APL64: USING ⎕WSE

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.

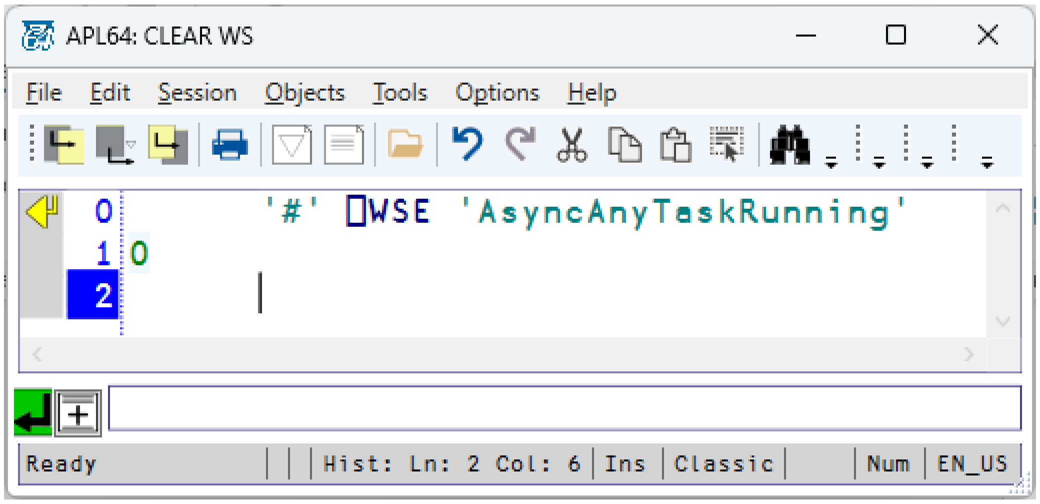
All Action examples use a new instance of the APL64 Developer version. Action examples may use multi-line APL64 executable statements.

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'
```


APL64: CLEAR WS
File Edit Session Objects Tools Options Help
Ready | Hist: Ln: 2 Col: 6 | Ins | Classic | Num | EN_US

