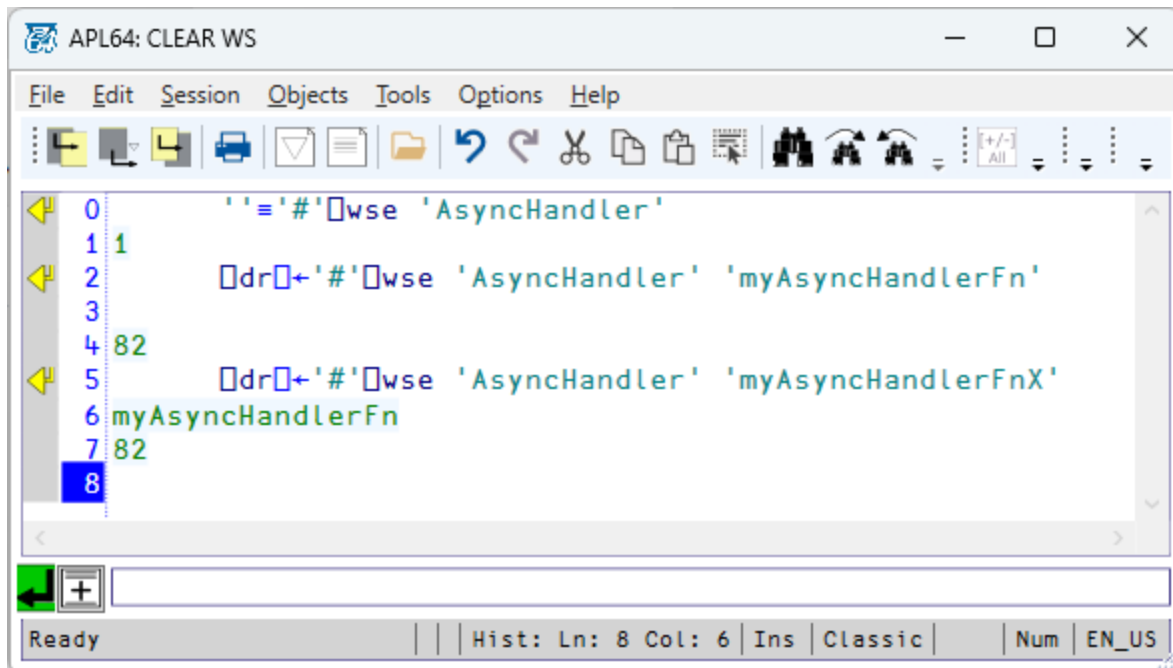
```
0 Run
1 10
2 10000 10000 10000 10000 10000 10000 10000 10000 10000 10000
3 0.8742175411
4 FOFI_1 0 CompleteWithResult 8747.514525
5 FOFI_0 1 CompleteWithResult 8731.390616
6 FOFI_1 2 CompleteWithResult 8765.626588
7 FOFI_0 3 CompleteWithResult 8740.135921
8 FOFI_1 4 CompleteWithResult 8753.165786
9 FOFI_0 5 CompleteWithResult 8753.389956
10 FOFI_1 6 CompleteWithResult 8728.863174
11 FOFI_0 7 CompleteWithResult 8750.165640
12 FOFI_1 8 CompleteWithResult 8738.818958
13 FOFI_0 9 CompleteWithResult 8712.682945
14
```

AsyncHandler

Specify the name of the APL64 programmer-developed function which will be used as the handler for events fired during ☐WSE asynchronous operations

priorHandlerFnName < '#' ☐WSE 'AsyncHandler' [NewHandlerFnName]

NewFnName is the optional name of the handler function, which will replace the previous handler function name, if any.



The handler function is necessary for the APL64 client instance to be notified of ☐WSE asynchronous operation events:

- ☐WSE asynchronous operation completed without result
- ☐WSE asynchronous operation completed with result
- ☐WSE asynchronous operation exception occurred
- ☐WSE asynchronous operation ☐PROGRESS information
- ☐WSE Asynchronous operation cancelled

The handler function:

- Cannot have a result.
- Is executed when the ☐WSE asynchronous operation fires an applicable event
- Cannot have a left argument
- Must have a right argument
- Must exist in the APL64 instance which created the ☐WSE instance at the time an ☐WSE asynchronous operation event is initiated.

The ☐WSE AsyncHandler property value cannot be modified while any ☐WSE asynchronous operation is in progress.

Because this event handler function is running in the 'client' APL64 instance, care must be taken to keep its actions independent of any other operations occurring in the APL64 client instance.

When an ☐WSE asynchronous operation event fires, the programmer-specified handler function will be called by the APL64 instance which created the ☐WSE instance. The value of the argument to the handler function is an APL64 variable of rank 1:

Argument provided to the ☐WSE AsyncHandler function

Element #1	Element #2	Element #3	Element #4
☐ WSE instance name	APL64 Context Variable	'CompletionWithResult'	APL64 Result Variable
☐ WSE instance name	APL64 Context Variable	'CompletionWithoutResult'	0
☐ WSE instance name	APL64 Context Variable	'Error'	Exception message
☐ WSE instance name	APL64 Context Variable	'Progress'	APL64 Progress Variable
☐ WSE instance name	APL64 Context Variable	'Cancel'	"

☐WSE instance name is the APL64 programmer-defined name of the ☐WSE instance which received the request to perform an ☐WSE asynchronous operation.

APL64 Context Variable is an APL64 programmer-defined variable, provided when a ☐WSE asynchronous operation is initiated, which identifies a specific request to perform an ☐WSE asynchronous operation.

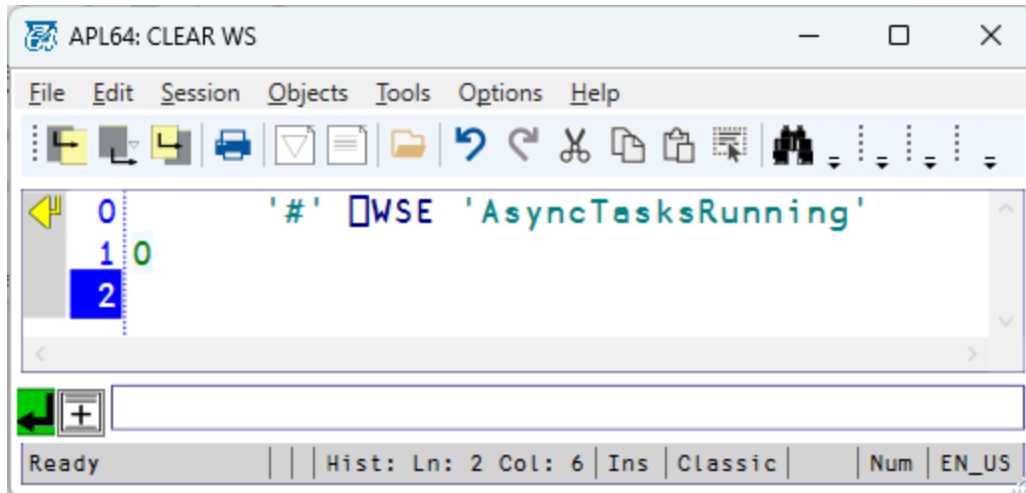
APL64 Result Variable is the APL64 programmer-defined variable which is returned by the APL64 programmer-defined function run in the ☐WSE server APL instance.

APL64 Progress Variable is an APL64 programmer-defined variable which is used to provide the progress of the ☐WSE asynchronous operation in a ☐WSE server APL instance to the ☐WSE client APL instance. The APL64 asynchronous operation running in the server APL instance can use the APL64 ☐Progress system function to provide this information. Using the APL64 ☐Progress system function causes the ☐WSE server APL instance to fire the Progress event.

AsyncTasksRunning

Provides the number of ☐WSE asynchronous operations is in progress. This information is transient, as requesting this information does not modify the running state of an asynchronous operation in the ☐WSE server APL instance.

Int32 ←'#' ☐WSE 'AsyncAnyTaskRunning'



AsyncWait

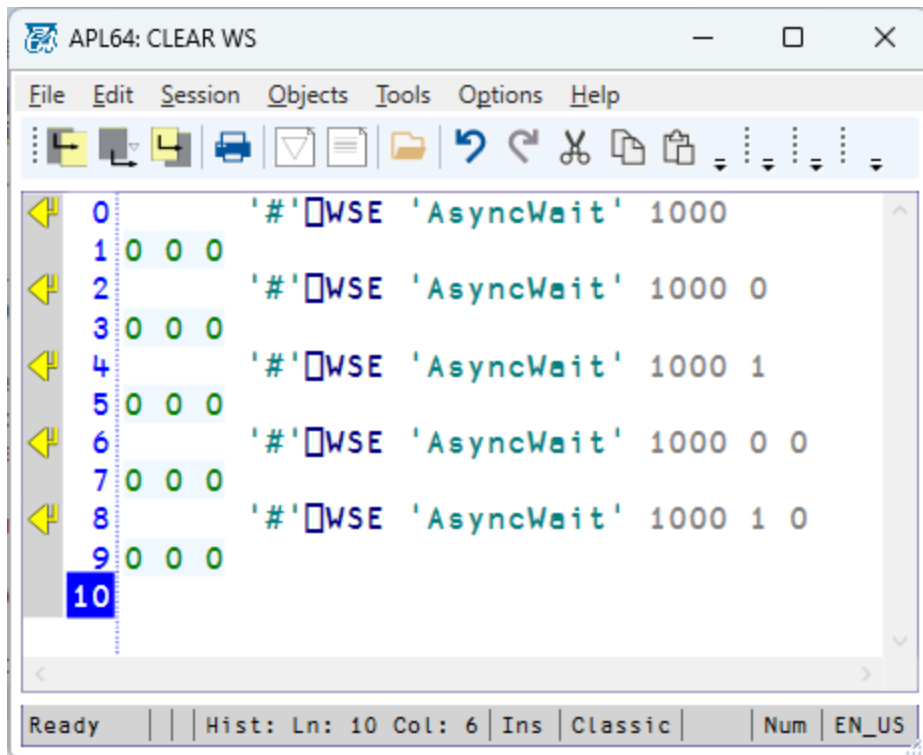
The AsyncWait action is used when the number of asynchronous AsyncCall requests exceeds the number of WSE server instances. When all WSE servers are working on AsyncCall actions, use the AsyncWait action to pause the operation of the client APL instance until a WSE server instance becomes available to process the next AsyncCall action..

(nRunning nAvail nPending) ← '# Wse 'AsyncWait' waitLimit [all] [final]

AsyncWait Argument	Argument Description
waitLimit (milliseconds)	Wait will end if the waitLimit expires
waitFor	Wait for completion of 0/Any one 1/Wait for all
Final (bool)	Wait for any one AsyncHandler to complete

```

'# WSE 'AsyncWait' 1000
'# WSE 'AsyncWait' 1000 0
'# WSE 'AsyncWait' 1000 1
'# WSE 'AsyncWait' 1000 0 0
'# WSE 'AsyncWait' 1000 1 0
  
```



Clear

Delete all ☐WSE instances. An exception is not thrown if there are no existing ☐WSE instances. The ☐wse Clear action returns no result. Any ☐WSE asynchronous operations running in an ☐WSE server APL instance will be cancelled when the Clear action is used.

```
'#' ☐WSE 'Clear'
```

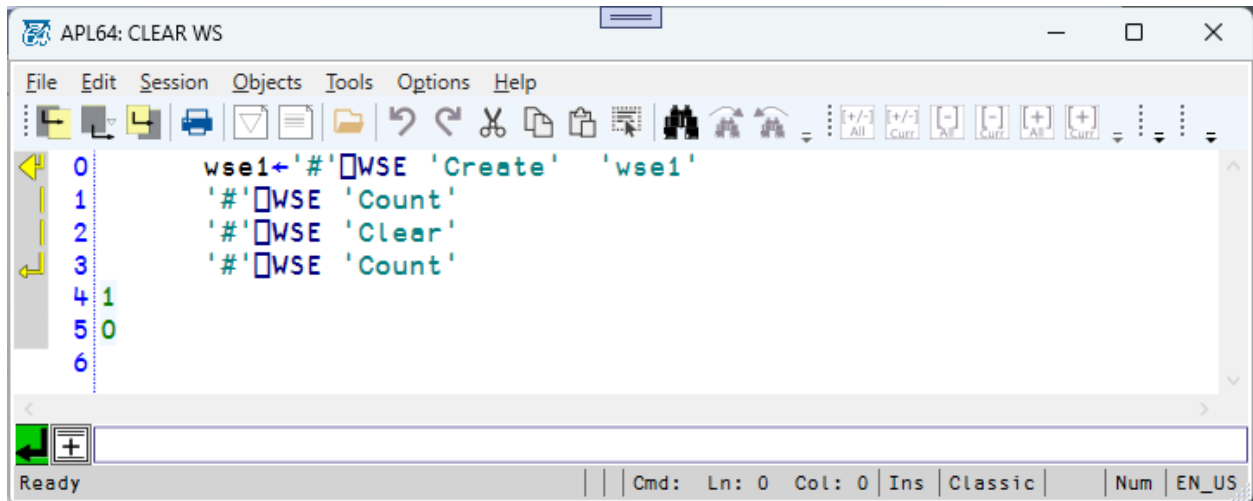
```
wse1<-'#' ☐WSE 'Create' 'wse1'
'#' ☐WSE 'Count'
'#' ☐WSE 'Clear'
'#' ☐WSE 'Count'
'#' ☐WSE 'Clear'
```

Count

Obtain the number of existing ☐WSE instances as an Int32 scalar.

```
Int32 <-'#' ☐WSE 'Count'
```

```
wse1<-'#' ☐WSE 'Create' 'wse1'
'#' ☐WSE 'Count'
'#' ☐WSE 'Clear'
'#' ☐WSE 'Count'
```



Create

Create a ☐wse instance. The result of the ☐wse Create action is a character vector containing the instance name.

☐WSE instances persist during the current APL64 instance, unless all are cleared or individually deleted during that APL64 instance. An exception is not thrown if the specified ☐WSE instance already exists. Instead, the previously existing ☐wse instance is deleted, and all asynchronous operations are cancelled. The instance name result is an APL64 character vector. The instance name argument may be a character scalar, character vector or a string scalar.

instanceName(charVec) ← '#' ☐WSE 'Create' instanceName(textVal)

☐ dr ☐ ←'# ☐ wse 'Create' 'wse1'
'# ☐ wse 'Instances'
'# ☐ wse 'New' 'wse1'
☐ dr ☐ ←'# ☐ wse 'Create' 'wse1'

The screenshot shows the APL64: CLEAR WS window with the following content:

```
0      dr←'#'wse 'Create' 'wse1'
1 wse1
2 82
3      '#'wse 'Instances'
4 wse1
5
6      '#'wse 'New' 'wse1'
7 DOMAIN ERROR: WSE: Instance wse1 already exists
8 [imm] '#'wse 'New' 'wse1'
9      ^
10     dr←'#'wse 'Create' 'wse1'
11 wse1
12 82
13
```

The status bar at the bottom indicates: Ready | Hist: Ln: 13 Col: 6 | Ins | Classic | Num | EN_US

Help (?)

Obtain a new line-delimited character vector containing summary documentation of the `wse` system function.

```
'#'wse '?'
```

```

APL64: CLEAR WS
File Edit Session Objects Tools Options Help
0 1 '# wse '?'
1 wse Object Actions:
2 wse Summary Documentation
3 4 '# wse 'AsyncCancelAll'
4 priorFnName + '# wse 'AsyncHandler' [aplFnName]
5 bool + '# wse 'AsyncAnyTaskRunning'
6 Rank2TextArray + '# wse 'AsyncAvailableServerNames'
7 Int32 + '# wse 'AsyncTasksRunning'
8 fnResultsVector[] + '# wse 'AsyncFoFi' nServers wsPath fnName fnTimeOut fnArgsVector[] overallTimeout
9 (nRunning nAvail nPending) + '# wse 'AsyncWait' waitLimit [waitFor(0/AnyOne 1/All)] [final(bool)]
10 11 '# wse 'Clear'
11 Int32 + '# wse 'Count'
12 instanceName(charVec) + '# wse 'Create' instanceName(textVal)
13 vector of charVec + '# wse 'Instances'
14 charVec + '# wse 'Help'
15 charVec + '# wse '?'
16 instanceName(charVec) + '# wse 'New' instanceName(textVal)
17 wse Instance Actions:
18 instanceName wse 'AsyncCall' context fnName [RightArg] [LeftArg]
19 bool + instanceName wse 'AsyncCancel'
20 char[] + instanceName wse 'AsyncStatus'
21 bool + instanceName wse 'AsyncTaskIsRunning'
22 res + instanceName wse 'Call' 'NiladicFn'
23 res + instanceName wse 'Call' 'MonadicFn' RightArg
24 res + instanceName wse 'Call' 'DyadicFn' RightArg LeftArg
25 char[] + instanceName wse 'Delete'
26 res + instanceName wse 'Exec' 'Expression'
27 res + instanceName wse 'GetSysVariable' 'variableName'
28 res + instanceName wse 'GetVariable' 'variableName'
29 instanceName wse 'LoadWs' wsPath
30 (hasError charVec) + instanceName wse 'MsgSvr' aplVar
31 priorFnName + instanceName wse 'MsgSvrEh' fnName
32 res + instanceName wse 'SysCall' 'NiladicSystemFn'
33 res + instanceName wse 'SysCall' 'MonadicSystemFn' RightArg
34 res + instanceName wse 'SysCall' 'DyadicSystemFn' RightArg LeftArg
35 instanceName wse 'SetSysVariable' 'variableName' variableValue
36 instanceName wse 'SetVariable' 'variableName' variableValue
37 instanceName wse 'SysCommand' 'sysCmdName Arg'
38 res + instanceName wse 'Variable' 'variableName'
39 instanceName wse 'Variable' 'variableName' variableValue
40
41 Notes:
42 (1) textVal is an APL64 string scalar, character scalar or character vector
43 (2) Expression is an APL64 executable statement (textVal)
44 (3) fnName (textval)
45 (4) context
46 (5) LeftArg
47 (6) RightArg
48 (7) variableName (textval)
49 (7) wsPath: Full path to workspace (textval)
50

```

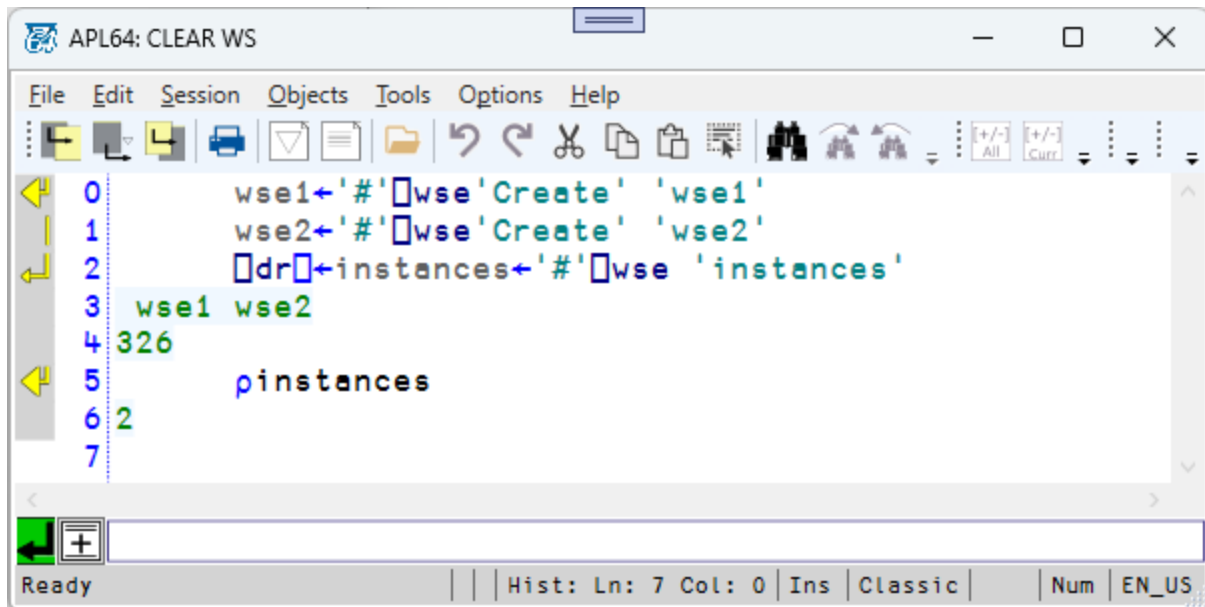
Instances

Obtain a vector of character vectors of the names of existing `wse` instances.

```

wse1←'# wse'Create' 'wse1'
wse2←'# wse'Create' 'wse2'
dr←instances←'# wse 'instances'
instances

```



New

Create a ☐wse instance, but do not delete a prior ☐wse instance of the same name. The result of the ☐wse Create action is a character vector containing the instance name.

☐WSE instances persist during the current APL64 instance, unless all are cleared or individually deleted during that APL64 instance. An exception is thrown, if the specified ☐WSE instance already exists. The instance name result is an APL64 character vector. The instance name argument may be a character scalar, character vector or a string scalar.

instanceName(charVec) ←'#' ☐WSE 'New' instanceName(textVal)

For an example, see the 'Create' action example.

☐WSE Instance Actions

The left argument of ☐WSE for all ☐WSE instances actions is a character scalar, character vector or string scalar which is a reference to a ☐WSE instance.

When a ☐WSE server APL instance is running ☐WSE asynchronous actions, it is up to the APL64 programmer to avoid using ☐WSE instances actions which would disrupt those asynchronous actions.

For example, if one or more ☐WSE AsyncCall actions are running in a ☐WSE server instance, the APL64 programmer-defined functions running in a ☐WSE server APL instance as part of the AsyncCall actions, should not load a workspace or delete APL variables upon which other AsyncCall actions are dependent.

AsyncCall

AsyncCall Overview

Asynchronously run an APL64 programmer-developed function in the APL64 instance of a specified □WSE instance.

Consider each □WSE instance as an APL64 server, and the APL64 instance which created the □WSE instances as the APL64 client. Using the AsyncCall action, several independent calculation processes may be initiated by the □WSE APL64 client. Each independent calculation process uses the resources of its respective □WSE APL64 server instance.

When the APL64 client makes a request to an APL64 server, using the AsyncCall action, the execution of the APL64 client is not suspended. Instead, the APL64 client execution continues, permitting the APL64 client to make additional requests. As each APL64 server completes its calculation process, the results are provided to the APL64 client via an event which runs the APL client [AsyncHandler](#) function. The AsyncHandler function is an APL64 programmer-defined function in the APL64 client instance which receives the server results. The AsyncHandler function uses the resources of the APL64 client instance to consolidate multiple APL64 server results.

Syntax: instanceName □WSE 'AsyncCall' context fnName [RightArg] [LeftArg]

context is the arbitrary APL64 programmer-defined variable provided when the □WSE asynchronous operation was requested. The context value should be unique among all the AsyncCall actions made by the APL64 client. Consider using □guid for the context value. The context value may be used by □WSE APL64 client as a reference value identifying a specific AsyncCall action, because the original context value is provided to the □WSE APL64 client instance when an APL64 server completes an AsyncCall operation. The context value is not modified once it is provided to the □WSE AsyncCall action.

The name of the specified, fnName, may be a character scalar, character vector or string scalar, which will be run asynchronously in the specified □WSE APL64 server instance. This function must exist in the specified □WSE APL64 server instance when the □WSE 'AsyncCall' action is initiated. The result, left argument, and right argument of this function are optional. Explicit or implicit output of this function, if any, is not visible because it is run in an □WSE APL64 server instance.

When any □WSE 'AsyncCall' action is running, the □WSE actions available to the □WSE client APL instance are limited:

- All □WSE object actions, except modifying the AsyncHandler are permitted
- All □WSE instance actions are permitted on □WSE instances which are not currently running an asynchronous operation
- □WSE actions permitted on a □WSE instance running an asynchronous operation are AsyncCancel, AsyncStatus, AsyncTaskIsRunning

The □WSE AsyncCall action returns immediately without a result. The asynchronously called (fnName) function runs in the □WSE APL64 server instance. Only one AsyncCall action can run at a time in a specific □WSE APL64 server instance. AsyncCall actions can be simultaneously run in separate □WSE apl64 SERVER instances.

The processing of a □WSE client request by a □WSE server always results in an ‘outcome’ which triggers an event in the □WSE client scope. Information about this ‘outcome’ is available to the □WSE client by subscribing to this event using the □WSE [AsyncHandler](#) action. The APL64 client instance which initiated the □WSE asynchronous operation does not control when the applicable □WSE server outcome events fire.

AsyncCall Outcome Events	Event Type (text) Provided to □WSE Client
Cancellation of processing	Cancel
Completion of processing without result	CompleteWithoutResult
Completion of processing with result	CompleteWithResult
Exception occurred during processing	Error
Info. from □WSE server to □WSE client	Progress

Example: AsyncCall: Using One □WSE Server

Sometimes it is desirable to start a long-running process, and immediately continue with other processing while the long-running calculation is running. The □WSE AsyncCall action may be used for this purpose where the APL64 client, and a □WSE APL64 server instance are performing independent processing.

□WSE Client Instance Functions

APL64 programmer-defined ‘TestEH’ function will run in the APL64 client instance when progress, exception or completion events fire in the □WSE APL64 server instance.

```
TestEH arg
⌈⌋ 1⌋arg Instance Name
⌈⌋ 2⌋arg Context
⌈⌋ 3⌋arg Event Type
⌈⌋ 4⌋arg Event-provided Info

'Instance name: ',1⌋arg
'Context info: ', 2⌋arg
'Event type: ',3⌋arg
:SELECT 3⌋arg
:CASE 'Cancel'
'Asynchronous operation cancelled'
:CASE 'CompleteWithResult'
'Asynchronous operation completed with result: ', 4⌋arg
:CASE 'CompleteWithoutResult'
'Asynchronous operation completed without result'
:CASE 'Error'
'Error message: ',4⌋arg
:CASE 'Progress'
'Progress information: ', 4⌋arg
:ENDSELECT
```

```

VTestEH
0  TestEH arg
1  A 1>arg Instance Name
2  A 2>arg Context
3  A 3>arg Event Type
4  A 4>arg Event-provided Info
5
6  'Instance name: ',1>arg
7  'Context info: ', 2>arg
8  'Event type: ',3>arg
9  :SELECT 3>arg
10 :CASE 'Cancel'
11   'Asynchronous operation cancelled'
12 :CASE 'CompleteWithResult'
13   'Asynchronous operation completed with result: ', 4>arg
14 :CASE 'CompleteWithoutResult'
15   'Asynchronous operation completed without result'
16 :CASE 'Error'
17   'Error message: ',4>arg
18 :CASE 'Progress'
19   'Progress information: ', 4>arg
20 :ENDSELECT
21

```

APL64 programmer-defined 'Run' function which creates the ☐WSE instance and initiates the AsyncCall action. This function exists in the ☐WSE APL64 client instance.

```

Run
testOut<-"
<'# ' ☐WSE 'AsyncHandler' 'TestEH'
⌈⌋ Set the ☐WSE AsyncHandler function name

'# ' ☐WSE 'Create' 'wse1'
⌈⌋ Create the ☐WSE instance named 'wse1'

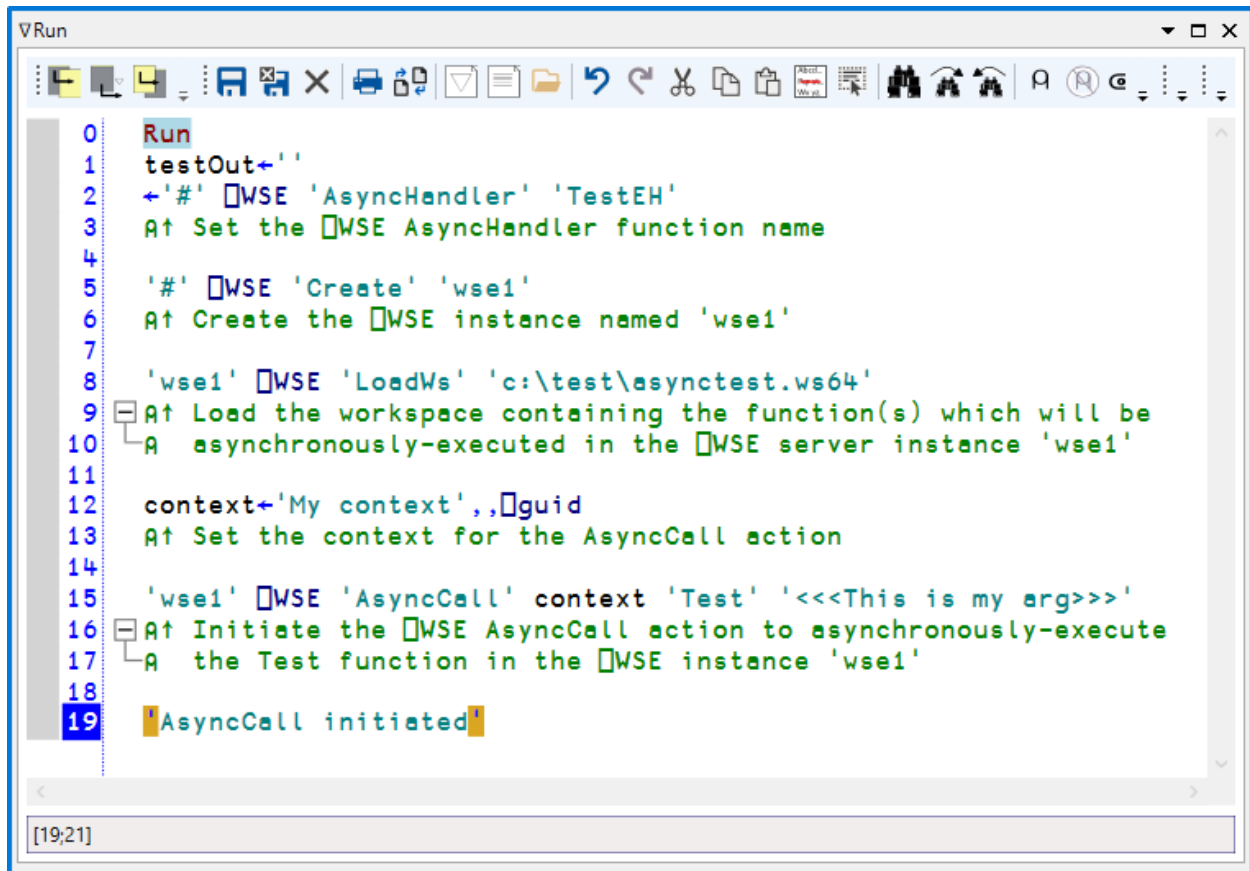
'wse1' ☐WSE 'LoadWs' 'c:\test\asynctest.ws64'
⌈⌋ Load the workspace containing the function(s) which will be
⌈⌋ asynchronously-executed in the ☐WSE server instance 'wse1'

context<'My context',, ☐guid
⌈⌋ Set the context for the AsyncCall action

'wse1' ☐WSE 'AsyncCall' context 'Test' '<<<This is my arg>>>'
⌈⌋ Initiate the ☐WSE AsyncCall action to asynchronously-execute
⌈⌋ the Test function in the ☐WSE instance 'wse1'

```

'AsyncCall initiated'



```
0 Run
1 testOut←''
2 ←'#' □WSE 'AsyncHandler' 'TestEH'
3 ⚡ Set the □WSE AsyncHandler function name
4
5 '# ' □WSE 'Create' 'wse1'
6 ⚡ Create the □WSE instance named 'wse1'
7
8 'wse1' □WSE 'LoadWs' 'c:\test\asynctest.ws64'
9 ⚡ Load the workspace containing the function(s) which will be
10 ⚡ asynchronously-executed in the □WSE server instance 'wse1'
11
12 context←'My context',.□guid
13 ⚡ Set the context for the AsyncCall action
14
15 'wse1' □WSE 'AsyncCall' context 'Test' '<<<This is my arg>>>'
16 ⚡ Initiate the □WSE AsyncCall action to asynchronously-execute
17 ⚡ the Test function in the □WSE instance 'wse1'
18
19 'AsyncCall initiated'
```

[19;21]

□WSE Server Instance Function

The APL64 programmer-defined 'Test' function, in the APL64 in the 'asynctest.ws64' workspace loaded into the □WSE APL64 server instance, will run the long-running process when the AsyncCall □WSE method is initiated. This function exists in the 'c:\test\asynctest.ws64' workspace which is loaded into the □WSE APL64 server instance. This function uses the APL64 □Progress system function to provide feedback from the □WSE APL64 server instance to the □WSE APL64 client instance

```
Result←Test arg
```

```
□progress 'Test function running: With right argument: ',⌥ □enlist arg
```

```
□progress 'Starting workload 1'
```

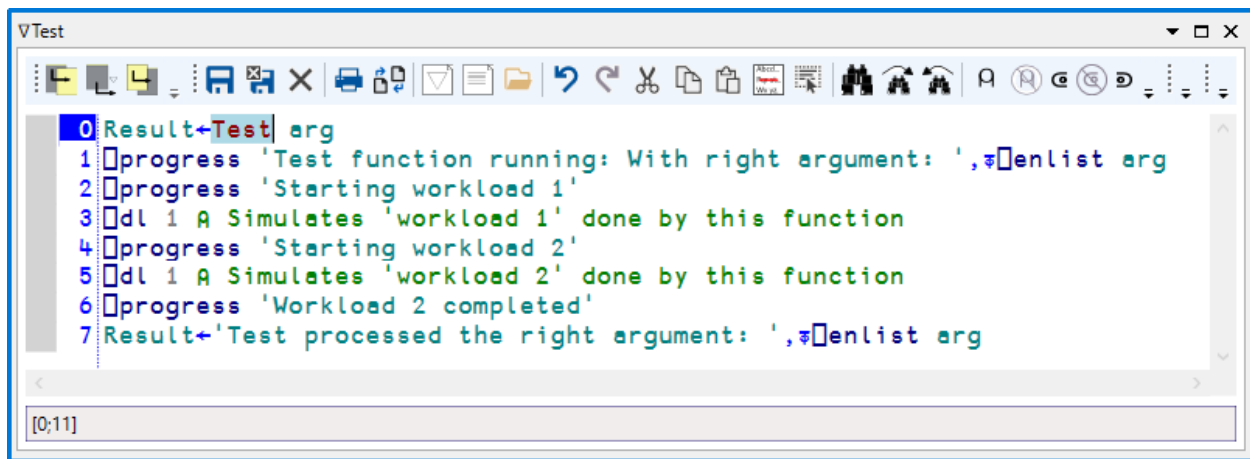
```
□dl 1 ⌥ Simulates 'workload 1' done by this function
```

```
□progress 'Starting workload 2'
```

```
□dl 1 ⌥ Simulates 'workload 2' done by this function
```

```
□progress 'Workload 2 completed'
```

```
Result←'Test processed the right argument: ',⌥ □enlist arg
```



```
0 Result←Test arg
1 ⌘progress 'Test function running: With right argument: ',⌘enlist arg
2 ⌘progress 'Starting workload 1'
3 ⌘dl 1 A Simulates 'workload 1' done by this function
4 ⌘progress 'Starting workload 2'
5 ⌘dl 1 A Simulates 'workload 2' done by this function
6 ⌘progress 'Workload 2 completed'
7 Result←'Test processed the right argument: ',⌘enlist arg
```

[0;11]

Test the Example

When the 'Run' function is executed in the ☐ WSE APL64 client instance:

- The ☐ WSE AsyncCall action is initiated
- The execution of the Run function continues to completion
- The Test function is asynchronously-executed in the ☐ WSE APL64 server instance
- The ☐ WSE 'Progress' event is fired several times during execution of the Test function, providing feedback from the ☐ WSE APL64 server instance to the ☐ WSE APL64 client instance.
- The Test function prepares the result and finishes execution.
- The ☐ WSE 'Completion' event fires.
- When the Progress and Completion events fire, the TestEH function is run in the ☐ WSE APL64 client instance. The implicit output of the TestEH function is rendered in the history pane of the ☐ WSE APL64 client instance which in this example is running the APL64 developer version.

```

0 Run
1 wse1
2 AsyncCall initiated
3 >[WSE:wse1;Progress] TestEH
4 Instance name: wse1
5 Context info: My context{440AB2B3-D591-4B27-B064-85F4BA22BA2B}
6 Event type: Progress
7 Progress information: Test function running: With right argument: <<<This is my arg>>>
8 >[WSE:wse1;Progress] TestEH
9 Instance name: wse1
10 Context info: My context{440AB2B3-D591-4B27-B064-85F4BA22BA2B}
11 Event type: Progress
12 Progress information: Starting workload 1
13 >[WSE:wse1;Progress] TestEH
14 Instance name: wse1
15 Context info: My context{440AB2B3-D591-4B27-B064-85F4BA22BA2B}
16 Event type: Progress
17 Progress information: Starting workload 2
18 >[WSE:wse1;Progress] TestEH
19 Instance name: wse1
20 Context info: My context{440AB2B3-D591-4B27-B064-85F4BA22BA2B}
21 Event type: Progress
22 Progress information: Workload 2 completed
23 >[WSE:wse1;CompleteWithResult] TestEH
24 Instance name: wse1
25 Context info: My context{440AB2B3-D591-4B27-B064-85F4BA22BA2B}
26 Event type: CompleteWithResult
27 Asynchronous operation completed with result: Test processed the right argument: <<<This is my arg>>>
28

```

Example: AsyncCall: Using Multiple WSE Servers

Sometimes an application needs to process multiple data sets of the same format, but with each data set containing different values. If the processing of a data set is an independent calculation, the APL64 client can create multiple WSE servers, and each APL64 server can process a data set.

APL64 Client Instance Functions

The 'Run' function creates the WSE servers and initiates the WSE AsyncCall actions on individual WSE servers.

Run;context;dataSet;nDataSets;nServers;serverName;waitTime

nDataSets←8

results←(nDataSets,2)ρ'

⌈ Global: results columns: Context(dataSet# WseServerName) Result

nServers←5

⌈ Depends on the capacity of the workstation

⌈ Check 7⇒SYSINIT

waitTime←20000

⌈ Adjust based on the anticipated maximum time to complete a server task

←'#' WSE 'AsyncHandler' 'ProcessEH'

⌈ Set the WSE AsyncHandler function name

:FOR I:IN nServers

serverName←'wse',⌈ I

←'#' WSE 'Create' serverName

```

serverName □ WSE 'LoadWs' 'c:\ProcessDataSet\ProcessDataSet.ws64'
:ENDFOR

:FOR I :IN inDataSets
serverNames←'#'□ WSE 'AsyncAvailableServerNames'
:IF (0<1↑pserverNames)
serverName←serverNames[1;]
:ELSE
'#'□ WSE 'AsyncWait' waitTime 0
Ⓢ↑ Wait until a □ WSE server instance is available
Ⓢ or until waitTime expires
:IF ('#'□ WSE 'AsyncTasksRunning')=nServers
'Exception: No available servers and waitTime expired'
'Increase waitTime or check for programming errors'
:RETURN
:ELSE
serverName←('#'□ WSE 'AsyncAvailableServerNames')[1;]
:ENDIF
:ENDIF
dataSet←'dataSet', Ⓢ I
context←I serverName
serverName □ WSE 'AsyncCall' context 'ProcessDataSet' dataSet
results[I;]←( Ⓢ context) "
:ENDFOR

```

```

VRun
0 Run;context;dataSet;nDataSets;nServers;serverName;waitTime
1
2 nDataSets←8
3 results←(nDataSets,2)p' '
4 A† Global: results columns: Context(dataSet# WseServerName) Result
5 nServers←5
6 A† Depends on the capacity of the workstation
7 A Check 7→SYSINIT
8 waitTime←20000
9 A† Adjust based on the anticipated maximum time to complete a server task
10
11 ←'#' □WSE 'AsyncHandler' 'ProcessEH'
12 A† Set the □WSE AsyncHandler function name
13
14 □:FOR I :IN nServers
15   serverName←'wse',I
16   ←'#' □WSE 'Create' serverName
17   serverName □WSE 'LoadWs' 'c:\ProcessDataSet\ProcessDataSet.ws64'
18 :ENDFOR
19
20 □:FOR I :IN nDataSets
21   serverNames←'#'□WSE 'AsyncAvailableServerNames'
22   :IF (0<1pserverNames)
23     serverName←serverNames[1;]
24   :ELSE
25     '#'□WSE 'AsyncWait' waitTime 0
26     A† Wait until a □WSE server instance is available
27     A or until waitTime expires
28     :IF ('#'□WSE 'AsyncTasksRunning')=nServers
29       'Exception: No available servers and waitTime expired'
30       'Increase waitTime or check for programming errors'
31     :RETURN
32   :ELSE
33     serverName←('#'□WSE 'AsyncAvailableServerNames')[1;]
34   :ENDIF
35 :ENDIF
36   dataSet←'dataSet',I
37   context←I serverName
38   serverName □WSE 'AsyncCall' context 'ProcessDataSet' dataSet
39   results[I;]←(context) ''
40 :ENDFOR

```

The 'ProcessEH' function handles the callback, exception and complete events fired by the □WSE servers.

Z←ProcessEH arg;context;result

- ⌚ 1→arg Instance Name
- ⌚ 2→arg Context
- ⌚ 3→arg Event Type: 'Cancel', 'Completion', 'Exception', or 'Progress'
- ⌚ 4→arg Event-provided Info
- ⌚ Event Type Event-provided Info
- ⌚ =====
- ⌚ Completion Result of the asynchronously-executed APL function
- ⌚ Exception Error message
- ⌚ Progress Progress information

⌈ result: application-specific results array

⌈↑ Global: results columns: Context(dataSet# WseServerName) Result

'Instance name: ',1⊃arg

'Context: ',⌈ ENLIST context←2⊃arg

'Event type: ',3⊃arg

:SELECT 3⊃arg

:CASE 'Cancel'

'Async function execution cancelled'

:CASE 'CompleteWithResult'

'Async function completed with result: ',⌈ ENLIST result←4⊃arg

results[1↑context;2]←⌈ result

:CASE 'CompleteWithoutResult'

'Async function completed without result'

:CASE 'Error'

'Error message: ',⌈ ENLIST 4⊃arg

:CASE 'Progress'

'Progress information: ',⌈ ENLIST 4⊃arg

:ENDSELECT

```

VProcessEH
0  Z←ProcessEH arg;context;result
1  A 1>arg Instance Name
2  A 2>arg Context
3  A 3>arg Event Type: 'Completion', 'Exception', or 'Progress'
4  A 4>arg Event-provided Info
5  A Event Type Event-provided Info
6  A =====
7  A Completion Result of the asynchronously-executed APL function
8  A Exception Error message
9  A Progress Progress information
10
11 A result: application-specific results array
12 A Global: results columns: Context(dataSet# WseServerName) Result
13
14 'Instance name: ',1>arg
15 'Context: ',⌈ENLIST context←2>arg
16 'Event type: ',3>arg
17 :SELECT 3>arg
18 :CASE 'Cancel'
19   'Async function execution cancelled'
20 :CASE 'CompleteWithResult'
21   'Async function completed with result: ',⌈ENLIST result←4>arg
22   results[1↑context;2]←⌈result
23 :CASE 'CompleteWithoutResult'
24   'Async function completed without result'
25 :CASE 'Error'
26   'Error message: ',⌈ENLIST 4>arg
27 :CASE 'Progress'
28   'Progress information: ',⌈ENLIST 4>arg
29 :ENDSELECT

```

APL64 Server Instance Function

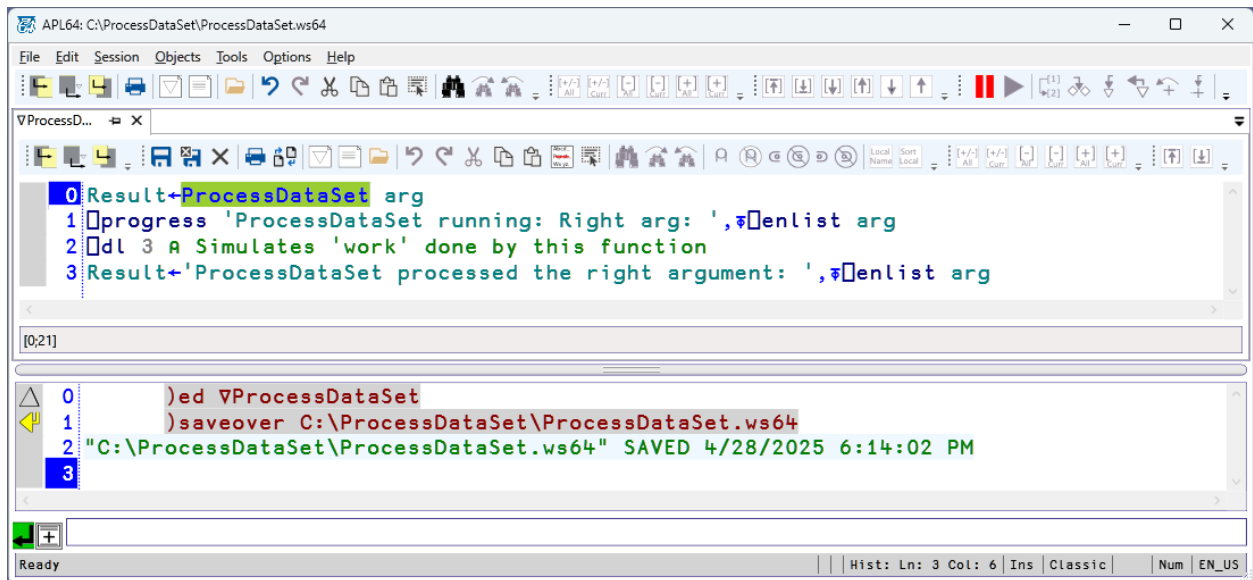
The 'ProcessDataSet' function, in the 'ProcessDataSet.ws64' workspace loaded into each ☐WSE APL64 server instance, processes the data set information provided by the AsyncCall action, uses the ☐Progress system function if necessary, and throws exceptions if necessary.

Result←ProcessDataSet arg

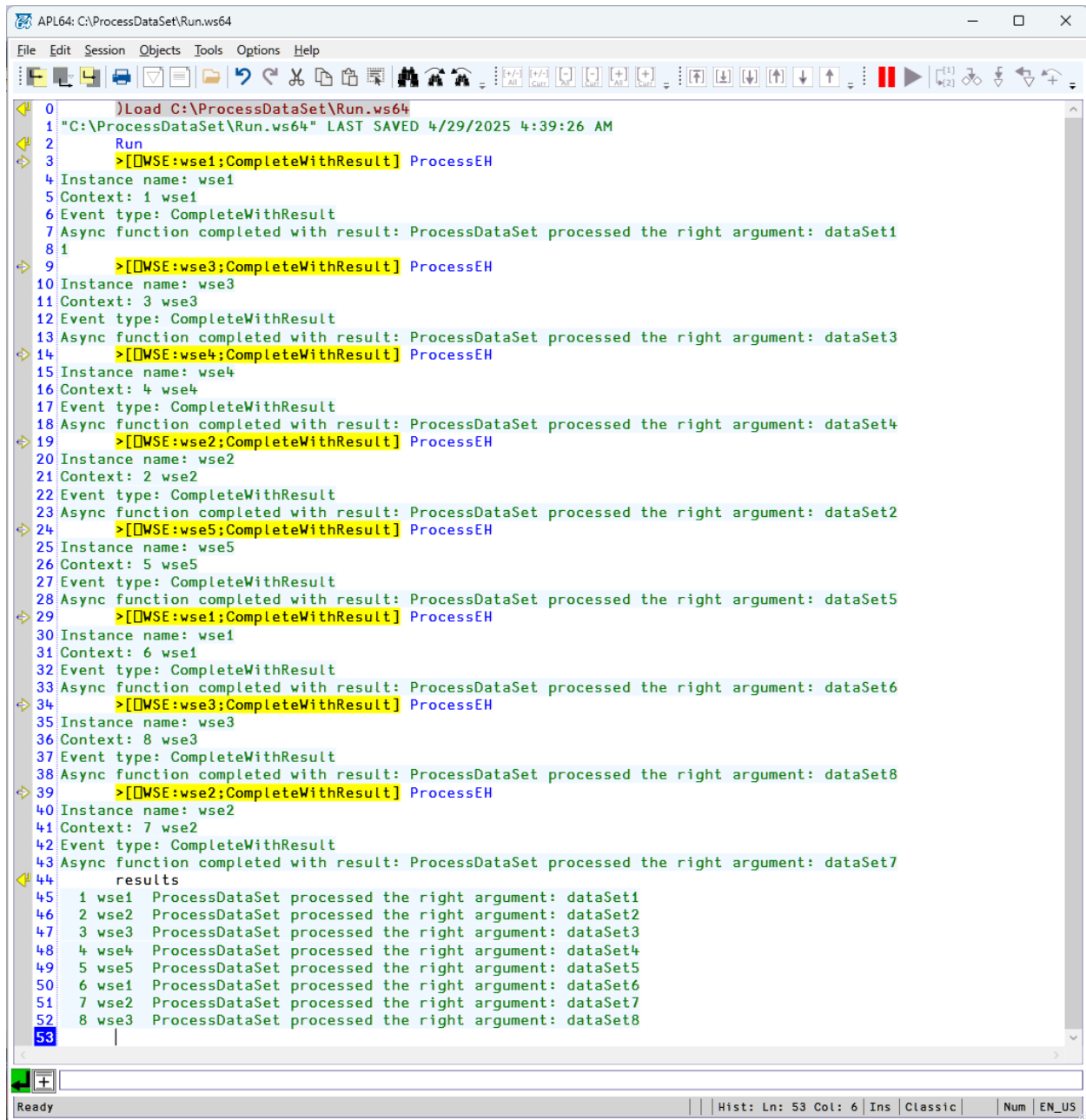
☐progress 'ProcessDataSet running: Right arg: ',⌈☐enlist arg

☐dl 3 ☐ Simulates 'work' done by this function

Result←'ProcessDataSet processed the right argument: ',⌈☐enlist arg



Test the Example



```
APL64: C:\ProcessDataSet\Run.ws64
File Edit Session Objects Tools Options Help
[Icons] [Zoom] [Find] [Run] [Stop] [Break] [Undo] [Redo] [Copy] [Paste] [Print] [Help]
0  )Load C:\ProcessDataSet\Run.ws64
1  "C:\ProcessDataSet\Run.ws64" LAST SAVED 4/29/2025 4:39:26 AM
2  Run
3  >[QWSE:wse1;CompleteWithResult] ProcessEH
4  Instance name: wse1
5  Context: 1 wse1
6  Event type: CompleteWithResult
7  Async function completed with result: ProcessDataSet processed the right argument: dataSet1
8  1
9  >[QWSE:wse3;CompleteWithResult] ProcessEH
10 Instance name: wse3
11 Context: 3 wse3
12 Event type: CompleteWithResult
13 Async function completed with result: ProcessDataSet processed the right argument: dataSet3
14 >[QWSE:wse4;CompleteWithResult] ProcessEH
15 Instance name: wse4
16 Context: 4 wse4
17 Event type: CompleteWithResult
18 Async function completed with result: ProcessDataSet processed the right argument: dataSet4
19 >[QWSE:wse2;CompleteWithResult] ProcessEH
20 Instance name: wse2
21 Context: 2 wse2
22 Event type: CompleteWithResult
23 Async function completed with result: ProcessDataSet processed the right argument: dataSet2
24 >[QWSE:wse5;CompleteWithResult] ProcessEH
25 Instance name: wse5
26 Context: 5 wse5
27 Event type: CompleteWithResult
28 Async function completed with result: ProcessDataSet processed the right argument: dataSet5
29 >[QWSE:wse1;CompleteWithResult] ProcessEH
30 Instance name: wse1
31 Context: 6 wse1
32 Event type: CompleteWithResult
33 Async function completed with result: ProcessDataSet processed the right argument: dataSet6
34 >[QWSE:wse3;CompleteWithResult] ProcessEH
35 Instance name: wse3
36 Context: 8 wse3
37 Event type: CompleteWithResult
38 Async function completed with result: ProcessDataSet processed the right argument: dataSet8
39 >[QWSE:wse2;CompleteWithResult] ProcessEH
40 Instance name: wse2
41 Context: 7 wse2
42 Event type: CompleteWithResult
43 Async function completed with result: ProcessDataSet processed the right argument: dataSet7
44 results
45 1 wse1 ProcessDataSet processed the right argument: dataSet1
46 2 wse2 ProcessDataSet processed the right argument: dataSet2
47 3 wse3 ProcessDataSet processed the right argument: dataSet3
48 4 wse4 ProcessDataSet processed the right argument: dataSet4
49 5 wse5 ProcessDataSet processed the right argument: dataSet5
50 6 wse1 ProcessDataSet processed the right argument: dataSet6
51 7 wse2 ProcessDataSet processed the right argument: dataSet7
52 8 wse3 ProcessDataSet processed the right argument: dataSet8
53
```

Example: Stochastic Numerical Integration using Multiple $\square$ WSE Instances

Math Details

In this example a simple form of [numerical integration](#) is implemented using multiple $\square$ WSE server instances based on the [mean value theorem for integrals](#).

The value of the integral of a continuous, differentiable, real-valued function $f(x)$ over the range $[a, b]$ can be estimated using:

$$\int_a^b dx f(x) = \frac{b-a}{M} \sum_{i=1}^M f(X_i)$$

To increase the accuracy of the estimate, the value of M is generally large. Using multiple □WSE server instances to obtain the evaluation arguments {X_i}, and the values of the function for those arguments {f(X_i)} will significantly improve performance of the estimate process.

The evaluation arguments are selected pseudo-randomly over the interval [a, b] with a ≤ b:

$a + (b-a) \times r_i$, where r_i is a uniform pseudo random number in the range [0, 1]

The selection of a numerical integration method is beyond the scope of this document.

Specific Example

An [elliptic integral](#) which has no explicit result:

$$\int_0^1 \sqrt{1-x^4} dx$$

□WSE Implementation using AsyncCall Action

- The □WSE client Run function creates a pool of □WSE servers
- The □WSE client Run function creates the pseudo-random X_i function evaluation points
- The □WSE client Run function requests the □WSE servers to compute the function values for a subset of the evaluation points
- The □WSE server ProcessDataSet function performs the calculations, and provides the results to the □WSE client via client-side events
- The □WSE client ProcessEH event handler function accumulates the results using the mean value theorem for integrals.

In this example the classic APL roll (?) function is used to obtain the pseudo-random function evaluation points. In a production environment when the number of function evaluation points is very large, the periodicity of the roll function may make it unsuitable. In this case, the APL64 □PDIST system function makes available longer period, pseudo-random number generators.

This example assumes that all servers are tasked with processing requests necessary for this example. If additional pools of □WSE servers are tasked with unrelated processing requests, the example APL functions will need modification to carefully consider the □WSE server instance names. For example, the □WSE 'AsyncWait' action applies to all □WSE server instances created by the same □WSE client APL instance.

APL64 □WSE Client Instance Functions

The 'Run' function creates the □WSE servers and initiates the □WSE AsyncCall actions on individual □WSE servers.

The 'context' value used for the AsyncCall action is selected to simplify the consolidation of the results from the □WSE server instances.

Run

M;context;l;nServers;serverName;serverNames;waitTime;Xi;FunctionValuesTotals;NFunctionValuesCalcd;W;nEvalPoints

⌚ M: The number of □WSE client requests to be made to the □WSE servers

⌚ The Run function is designed so that M can exceed the number of □WSE servers (nServers)

nEvalPoints←10000

⌚ nEvalPoints: Number of function evaluations for each □WSE client request to a □WSE server

□rl←7*5

FunctionValuesTotals←Mp0

NFunctionValuesCalcd←Mp0

⌚ Application-specific, □WSE client side, global variables to contain □WSE server results

nServers←5

⌚↑ Depends on the capacity of the workstation

⌚ Check 7⇒□SYSINIT

waitTime←3 ⌚ 3 seconds

⌚↑ Adjust based on the anticipated maximum time to complete a server task

←'#' □WSE 'AsyncHandler' 'ProcessEH'

⌚↑ Set the □WSE AsyncHandler function name

'#'□WSE 'Clear'

:FOR I :IN nServers

serverName←'wse', ⌘ I

←'#' □WSE 'Create' serverName

'*** Created server: ',serverName

serverName □WSE 'LoadWs' 'C:\WseNumInteg.ws64'

:ENDFOR

⌚ Run until all M cases have been started

I←0

:While I<M

⌚ Get available server names

serverNames←C[2]'#'□WSE 'AsyncAvailableServerNames'

□←'>>>> AsyncAvailableServerNames:' serverNames

⌚ Assign required tasks to available □WSE servers until all

⌚ servers have been started OR all cases have been assigned

:While 0<pserverNames

:And I<M

⌚ Get next server to be assigned

I←I+1

serverName←↑serverNames

serverNames←1↓serverNames

```

    ⌈ Create argument, context, and call server to process data
    Xi ← (1 + nEvalPoints * 1E9) / 1E9
    ⌈ nEvalPoints evaluation points of the function in range [0,1]
    ⌈ Pseudo-random number in range [0,1]
    context ← I serverName
    '*** Sending Calc Request #' (⌈ I), ' to server ', ⌈ serverName
    serverName ⌈ WSE 'AsyncCall' context 'ProcessDataSet' Xi
:EndWhile

```

```

    ⌈ Allow event handlers to run and wait for one or more servers to finish running
    ⌈ The wait time is application-specific
    '*** Waiting for servers to fire events and/or at least one to finish'
    W ← '#' ⌈ WSE 'AsyncWait' waitTime
    '*** AsyncWait returned ', ⌈ W
:EndWhile

```

```

    ⌈ Wait (up to 100 seconds) for all tasks to finish running
    ⌈ The wait for all server-side tasks to complete is application-specific
    '*** Waiting for all tasks to finish running'
    W ← '#' ⌈ WSE 'AsyncWait' 100 1 1
    '*** Final AsyncWait returned ', ⌈ W
    '*** Done waiting for all tasks to finish running'
    'Estimated value of the integral: ', ⌈ (+ / FunctionValuesTotals) / NFunctionValuesCalcd
    'NFunctionValuesCalcd: ', ⌈ + / NFunctionValuesCalcd

```

```

VRun
0 Run M;context;I;nServers;serverName;serverNames;waitTime;Xi;FunctionValuesTotals;NFunctionValuesCalcd;W;nEvalPoints
1 A M: The number of QWSE client requests to be made to the QWSE servers
2 A The Run Function is designed so that M can exceed the number of QWSE servers (nServers)
3
4 nEvalPoints=10000
5 A nEvalPoints: Number of function evaluations for each QWSE client request to a QWSE server
6
7 Qrl=7*5
8 FunctionValuesTotals=Mp0
9 NFunctionValuesCalcd=Mp0
10 A Application-specific, QWSE client side, global variables to contain QWSE server results
11
12 nServers=5
13 A Depends on the capacity of the workstation
14 A Check 7>QSYSINIT
15
16 waitTime=3 A 3 seconds
17 A Adjust based on the anticipated maximum time to complete a server task
18
19 +'# QWSE 'AsyncHandler' 'ProcessEH'
20 A Set the QWSE AsyncHandler function name
21
22 '#QWSE 'Clear'
23 :FOR I :IN nServers
24   serverName='vse',*I
25   +'# QWSE 'Create' serverName
26   '*** Created server: ',serverName
27   serverName QWSE 'LoadWs' 'C:\VseNumInteg.ws64'
28 :ENDFOR
29
30 A Run until all M cases have been started
31 I=0
32 :While I<M
33   A Get available server names
34   serverNames+=([2] '#QWSE 'AsyncAvailableServerNames'
35   Q->>>> AsyncAvailableServerNames:' serverNames
36
37   A Assign required tasks to available QWSE servers until all
38   A servers have been started OR all cases have been assigned
39   :While 0<pserverNames
40     :And I<M
41     A Get next server to be assigned
42     I=I+1
43     serverName=tserverNames
44     serverNames+=1serverNames
45
46     A Create argument, context, and call server to process data
47     Xi=(1+nEvalPoints?1E9)+1E9
48     A nEvalPoints evaluation points of the function in range [0,1]
49     A Pseudo-random number in range [0,1]
50     context=I serverName
51     '*** Sending Calc Request #'(*I),' to server ',*serverName
52     serverName QWSE 'AsyncCall' context 'ProcessDataSet' Xi
53   :EndWhile
54
55   A Allow event handlers to run and wait for one or more servers to finish running
56   A The wait time is application-specific
57   '*** Waiting for servers to fire events and/or at least one to finish'
58   W='# QWSE 'AsyncWait' waitTime
59   '*** AsyncWait returned ',*W
60 :EndWhile
61
62 A Wait (up to 100 seconds) for all tasks to finish running
63 A The wait for all server-side tasks to complete is application-specific
64 '*** Waiting for all tasks to finish running'
65 W='# QWSE 'AsyncWait' 100 1 1
66 '*** Final AsyncWait returned ',*W
67 '*** Done waiting for all tasks to finish running'
68 'Estimated value of the integral: ',*(+/FunctionValuesTotals)/NFunctionValuesCalcd
69 'NFunctionValuesCalcd: ',*+/NFunctionValuesCalcd

```

The 'ProcessEH' function handles the callback, exception and complete events fired by the QWSE servers:

ProcessEH arg;context;result

- ⌚ Information provided by a QWSE server is transmitted to the QWSE client via events.
- ⌚ ProcessEH is an event handler which runs in the QWSE client every time a QWSE server:
- ⌚ (a) Completes the server-side task requested by the QWSE client

- ⌚ (b) Receives info from the □WSE server via server-side □PROGRESS use
- ⌚ (c) Experiences a server-side exception
- ⌚ (d) Has its processing cancelled by the □WSE client

- ⌚ 1⌚arg Instance Name
- ⌚ 2⌚arg Context
- ⌚ 3⌚arg Event Type: 'Completion', 'Exception', or 'Progress'
- ⌚ 4⌚arg Info from the □WSE server
- ⌚ Event Type Info Description

- ⌚ -----
- ⌚ (a) Completion Result of the asynchronously-executed, server-side APL function
- ⌚ (b) Exception Exception message from the server side
- ⌚ (c) Progress Progress information from the server side
- ⌚ (d) Cancellation

```
'Instance name: ',1⌚arg
'Context: ',⌚ □ENLIST context←2⌚arg
'Event type: ',3⌚arg
:SELECT 3⌚arg
:CASE 'Cancel'
'Async function execution cancelled'
:CASE 'CompleteWithResult'
'Async function completed with result: ',⌚ □ENLIST result←4⌚arg
FunctionValuesTotals[1⌚context]←2⌚result
NFunctionValuesCalcd[1⌚context]←1⌚result
⌚↑ Assign the result to the global variable in the □WSE client instance
:CASE 'CompleteWithoutResult'
'Async function completed without result'
:CASE 'Error'
'Error message: ',⌚ □ENLIST 4⌚arg
:CASE 'Progress'
'Progress information: ',⌚ □ENLIST 4⌚arg
:ENDSELECT
```

```

VProcessEH
0 ProcessEH arg;context;result
1 A Information provided by a WSE server is transmitted to the WSE client via events.
2 A ProcessEH is an event handler which runs in the WSE client every time a WSE server:
3 A (a) Completes the server-side task requested by the WSE client
4 A (b) Receives info from the WSE server via server-side PROGRESS use
5 A (c) Experiences a server-side exception
6 A (d) Has its processing cancelled by the WSE client
7
8
9 A 1>arg Instance Name
10 A 2>arg Context
11 A 3>arg Event Type: 'Completion', 'Exception', or 'Progress'
12 A 4>arg Info from the WSE server
13 A Event Type Info Description
14 A -----
15 A (a) Completion Result of the asynchronously-executed, server-side APL function
16 A (b) Exception Exception message from the server side
17 A (c) Progress Progress information from the server side
18 A (d) Cancellation
19
20 'Instance name: ',1>arg
21 'Context: ',*ENLIST context+2>arg
22 'Event type: ',3>arg
23 :SELECT 3>arg
24 :CASE 'Cancel'
25 'Async function execution cancelled'
26 :CASE 'CompleteWithResult'
27 'Async function completed with result: ',*ENLIST result+4>arg
28 FunctionValuesTotals[1>context]+2>result
29 NFunctionValuesCalcd[1>context]+1>result
30 At Assign the result to the global variable in the WSE client instance
31 :CASE 'CompleteWithoutResult'
32 'Async function completed without result'
33 :CASE 'Error'
34 'Error message: ',*ENLIST 4>arg
35 :CASE 'Progress'
36 'Progress information: ',*ENLIST 4>arg
37 :ENDSELECT

```

[37;11] Commit Changes Commit & Close

APL64 WSE Server Instance Function

The workspace containing the 'ProcessDataSet' function is loaded into each WSE APL64 server instance. The ProcessDataSet function processes the data set information provided by the AsyncCall action, uses the WProgress system function, if necessary, throws exceptions, if necessary, and returns the result which will be used to obtain the integral estimate.

```
result←ProcessDataSet Xi;fValues
```

⌈ ProcessDataSet runs in a WSE server

⌈ progress 'ProcessDataSet running: Calculating $\int_0^1 f(x) dx$, ' function values'

⌈ Inform the WSE client that the ProcessDataSet function is running

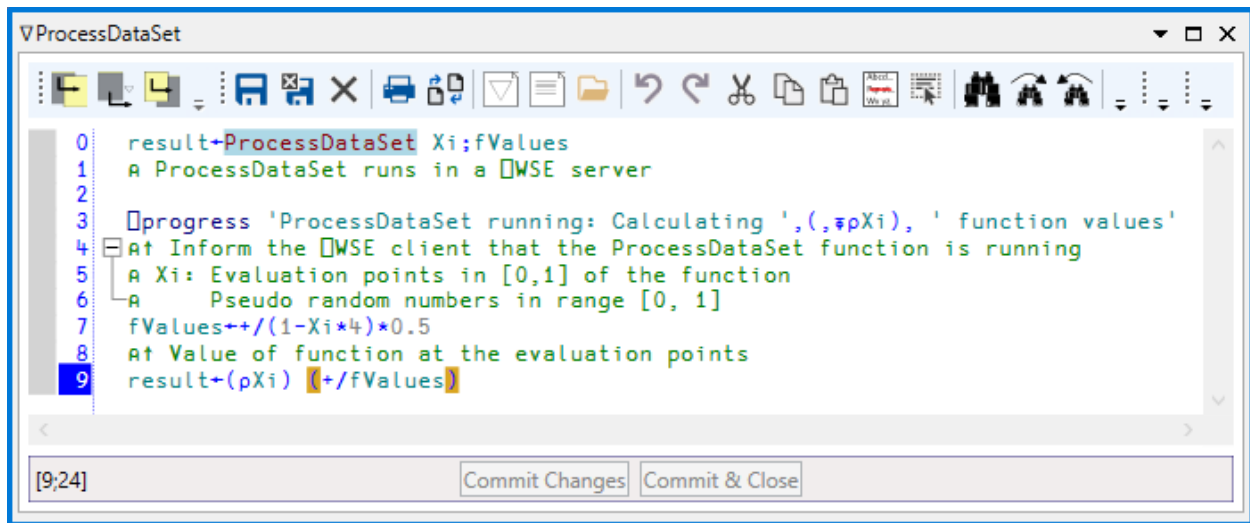
⌈ Xi: Evaluation points in [0,1] of the function

⌈ Pseudo random numbers in range [0, 1]

```
fValues←+/(1-Xi*4)*0.5
```

⌈ Value of function at the evaluation points

```
result←(pXi) (+/fValues)
```



The screenshot shows a window titled 'ProcessDataSet' with a toolbar and a code editor. The code editor contains the following APL code:

```
0 result←ProcessDataSet Xi:fValues
1 A ProcessDataSet runs in a WSE server
2
3 ⎕progress 'ProcessDataSet running: Calculating ',(⊆pXi), ' function values'
4 At Inform the WSE client that the ProcessDataSet function is running
5 A Xi: Evaluation points in [0,1] of the function
6 A Pseudo random numbers in range [0, 1]
7 fValues←+/(1-Xi*4)*0.5
8 At Value of function at the evaluation points
9 result←(pXi) (+/fValues)
```

At the bottom of the window, there is a status bar showing '[9;24]' and two buttons: 'Commit Changes' and 'Commit & Close'.

Test the Implementation

In this example:

- The WSE client instance and server instance functions are contained in the same workspace: 'c:\WseNumInteg\WseNumInteg.w64'. In a production environment, the workspaces used by the client and each server can be different.
- The feedback displayed in the APL64 Developer GUI from the WSE server instances to the WSE client instance would be used for debugging purposes only.
- The WSE client runs the Run function with a right argument of 10, indicating that 10 data sets will be processed asynchronously by the pool of WSE servers. When all the server-side processing is complete, the WSE client presents the result.

Here an excerpt of the output illustrating the result:

APL64: C:\WseNumInteg.ws64

File Edit Session Objects Tools Options Help

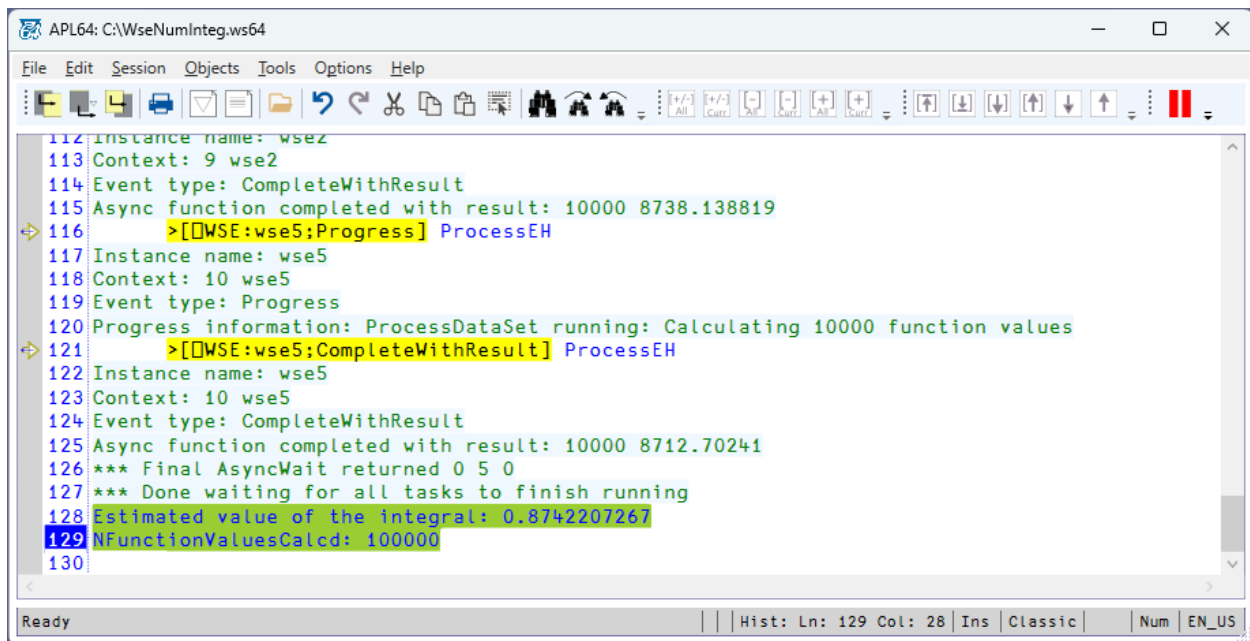
Run 10

```

1 *** Created server: wse1
2 *** Created server: wse2
3 *** Created server: wse3
4 *** Created server: wse4
5 *** Created server: wse5
6 >>>> AsyncAvailableServerNames: wse5 wse4 wse3 wse2 wse1
7 *** Sending Calc Request # 1 to server wse5
8 *** Sending Calc Request # 2 to server wse4
9 *** Sending Calc Request # 3 to server wse3
10 *** Sending Calc Request # 4 to server wse2
11 *** Sending Calc Request # 5 to server wse1
12 *** Waiting for servers to fire events and/or at least one to finish
13 *** AsyncWait returned 1 4 5
14 >>>> AsyncAvailableServerNames: wse5 wse4 wse3 wse2
15 *** Sending Calc Request # 6 to server wse5
16 *** Sending Calc Request # 7 to server wse4
17 *** Sending Calc Request # 8 to server wse3
18 *** Sending Calc Request # 9 to server wse2
19 *** Waiting for servers to fire events and/or at least one to finish
20 *** AsyncWait returned 1 4 9
21 >>>> AsyncAvailableServerNames: wse5 wse4 wse3 wse1
22 *** Sending Calc Request # 10 to server wse5
23 *** Waiting for servers to fire events and/or at least one to finish
24 *** AsyncWait returned 1 4 10
25 *** Waiting for all tasks to finish running
26 >[DWSE:wse5;Progress] ProcessEH
27 Instance name: wse5
28 Context: 1 wse5
29 Event type: Progress
30 Progress information: ProcessDataSet running: Calculating 10000 function values
31 >[DWSE:wse5;CompleteWithResult] ProcessEH
32 Instance name: wse5
33 Context: 1 wse5
34 Event type: CompleteWithResult
35 Async function completed with result: 10000 8747.514525
36 >[DWSE:wse4;Progress] ProcessEH
37 Instance name: wse4
38 Context: 2 wse4
39 Event type: Progress

```

Ready | Hist: Ln: 0 Col: 0 | Ins | Classic | Num | EN_US

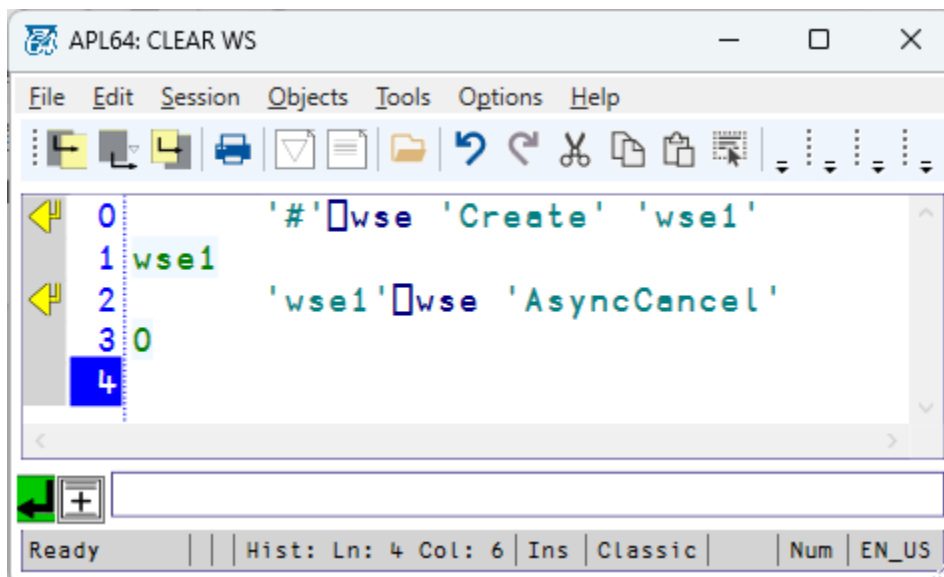


```
112 Instance name: wse2
113 Context: 9 wse2
114 Event type: CompleteWithResult
115 Async function completed with result: 10000 8738.138819
116 >[WSE:wse5;Progress] ProcessEH
117 Instance name: wse5
118 Context: 10 wse5
119 Event type: Progress
120 Progress information: ProcessDataSet running: Calculating 10000 function values
121 >[WSE:wse5;CompleteWithResult] ProcessEH
122 Instance name: wse5
123 Context: 10 wse5
124 Event type: CompleteWithResult
125 Async function completed with result: 10000 8712.70241
126 *** Final AsyncWait returned 0 5 0
127 *** Done waiting for all tasks to finish running
128 Estimated value of the integral: 0.8742207267
129 NFunctionValuesCalcd: 100000
130
```

AsyncCancel

Cancel the asynchronous operation, if any, currently in progress in this ☐WSE instance. This action, if possible, is not instantaneous, but instead occurs when the APL64 interpreter determines it is appropriate to end the execution of the APL64 programmer-defined function running in ☐WSE server APL instance.

bool ← instanceName ☐WSE 'AsyncCancel'



```
0      '#[WSE 'Create' 'wse1'
1 wse1
2      'wse1'[WSE 'AsyncCancel'
3 0
4
```

AsyncStatus

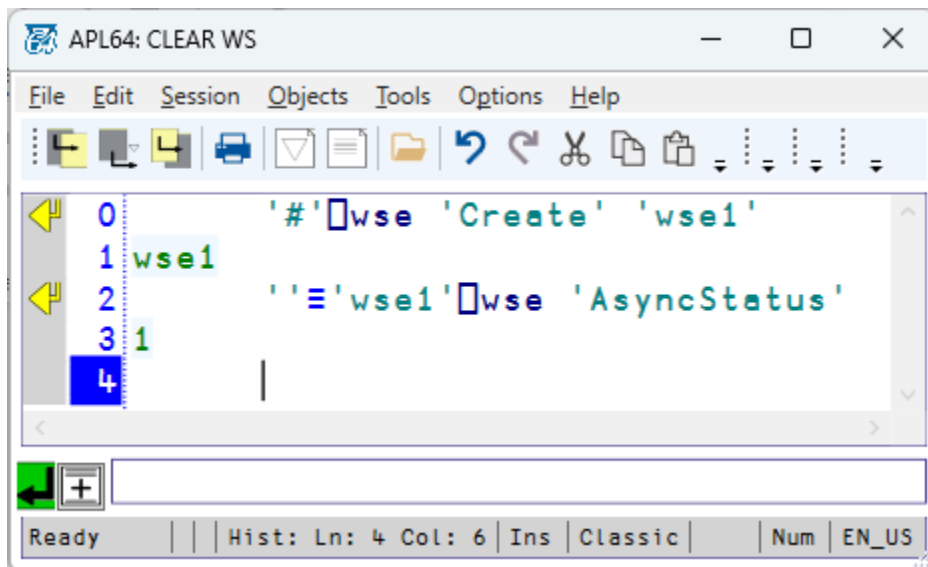
Obtain the status of the asynchronous operation, if any, currently in progress in this `WSE` instance. This information is transient, as requesting this information does not modify the running state of an asynchronous operation.

```
res ← instanceName WSE 'AsyncStatus'
```

res is an APL64 character vector. If no `WSE` asynchronous operation is in progress, the result is "".

Status values for an existing .Net asynchronous operation: Created, WaitingForActivation, WaitingToRun, Running, WaitingForChildrenToComplete, RanToCompletion, Cancelled, Faulted. Not all status states are possible in APL64.

```
'#' WSE 'Create' 'wse1'  
'wse1' WSE 'AsyncStatus'
```



```
def 'AsyncWseObjectEventEH eventInfo' 'eventInfo'  
'#' WSE 'AsyncHandler' 'AsyncWseObjectEventEH'  
'#' WSE 'Create' 'wse1'  
'wse1' WSE 'SysCall' 'DEF' ('Z←MyAsyncFn rarg' '←DL 1500' 'Z←"MyAsyncFn ran: ", ⍺ rarg')  
context←'Calculation id#1'  
fnName←'MyAsyncFn'  
rArg←123  
'wse1' WSE 'AsyncCall' context fnName rArg ⋄ 'wse1' WSE 'AsyncStatus'
```

```

0 | def 'AsyncWseObjectEventEH eventInfo' 'eventInfo'
1 | AsyncWseObjectEventEH
2 |   '# ' WSE 'AsyncHandler' 'AsyncWseObjectEventEH'
3 |
4 |   '# ' Wse 'Create' 'wse1'
5 | wse1
6 |   'wse1' Wse 'SysCall' 'DEF' ('Z+MyAsyncFn rarg' '+DL 1500' 'Z+MyAsyncFn ran: ",rarg')
7 | MyAsyncFn
8 |   context←'Calculation id#1'
9 |   fnName←'MyAsyncFn'
10 |   rArg←123
11 |   'wse1' Wse 'AsyncCall' context fnName rArg 'wse1' Wse 'AsyncStatus'
12 | Running
13 |

```

AsyncTaskIsRunning

Boolean value indicating if an asynchronous operation is currently in progress in this ☐WSE instance. This information is transient, as requesting this information does not modify the running state of an asynchronous operation.

res ← instanceName ☐WSE 'AsyncTaskIsRunning'

```

0 | '# ' Wse 'Create' 'wse1'
1 | wse1
2 |   '=' 'wse1' Wse 'AsyncTaskIsRunning'
3 | 0
4 |

```

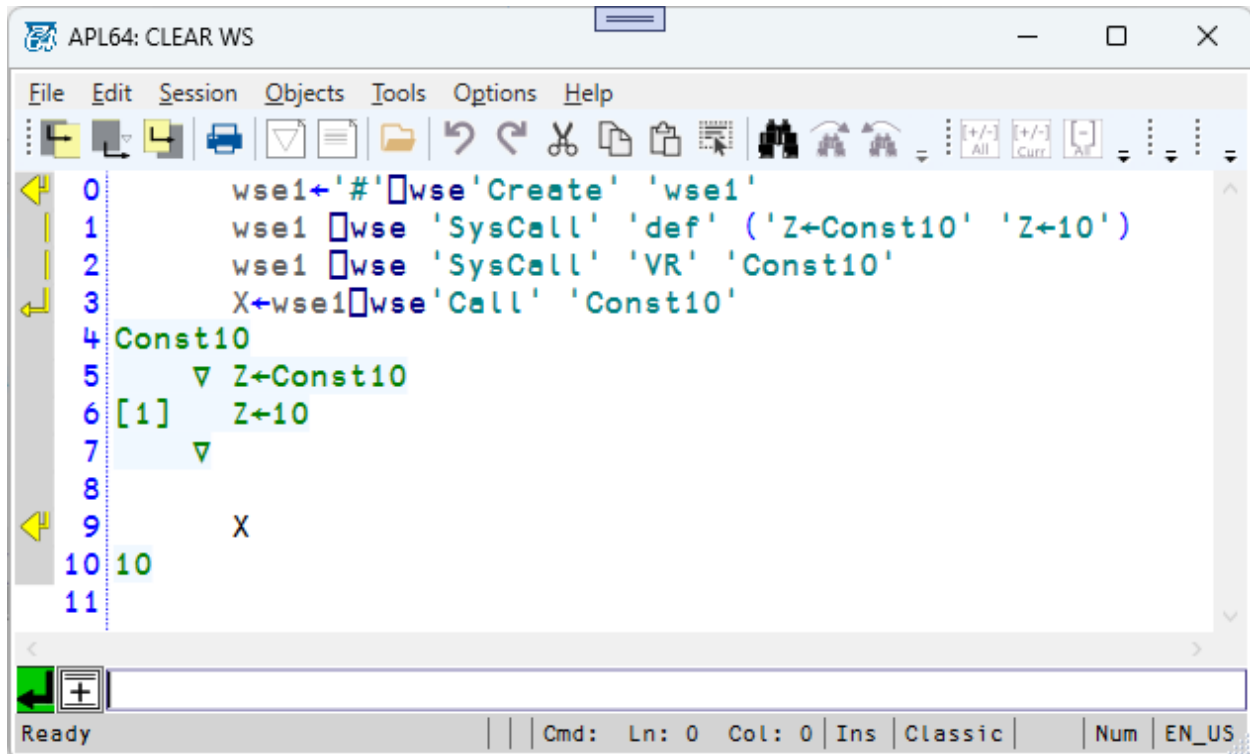
Call

Obtain the result, if any, of the interpreter execution of the specified APL64 programmer-defined function. The ☐WSE Call action has three overloads to execute niladic, monadic or dyadic APL64 programmer-defined functions. The specified APL64 programmer-defined function name may be a character scalar, character vector or string scalar. Because this function will run in the specified ☐WSE instance, and a ☐WSE instance does not support a GUI, implicit output of this function, if any, is not visible.

Niladic

res ← instanceName □ WSE 'Call' 'NiladicFn'

```
wse1←'#'□wse'Create' 'wse1'  
wse1 □wse 'SysCall' 'def' ('Z←Const10' 'Z←10')  
wse1 □wse 'SysCall' 'VR' 'Const10'  
X←wse1□wse'Call' 'Const10'
```



Monadic

res ← instanceName □ WSE 'Call' 'MonadicFn' RightArg

```
wse1←'#'□wse'Create' 'wse1'  
wse1 □wse 'SysCall' 'def' ('Z←Add10 X' 'Z←10+X')  
wse1 □wse 'SysCall' 'VR' 'Add10'  
□dr□←X←wse1□wse'Call' 'Add10' 100
```

```

0 wse1←'#'⊞wse'Create' 'wse1'
1 wse1 ⊞wse 'SysCall' 'def' ('Z←Add10 X' 'Z←10+X')
2 wse1 ⊞wse 'SysCall' 'VR' 'Add10'
3 ⊞dr⊞X←wse1⊞wse'Call' 'Add10' 100
4 Add10
5   ▽ Z←Add10 X
6 [1]   Z←10+X
7   ▽
8
9 110
10 323
11

```

Ready | Cmd: Ln: 0 Col: 0 Ins | Classic | Num | EN_US

Dyadic

The left argument to the APL64 programmer-defined function is specified after the right argument to that function: `res ← instanceName ⊞WSE 'Call' 'DyadicFn' RightArg LeftArg`

```

wse1←'#'⊞wse'Create' 'wse1'
wse1 ⊞wse 'SysCall' 'def' ('Z←L CAT R' 'Z←L,R')
wse1 ⊞wse 'SysCall' 'VR' 'CAT'
⊞dr⊞X←wse1⊞wse'Call' 'CAT' 'Right' 'Left'

```

```

0 wse1←'#'⊞wse'Create' 'wse1'
1 wse1 ⊞wse 'SysCall' 'def' ('Z←L CAT R' 'Z←L,R')
2 wse1 ⊞wse 'SysCall' 'VR' 'CAT'
3 ⊞dr⊞X←wse1⊞wse'Call' 'CAT' 'Right' 'Left'
4 CAT
5   ▽ Z←L CAT R
6 [1]   Z←L,R
7   ▽
8
9 LeftRight
10 82
11

```

Ready | Hist: Ln: 3 Col: 11 Ins | Classic | Num | EN_US

CallNow

The □WSE CallNow action can send information from the APL client to a specified □WSE server instance. The □WSE CallNow action will interrupt any □WSE server processing which may be underway.

Syntax: result←instanceName □wse 'CallNow' [fnName] [fnRightArg] [fnLeftArg]

fnName is the name of the function in the workspace currently loaded into the □WSE instance. This function will run in the □WSE server scope when the APL client which created the □WSE instance uses the □WSE CallNow action.

fnRightArg is the optional right argument of the function in the workspace.

fnLeftArg is the optional left argument of the function in the workspace.

result is the response, if any, of the □WSE server upon the receipt of the information sent by the APL client using the □WSE CallNow action.

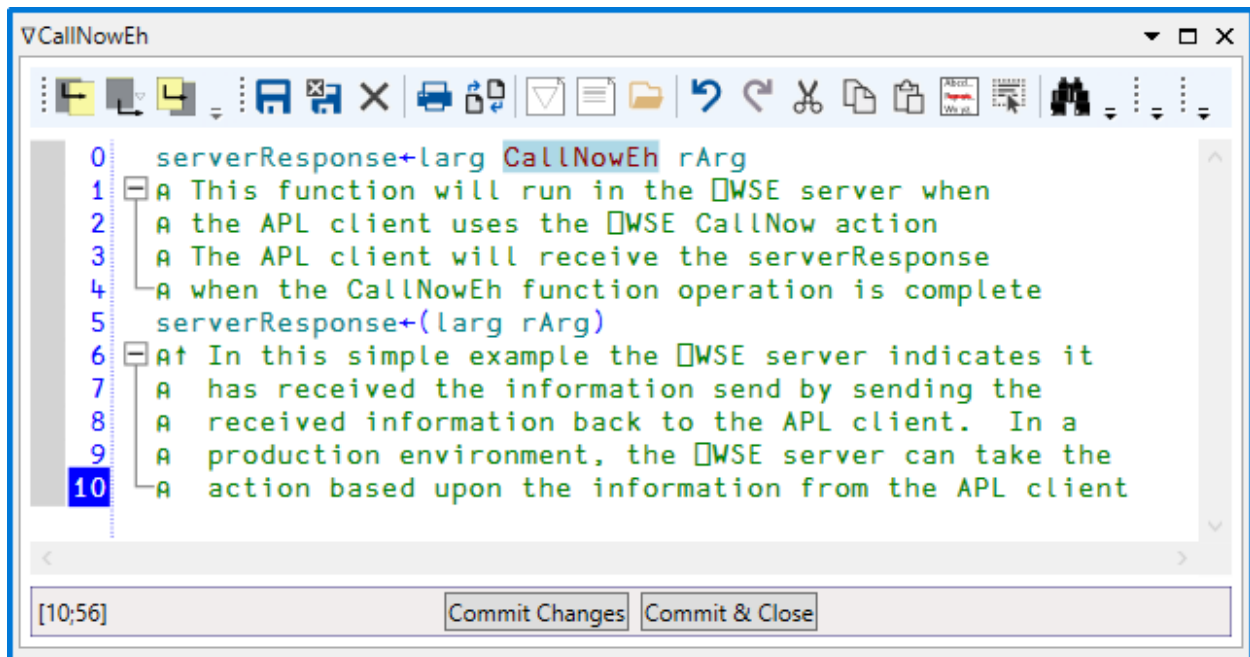
Example: □WSE CallNow synchronous environment

In this example the APL client creates a □WSE server, which is not running an processing requests. The APL client sends information to the □WSE server using the □WSE CallNow action.

- Create a workspace, CallNowEx1.ws64
- Create the CallNowEh function in the CallNowEx1 workspace
- Save the CallNowEh.ws64 workspace
- Create a □WSE server instance named 'wse1'
- Load the 'CallNowEx1' workspace into the □WSE server
- In the APL client scope run the □WSE CallNow action
- The CallNow action will cause the CallNowEh function to run in the □WSE server
- The CallNowEh function will create a global variable in the scope of the □WSE server
- The APL client will check that the global variable was created in the □WSE server

APL client use of the □WSE CallNow action:

```
serverResponse←larg CallNowEh rArg
⌈ This function will run in the □WSE server when
⌈ the APL client uses the □WSE CallNow action
⌈ The APL client will receive the serverResponse
⌈ when the CallNowEh function operation is complete
serverResponse←(larg rArg)
⌈↑ In this simple example the □WSE server indicates it
⌈ has received the information send by sending the
⌈ received information back to the APL client. In a
⌈ production environment, the □WSE server can take the
⌈ action based upon the information from the APL client
```



Run this example:

```

wse1←'#' WSE 'Create' 'wse1'
wse1 WSE 'LoadWs' 'c:\CallNowEx1.ws64'
wse1 WSE 'SysCall' 'wsid'
lNowEx1.ws64
rArg←ι4
lArg←«abcd» (2 3⍲6)
wse1 WSE 'SysCall' 'idlist' 3
serverResponse←wse1 WSE 'CallNow' 'CallNowEh' rArg lArg
(lArg rArg)≡serverResponse

```

```

APL64: C:\CallNowEx1.ws64
File Edit Session Objects Tools Options Help
[Icons]
0 | wse1←'#' □WSE 'Create' 'wse1'
1 | wse1 □WSE 'LoadWs' 'c:\CallNowEx1.ws64'
2 | wse1 □WSE 'SysCall' 'wsid'
3 | c:\CallNowEx1.ws64
4 | rArg←14
5 | lArg←«abcd» (2 3p16)
6 | wse1 □WSE 'SysCall' 'idlist' 3
7 | CallNowEh
8 | lArg
9 | rArg
10 | wse1
11 | serverResponse←wse1 □WSE 'CallNow' 'CallNowEh' rArg lArg
12 | (lArg rArg)≡serverResponse
13 | 1
14 |
Ready | Hist: Ln: 0 Col: 0 Ins | Classic | Num | EN_US

```

Example: □WSE CallNow asynchronous parallel environment

In this example the APL client creates □WSE servers which will process APL client data sets. The processing will be asynchronous and parallel because the APL client uses the □WSE CallAsync and AsyncWait actions to request the processing from the □WSE servers. While a □WSE server is processing an APL client request, the APL client uses the □WSE CallNow action to send information to the □WSE server. The processing of the APL client's request is temporarily paused for the □WSE server to receive this information from the APL client. Based upon the information received by the □WSE server, the processing of the APL client's request can be continued or stopped.

- Create a workspace c:\CallNowEx2.ws64
- Create the ProcessClientRequest function in the workspace. This function simulates processing to satisfy the APL client's requests. This function checks the local 'run' variable to stop running based on the information sent by the APL client using the □WSE CallNow action.
- Create the CallNowEH function in the workspace. This function which will run when the APL client uses the □WSE CallNow action to send information to the □WSE server. When this function runs it modifies the 'run' variable so that the ProcessClientRequest function can take appropriate action.
- Save the CallNowEh.ws64 workspace
- Create the ProcessEH function in the scope of the APL client. This function will be run when the □WSE server has completed the processing of an APL client's request.
- Create the Run function in the scope of the APL client. This function:
 - Creates the □WSE server instances. For the purposes of this example only one □WSE server is created called 'wse1'.

- Loads the 'CallNowEx2' workspace into the □WSE server
- Submits the processing requests in a loop using the □WSE AsyncCall action. Each request causes the ProcessClientRequest function to run in the □WSE server. While running the ProcessClientRequest function checks the variables 'now' and 'run', to take appropriate action. For the purposes of this example there is only one processing request.
- Within the loop, uses the □WSE CallNow several times to send information from the APL client to the □WSE server. This illustrates how this action can affect the processing of the APL client's request by the □WSE server. This action causes the CallNowEH function to run in the □WSE server creating the 'now' variable and updating the 'run' variable.

```

res←ProcessClientRequest;run;n;now;i
run←1
⌈⌋↑ To be updated by APL client using □WSE CallNow
n←0
now←0
⌈⌋↑ # uses of □WSE CallNow by APL client
:While run
⌈⌋↑ Check if the APL client has used □WSE CallNow to stop
n←n+1
⌈⌋↑ Increment iteration counter
:For i :ln 16
  ←(11)0.×11
  ⌈⌋↑ Simulate processing of APL client request
:EndFor
:EndWhile
:Return 'Done' n now □si
⌈⌋↑ APL client's processing request is satisfied

```

```

VProcessClientRequest
0  res←ProcessClientRequest;run;n;now;i
1  run←1
2  A↑ To be updated by APL client using □WSE CallNow
3  n←0
4  now←0
5  A↑ # uses of □WSE CallNow by APL cleint
6  □While run
7  A↑ Check if the APL client has used □WSE CallNow to stop
8      n←n+1
9      A↑ Increment iteration counter
10     □For i □In 116
11         ←(11)□.×11
12         A↑ Simulate processing of APL client request
13     □EndFor
14 □EndWhile
15 □Return 'Done' n now □si
16 A↑ APL client's processing request is satisfied

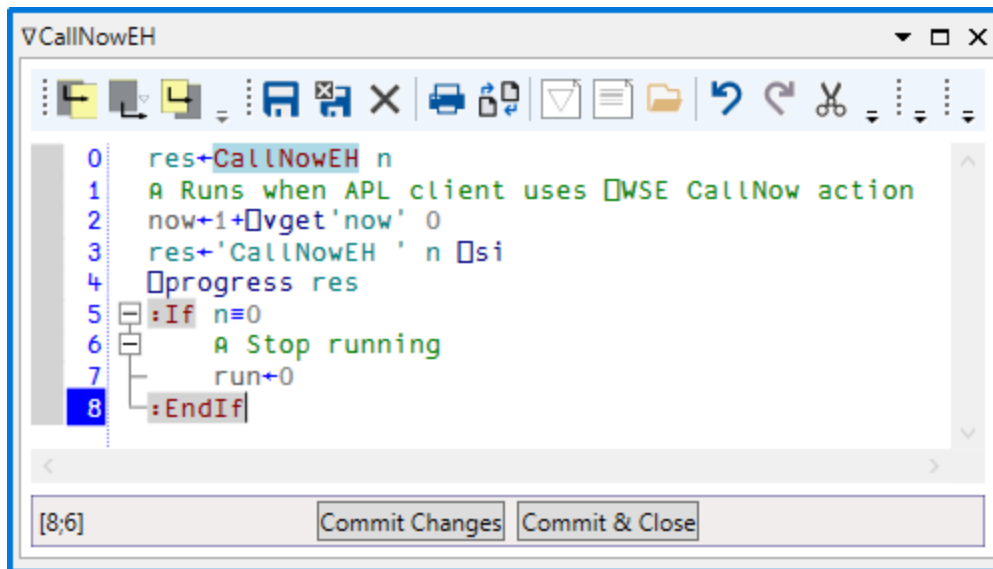
```

[16:47] Commit Changes Commit & Close

```

res←CallNowEH n
⊙ Runs when APL client uses □WSE CallNow action
now←1+□vget'now' 0
res←'CallNowEH ' n □si
□progress res
:If n≠0
    ⊙ Stop running
    run←0
:EndIf

```



ProcessEH arg;context;result

- ⌈ 1⌈arg Instance Name
- ⌈ 2⌈arg Context
- ⌈ 3⌈arg Event Type
- ⌈ 4⌈arg Event-provided Info
- ⌈ For the purposes of this example the result of the
- ⌈ the server side processing request (arg) is simply
- ⌈ displayed in the scope of the APL client

```
:SELECT 3⌈arg
:CASE 'Cancel'
  'Request cancelled'
:CASE 'CompleteWithResult'
  'Request Completed With Result:' (4⌈arg)
:CASE 'CompleteWithoutResult'
  'Request Completed Without Result'
:CASE 'Error'
  'Error: ' (4⌈arg)
:CASE 'Progress'
  'Progress:' (4⌈arg)
:ENDSELECT
```

```

0 ProcessEH arg;context;result
1 A 1>arg Instance Name
2 A 2>arg Context
3 A 3>arg Event Type
4 A 4>arg Event-provided Info
5 A For the purposes of this example the result of the
6 A the server side processing request (arg) is simply
7 A displayed in the scope of the APL client
8
9 :SELECT 3>arg
10 :CASE 'Cancel'
11 'Request cancelled'
12 :CASE 'CompleteWithResult'
13 'Request Completed With Result:' (4>arg)
14 :CASE 'CompleteWithoutResult'
15 'Request Completed Without Result'
16 :CASE 'Error'
17 'Error: ' (4>arg)
18 :CASE 'Progress'
19 'Progress: ' (4>arg)
20 :ENDSELECT

```

[0;0] Commit Changes Commit & Close

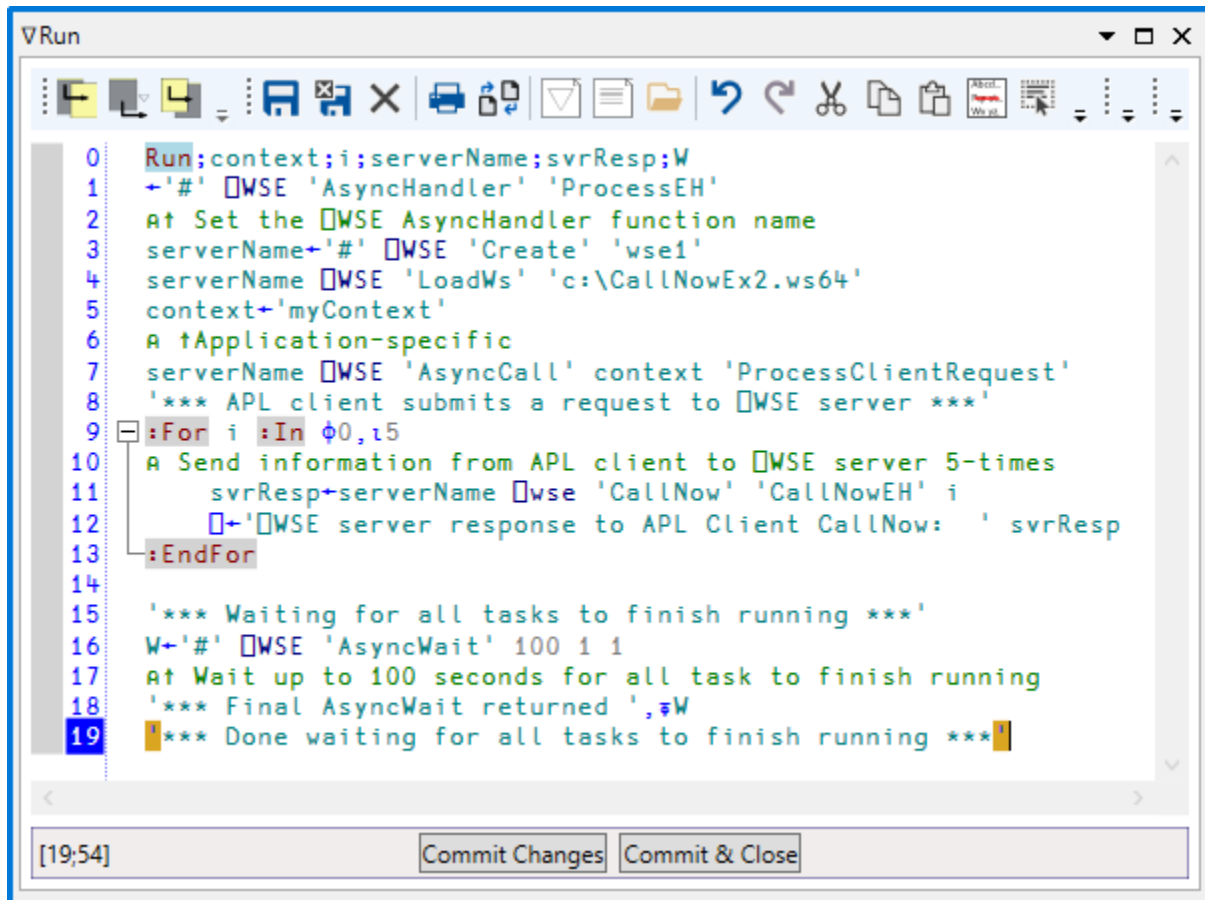
```

Run;context;i;serverName;svrResp;W
<-'#' WSE 'AsyncHandler' 'ProcessEH'
⌚↑ Set the WSE AsyncHandler function name
serverName<-'#' WSE 'Create' 'wse1'
serverName WSE 'LoadWs' 'c:\CallNowEx2.ws64'
context<'myContext'
⌚↑ Application-specific
serverName WSE 'AsyncCall' context 'ProcessClientRequest'
'*** APL client submits a request to WSE server ***'
:For i :In 0,1,5
⌚ Send information from APL client to WSE server 5-times
  svrResp<serverName WSE 'CallNow' 'CallNowEH' i
  W<'WSE server response to APL Client CallNow: ' svrResp
:EndFor

'*** Waiting for all tasks to finish running ***'
W<'#' WSE 'AsyncWait' 100 1 1
⌚↑ Wait up to 100 seconds for all task to finish running

```

```
*** Final AsyncWait returned ',φW  
*** Done waiting for all tasks to finish running ***'
```



```
0 Run;context;i;serverName;svrResp;W  
1 -'#' □WSE 'AsyncHandler' 'ProcessEH'  
2 At Set the □WSE AsyncHandler function name  
3 serverName←'#' □WSE 'Create' 'wse1'  
4 serverName □WSE 'LoadWs' 'c:\CallNowEx2.ws64'  
5 context←'myContext'  
6 A †Application-specific  
7 serverName □WSE 'AsyncCall' context 'ProcessClientRequest'  
8 '*** APL client submits a request to □WSE server ***'  
9 □:For i :In φ0,15  
10 A Send information from APL client to □WSE server 5-times  
11 svrResp←serverName □wse 'CallNow' 'CallNowEH' i  
12 □- '□WSE server response to APL Client CallNow: ' svrResp  
13 :EndFor  
14  
15 '*** Waiting for all tasks to finish running ***'  
16 W←'#' □WSE 'AsyncWait' 100 1 1  
17 At Wait up to 100 seconds for all task to finish running  
18 '*** Final AsyncWait returned ',φW  
19 □*** Done waiting for all tasks to finish running ***□
```

[19;54] Commit Changes Commit & Close

The output of this example illustrates that while asynchronous parallel processing by a □WSE server of an APL client's request is underway, the □WSE CallNow action can send information to that □WSE server and the □WSE server can response appropriately to that information and report back to the APL client the □WSE server's response.

```
APL64: C:\CallNowEx2.ws64
File Edit Session Objects Tools Options Help

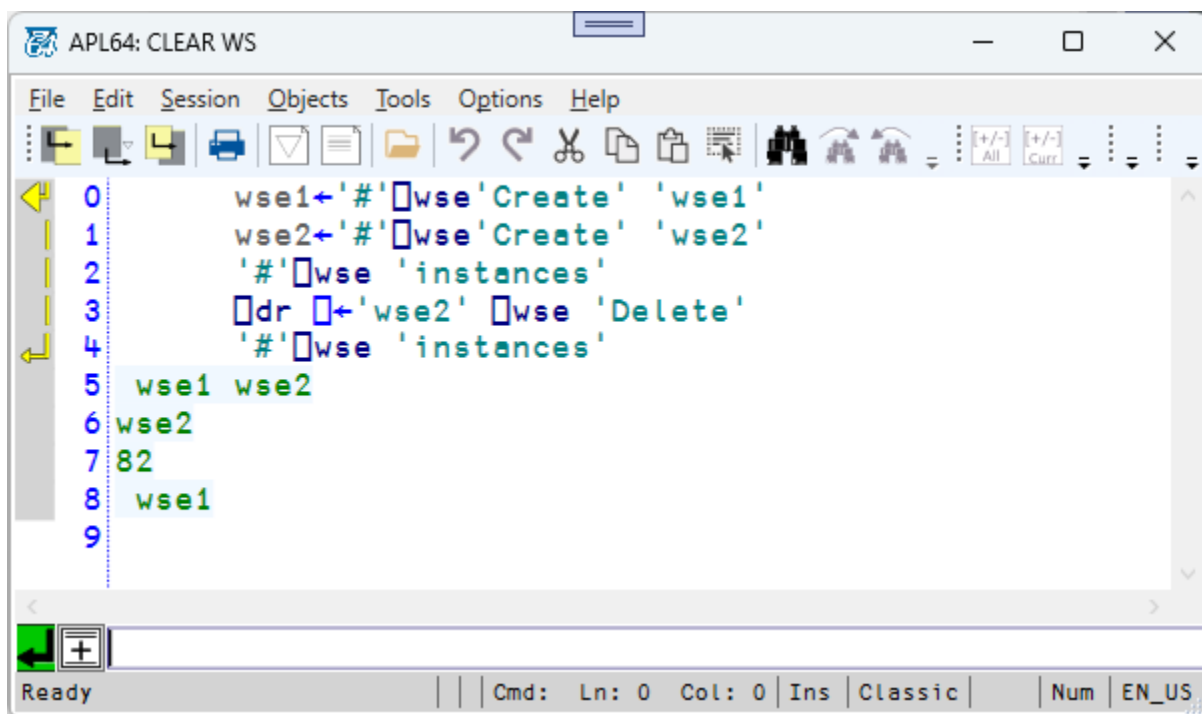
0 Run
1 *** APL client submits a request to WSE server ***
2 >[WSE:wse1;Progress] ProcessEH
3 Progress: CallNowEH 5 CallNowEH[3]
4 >[WSE:CallNow;CallNowEH]
5 ProcessClientRequest[1]
6 ???
7 WSE server response to APL Client CallNow: CallNowEH 5 CallNowEH[3]
8 >[WSE:CallNow;CallNowEH]
9 ProcessClientRequest[1]
10 ???
11 >[WSE:wse1;Progress] ProcessEH
12 Progress: CallNowEH 4 CallNowEH[3]
13 >[WSE:CallNow;CallNowEH]
14 ProcessClientRequest[11]
15 ???
16 WSE server response to APL Client CallNow: CallNowEH 4 CallNowEH[3]
17 >[WSE:CallNow;CallNowEH]
18 ProcessClientRequest[11]
19 ???
20 >[WSE:wse1;Progress] ProcessEH
21 Progress: CallNowEH 3 CallNowEH[3]
22 >[WSE:CallNow;CallNowEH]
23 ProcessClientRequest[11]
24 ???
25 WSE server response to APL Client CallNow: CallNowEH 3 CallNowEH[3]
26 >[WSE:CallNow;CallNowEH]
27 ProcessClientRequest[11]
28 ???
29 >[WSE:wse1;Progress] ProcessEH
30 Progress: CallNowEH 2 CallNowEH[3]
31 >[WSE:CallNow;CallNowEH]
32 ProcessClientRequest[11]
33 ???
34 WSE server response to APL Client CallNow: CallNowEH 2 CallNowEH[3]
35 >[WSE:CallNow;CallNowEH]
36 ProcessClientRequest[11]
37 ???
38 >[WSE:wse1;Progress] ProcessEH
39 Progress: CallNowEH 1 CallNowEH[3]
40 >[WSE:CallNow;CallNowEH]
41 ProcessClientRequest[11]
42 ???
43 WSE server response to APL Client CallNow: CallNowEH 1 CallNowEH[3]
44 >[WSE:CallNow;CallNowEH]
45 ProcessClientRequest[11]
46 ???
47 >[WSE:wse1;Progress] ProcessEH
48 Progress: CallNowEH 0 CallNowEH[3]
49 >[WSE:CallNow;CallNowEH]
50 ProcessClientRequest[11]
51 ???
52 WSE server response to APL Client CallNow: CallNowEH 0 CallNowEH[3]
53 >[WSE:CallNow;CallNowEH]
54 ProcessClientRequest[11]
55 ???
56 *** Waiting for all tasks to finish running ***
57 >[WSE:wse1;CompleteWithResult] ProcessEH
58 Request Completed With Result: Done 763 5 ProcessClientRequest[15]
59 ???
60 *** Final AsyncWait returned 0 1 0
61 *** Done waiting for all tasks to finish running ***
62
```

Delete

Delete the `WSE` instance with the specified instance name. The specified instance name may be a character scalar, character vector or string scalar. An exception will be thrown if the named instance does not exist. The `wse Delete` action is a character vector containing the name of the deleted `wse` instance.

instanceName `WSE 'Delete'`

```
wse1←'#'wse'Create' 'wse1'  
wse2←'#'wse'Create' 'wse2'  
'#'  
dr←'wse2' wse'Delete'  
'#'
```

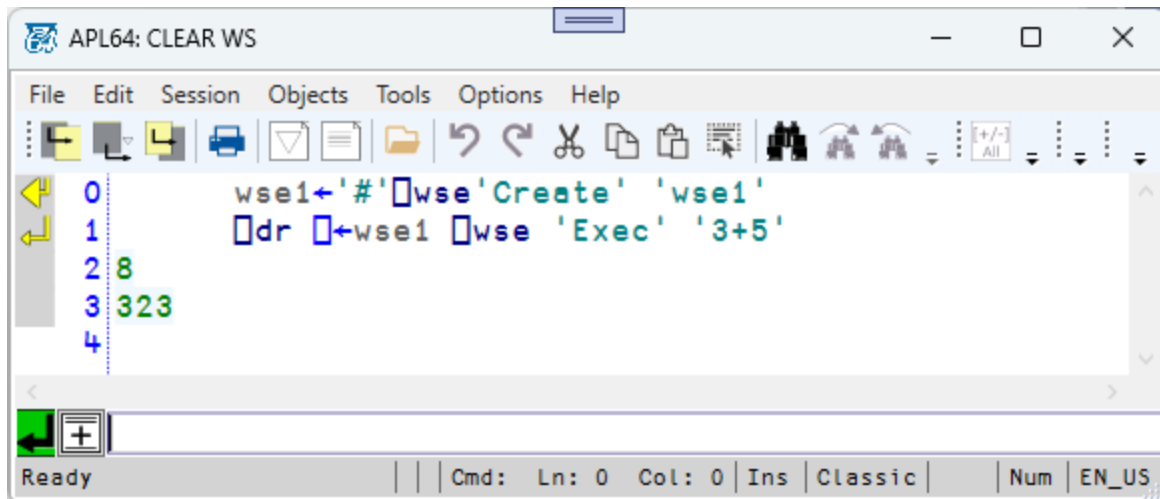


Exec

Obtain the result, if any, of the interpreter evaluation of the specified Expression. The specified expression may be a character scalar, character vector or string scalar.

res ← instanceName `WSE 'Exec' 'Expression'`

```
wse1←'#'wse'Create' 'wse1'  
dr←wse1 wse'Exec' '3+5'
```



GetSysVariable

Synonym: GetSysVar

Obtain the value of the specified APL64 ('quad') system variable. The specified APL64 system variable name may be a character scalar, character vector or string scalar. When specifying the APL64 system variable name, do not include the quad ([]) prefix.

```
res ← instanceName []WSE 'GetSysVariable' 'variableName'
```

Refer to the SetSysVariable action documentation for an example of the GetSysVariable action.

GetVariable

Synonym: GetVar

Obtain the value of the specified APL64 programmer-defined variable in a []wse instance. The specified APL64 programmer-defined variable name may be a character scalar, character vector or string scalar. Attempting to assign the result of the []WSE GetVariable action for a non-existent variable, will result in a value error exception.

```
res ← instanceName []WSE 'GetVariable' 'variableName'
```

Refer to the SetVariable action documentation for an example of the GetVariable action.

LoadWs

Load the specified workspace into the []WSE server APL instance.

```
instanceName []WSE 'LoadWs' wsPath
```

wsPath is a text value containing the full path to the workspace to be loaded.

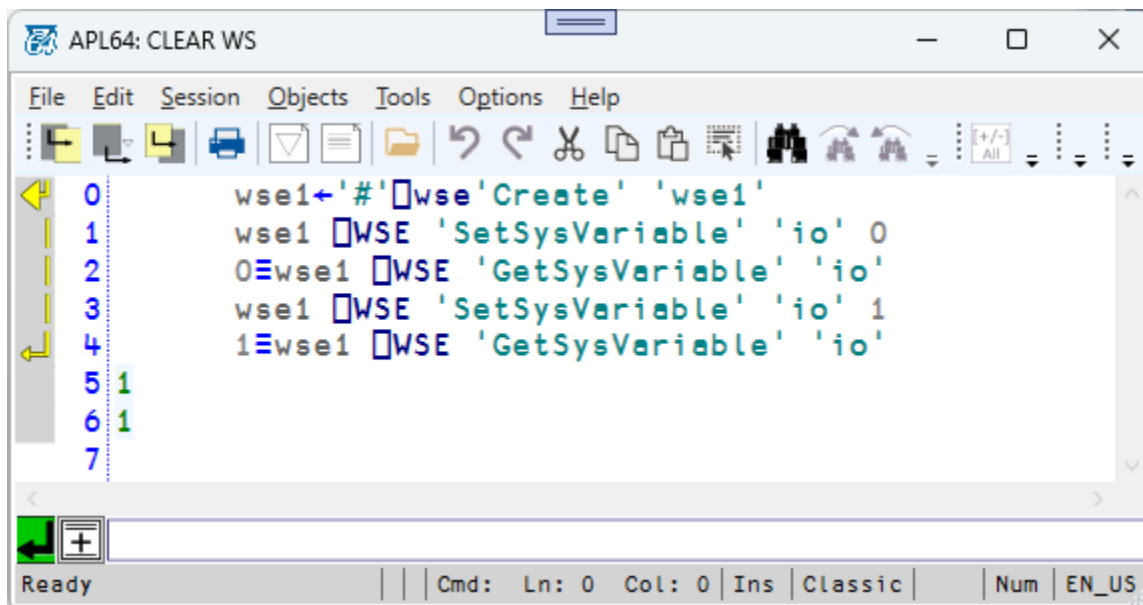
SetSysVariable

Synonym: SetSysVar

Set the value of the specified APL64 ('quad') system variable. The specified APL64 system variable name may be a character scalar, character vector or string scalar. When specifying the APL64 system variable name, do not include the quad (⎕) prefix.

instanceName ⎕WSE 'SetSysVariable' 'variableName' variableValue

```
wse1←'#'⎕wse'Create' 'wse1'  
wse1 ⎕WSE 'SetSysVariable' 'io' 0  
0≡wse1 ⎕WSE 'GetSysVariable' 'io'  
wse1 ⎕WSE 'SetSysVariable' 'io' 1  
1≡wse1 ⎕WSE 'GetSysVariable' 'io'
```



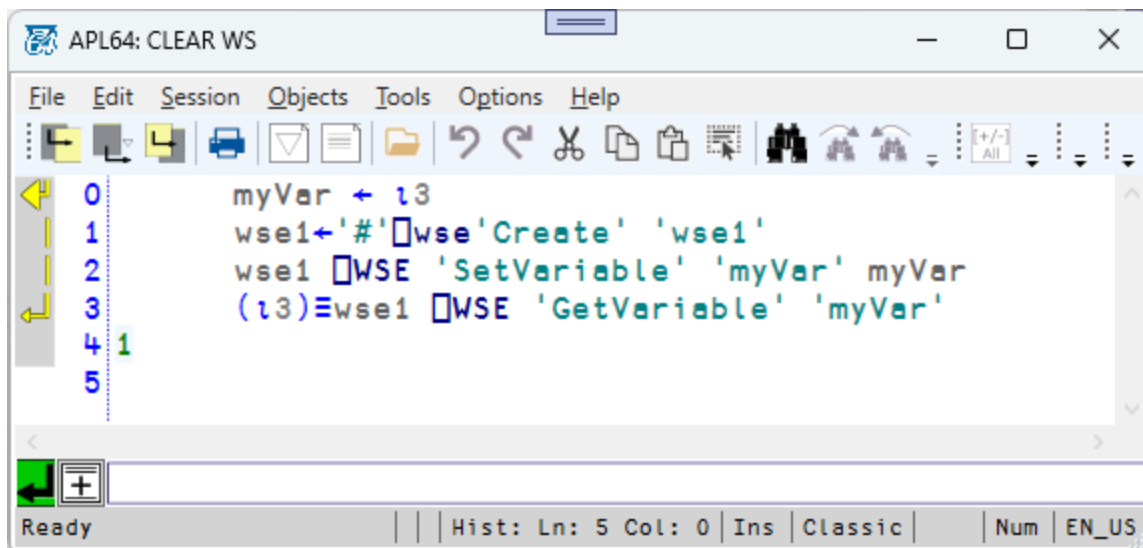
SetVariable

Synonym: SetVar

Set the value of the specified APL64 programmer-defined variable in a ⎕wse instance. If the specified variable does not exist it will be created. The specified APL64 programmer-defined variable name may be a character scalar, character vector or string scalar.

instanceName ⎕WSE 'SetVariable' 'variableName' variableValue

```
myVar ← ⌚3  
wse1←'#'⎕wse'Create' 'wse1'  
wse1 ⎕WSE 'SetVariable' 'myVar' myVar  
⌚3≡wse1 ⎕WSE 'GetVariable' 'myVar'
```



SysCall

Obtain the result, if any, of the interpreter execution of the specified APL64 system function. The `[wse SysCall` action has three versions to execute niladic, monadic or dyadic APL64 system ('quad') functions. When specifying the APL64 system function name, do not include the quad (`[`) prefix. The specified APL64 system function name may be a character scalar, character vector or string scalar.

Niladic

```
res ← instanceName [wse 'SysCall' 'NiladicSystemFn'
```

Refer to the SysCall dyadic action documentation for an example of the SysCall niladic action action.

Monadic

```
res ← instanceName [wse 'SysCall' 'MonadicSystemFn' RightArg
```

Refer to the SysCall dyadic action documentation for an example of the SysCall monadic action action.

Dyadic

The left argument to the APL64 system function is specified after the right argument to that function.

```
res ← instanceName [wse 'SysCall' 'DyadicSystemFn' RightArg LeftArg
```

```
wse1 ← '#'[wse 'Create' 'wse1'
wse1 [wse 'SysCall' 'MKDIR' 'c:\testDyadic\'
⌈↑ [MKDIR is monadic
wse1 [wse 'SysCall' 'NFE' ('Delete' 'c:\testDyadic\myfile.txt')
⌈↑ [NFE is monadic for the Delete action
wse1 [wse 'SysCall' 'XNCREATE' ~1 'c:\testDyadic\myfile.txt'
⌈↑ [XNCREATE is dyadic
wse1 [wse 'SysCall' 'XNNUMS'
⌈↑ [XNNUMS is niladic
wse1 [wse 'SysCall' 'NUNTIE' ~1
```

```

Ⓜ↑ □NUNTIE is monadic
wse1 □wse 'SysCall' 'NFE' ('Delete' 'c:\testDyadic\ test1.txt ')
Ⓜ↑ □NFE is monadic for the Delete action
wse1 □wse 'Syscall' 'NFE' ('Create' 'readwrite' 'readwrite') 'c:\testDyadic\test1.txt'
Ⓜ↑ □NFE is dyadic for the Create action
wse1 □wse 'SysCall' 'NFE' ('Delete' 'c:\testDyadic\myfile.txt')
Ⓜ↑ □NFE is monadic for the Delete action
wse1 □wse 'SysCall' 'NFE' ('Delete' 'c:\testDyadic\ test1.txt ')
Ⓜ↑ □NFE is monadic for the Delete action

```

```

APL64: CLEAR WS
File Edit Session Objects Tools Options Help
1 wse1←'#'□wse 'Create' 'wse1'
2 wse1 □wse 'SysCall' 'MKDIR' 'c:\testDyadic\'
3 A ↑ □MKDIR is monadic
4 wse1 □wse 'SysCall' 'NFE' ('Delete' 'c:\testDyadic\myfile.txt')
5 A ↑ □NFE is monadic for the Delete action
6 wse1 □wse 'SysCall' 'XNCREATE' -1 'c:\testDyadic\myfile.txt'
7 A ↑ □XNCREATE is dyadic
8 wse1 □wse 'SysCall' 'XNNUMS'
9 -1
10 A ↑ □XNNUMS is niladic
11 wse1 □wse 'SysCall' 'NUNTIE' -1
12 A ↑ □NUNTIE is monadic
13 wse1 □wse 'SysCall' 'NFE' ('Delete' 'c:\testDyadic\ test1.txt ')
14 A ↑ □NFE is monadic for the Delete action
15 wse1 □wse 'Syscall' 'NFE' ('Create' 'readwrite' 'readwrite') 'c:\testDyadic\test1.txt'
16 A ↑ □NFE is dyadic for the Create action
17 wse1 □wse 'SysCall' 'NFE' ('Delete' 'c:\testDyadic\myfile.txt')
18 A ↑ □NFE is monadic for the Delete action
19 wse1 □wse 'SysCall' 'NFE' ('Delete' 'c:\testDyadic\ test1.txt ')
20 A ↑ □NFE is monadic for the Delete action

```

SysCommand

Execute an APL64 system command. This □wse action has no result When specifying the APL64 system command name, do not include the right parenthesis ')' prefix.

```

wse1←'#'□wse'Create' 'wse1'
wse1 □wse 'SetVariable' 'X' (15)
ρ□←wse1 □wse 'SysCall' 'idlist' 2
(15)≡wse1 □wse
'GetVariable' 'X'
wse1 □wse 'SysCommand' 'erase X'
ρ□←wse1 □wse 'SysCall' 'idlist' 2

```

```

0      wse1←'#'⊞wse'Create' 'wse1'
1      wse1 ⊞wse 'SetVariable' 'X' (15)
2      p⊞wse1 ⊞wse 'SysCall' 'idlist' 2
3      (15)≡wse1 ⊞wse 'GetVariable' 'X'
4      wse1 ⊞wse 'SysCommand' 'erase X'
5      p⊞wse1 ⊞wse 'SysCall' 'idlist' 2
6      X
7      1 1
8      1
9      0 0
10

```

Ready | Hist: Ln: 10 Col: 6 | Ins | Classic | Num | EN_US

Variable

Get or set the value of a specified APL64 programmer-defined variable in a `⊞wse` instance. If the specified variable does not exist it will be created. The specified APL64 programmer-defined variable name may be a character scalar, character vector or string scalar. The `⊞wse Variable` action combines the `⊞wse GetVariable` and `⊞wse SetVariable` actions.

```

⊞wseSelf←'#'⊞wse'Create' 'wse1'
⊞wse 'Variable' 'X' (13)
(13)≡⊞←⊞wse 'Variable' 'X'

```

```

0      ⊞wseSelf←'#'⊞wse'Create' 'wse1'
1      ⊞wse 'Variable' 'X' (13)
2      (13)≡⊞←⊞wse 'Variable' 'X'
3      1 2 3
4      1
5

```

Ready | Hist: Ln: 5 Col: 6 | Ins | Classic | Num | EN_US

WSE Action Restrictions

 WSE asynchronous actions may be initiated any time a synchronous action is not running.

Object Actions	Permitted while Asynchronous Action Is Running	Permitted when Synchronous Action is Running
AsyncCancelAll	Y	N
AsyncHandler	N, if changing the value Y, if obtaining the value	N
AsyncAnyTaskRunning	Y	N
AsyncTasksRunning	Y	N
Clear	Y	N
Count	Y	N
Create	Y	N
Instances	Y	N
Help (?)	Y	N
New	Y	N

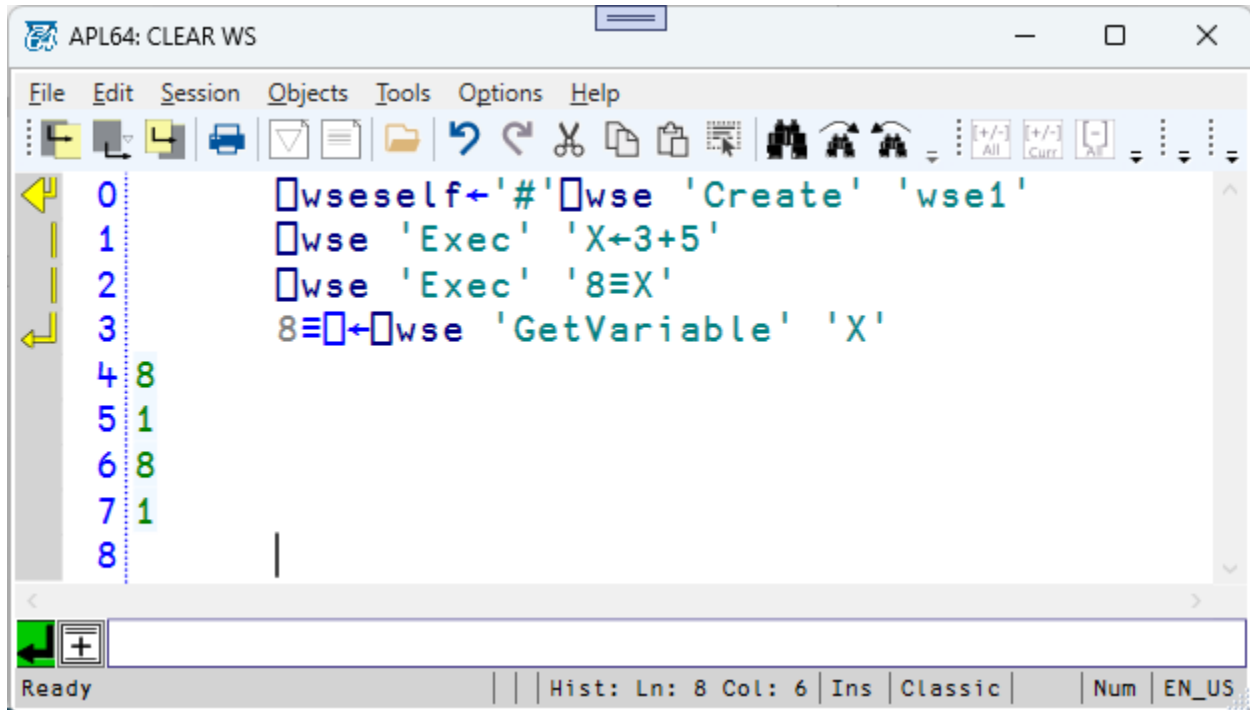
Instance Actions	Permitted while Asynchronous Action Is Running	Permitted when Synchronous Action is Running
AsyncCall	N, if targeting a  WSE instance already running an AsyncCall, otherwise Y	N
AsyncCancel	Y	N
AsyncStatus	Y	N
AsyncTaskIsRunning	Y	N
Call	N	N
Delete	Y	N
Exec	N	N
GetSysVariable	Y	N
GetVariable	Y	N
LoadWs	N	N
SysCall	N	N
SetSysVariable	N	N
SetVariable	N	N
SysCommand	N	N
Variable	N, if modifying a variable value Y, if querying a variable value	N

WseSelf System Variable

The  WseSelf system variable may be set to a character scalar, character vector or string scalar. The return value of the  WseSelf system variable is a character vector.

When the `⎕WseSelf` system variable has a programmer-defined value, the left argument to the `⎕Wse` system function is optional when a `⎕Wse` instance action is to be performed.

```
⎕wseSelf←'#'⎕wse 'Create' 'wse1'
⎕wse 'Exec' 'X←3+5'
⎕wse 'Exec' '8≡X'
8≡⎕←⎕wse 'GetVariable' 'X'
```



`⎕WseInfo` System Variable

The `⎕wseinfo` system variable provides information about a `⎕wse` workspace engine instance.

The first element of the `⎕wseinfo` system variable is a Boolean scalar which indicates if the APL64 instance is running within a `⎕wse` workspace engine instance.

In future versions of APL64, additional elements may be added to the result of the `⎕wseinfo` system variable.

Refer to the menu **Help | APL Language | Using `⎕WSE`** for additional information on this system variable.

```
⎕wseinfo ⓈAPL64 Developer version instance
wse1←'#'⎕wse'Create' 'wse1'
```

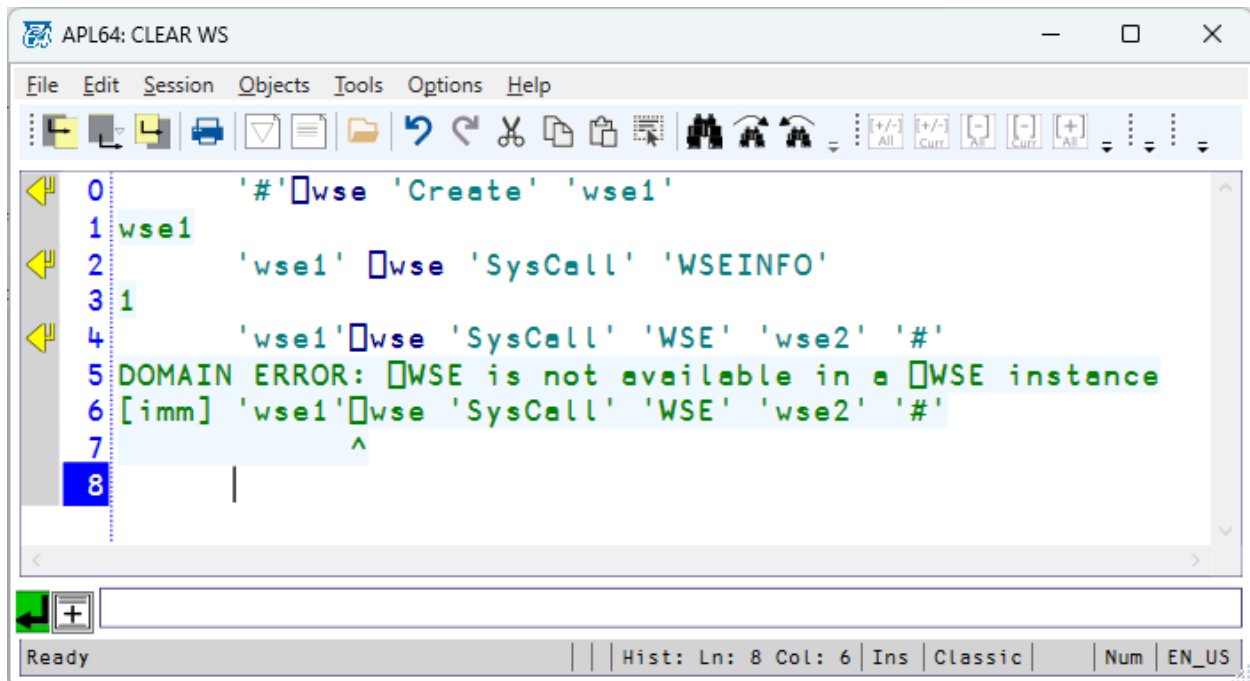
```
wse1␣wse 'Syscall' 'wseinfo' ␣wse1␣wse instance
␣wseinfo ␣APL64 Developer version instance
```

```
APL64: CLEAR WS
File Edit Session Objects Tools Options Help
␣wseinfo ␣APL64 Developer version instance
wse1←'#'␣wse'Create' 'wse1'
wse1␣wse 'Syscall' 'wseinfo' ␣wse1␣wse instance
␣wseinfo ␣APL64 Developer version instance
0
1
2
0
```

␣WSE Comparison with Legacy Workspace Engine

	␣WSE	APL+Win
Cross-platform	Yes	No
ActiveX dependent	No	Yes
Use all available workstation memory	Yes	No: Each instance limited to ~3Gb of memory
In-process instances	Yes	Yes: Client plus instances memory limited to ~3Gb of memory
Asynchronous Function Execution	Yes, use the ␣WSE AsyncCall, AsyncWait, AsyncFoFi actions	No
Instantiate by non-APL64 code	Yes, when used within an APL64 Cross-platform Component	Only by ActiveX-aware applications
Use in a Web Service	Yes, when used within an APL64 Cross-platform Component	Only by ActiveX-aware web servers
Unicode based	Yes	Limited to UExec method for Unicode-aware client
Out-of-process instances	N/A: Use APL64 Cross-platform Component (CPC runtime) for out-of-process instances	Yes, with security caveats
Visible property	No: Designed to be cross-platform	Yes
Instances use ␣WSE	No	Yes
Server =>Client Communication	Yes: ␣Progress	Limited: Notify SysNotify
Client=>Server Communication	Yes: ␣WSE 'CallNow'	Limited: Notify SysNotify

A `WSE` instance cannot use `WSE`:



```
0      '# WSE 'Create' 'wse1'
1 wse1
2      'wse1' WSE 'SysCall' 'WSEINFO'
3 1
4      'wse1' WSE 'SysCall' 'WSE' 'wse2' '#'
5 DOMAIN ERROR: WSE is not available in a WSE instance
6 [imm] 'wse1' WSE 'SysCall' 'WSE' 'wse2' '#'
7      ^
8
```

`Progress`

`Progress` may be used to send information from a `WSE` APL server instance to the `WSE` APL client instance which created that `WSE` APL server instance when the `WSE AsyncCall` action is underway.

Syntax: `Progress AplVar`

The right argument of `Progress` is any APL64 variable.

`Progress` has no result. `Progress` has no effect if it is used by the `WSE` APL client instance. `Progress` has no effect if it is used by a `WSE` APL server instance if the `WSE AsyncCall` action is not underway.

The right argument of the `Progress` system function is an APL64 variable value provided by the APL64 programmer. The `Progress` system function has no left argument.

When the `Progress` system function is used in a `WSE` server APL instance, the `Progress` information provided by the server APL instance can be accessed by the client APL instance by specifying a `WSE AsyncHandler` function.

The `WSE AsyncHandler` function must exist in the client APL instance, and the `WSE AsyncHandler` function name must be specified, prior to the use of the `Progress` system function by the server APL instance. Use the [WSE AsyncHandler action](#) to specify the name of the handler function.

Using `WSE Progress` when an `WSE AsyncCall` action is underway

Refer to the [WSE AsyncCall action example](#) to see how the `WSE Progress` system function may be used while a `WSE` asynchronous operation is underway.

`WseMc`

`WseMc` (`Message Client`) may be used to send information from a `WSE APL` server instance to the `WSE APL` client instance which created that `WSE APL` server instance.

`WseMc` has no result. `WseMc` has no effect if it is used by the `WSE APL` client instance.

The right argument of the `WseMc` system function is an `APL64` variable value provided by the `APL64` programmer. The `WseMc` system function has no left argument.

When the `WseMc` system function is used in a `WSE` server `APL` instance, an event is fired in the scope of the `WSE APL` client instance. The `WSE APL` client instance must subscribe to this event using the `WSE AsyncHandler` action.

The `WSE AsyncHandler` function must exist in the `WSE APL` client instance, and the `WSE AsyncHandler` function name must be specified, prior to the use of the `WseMc` system function by the `WSE` server `APL` instance. Use the [WSE AsyncHandler action](#) to specify the name of the handler function.

When the `WseMc` system function is used, the value of the 'context' argument of the `WSE AsyncHandler` function is "".

Use `WseMc` to send a message from an `APL` Server to the `APL` Client

```
⎕def 'MyAsyncEh arg' 'arg' ⎕ Define function to handle the ⎕WseMc event
'#'⎕WSE 'AsyncHandler' 'MyAsyncEh' ⎕ Specify the event handler function
'#'⎕WSE 'Create' 'wse1' ⎕ Create a ⎕WSE APL server instance
'wse1'⎕WSE 'Exec' ""⎕WseMc ('abcd' (2 3pt6))" ⎕ APL server sends info to APL client
⎕↑ Server info received by Client
```

```

0  ⍋def 'MyAsyncEh arg' 'arg'  ⍝ Define function to handle the ⍋WseMc event
1  MyAsyncEh
2      '#⍋WSE 'AsyncHandler' 'MyAsyncEh' ⍝ Specify the event handler function
3
4      '#⍋WSE 'Create' 'wse1' ⍝ Create a ⍋WSE APL server instance
5  wse1
6      'wse1⍋WSE 'Exec' "⍋WseMc ('abcd' (2 3⍋16))" ⍝ APL server sends info to APL client
7      >[⍋WSE:wse1;WseMc] MyAsyncEh
8  wse1 WseMc abcd 1 2 3
9              4 5 6
10
11  ⍝ Server info received by Client

```

In this ‘artificial’ example, the statement: 'wse1'⍋WSE 'Exec' "⍋WseMc ('abcd' (2 3⍋16))" is causing the client APL instance to send a message to itself routing it through the server APL instance. In a production environment, the server APL instance will use the ⍋WseMc during its processing of client requests.

Learn More

To learn more about ⍋WSE contact sales@apl2000.com to set up customized training or consulting. For APL64 subscribers, contact support@apl2000.com for technical assistance.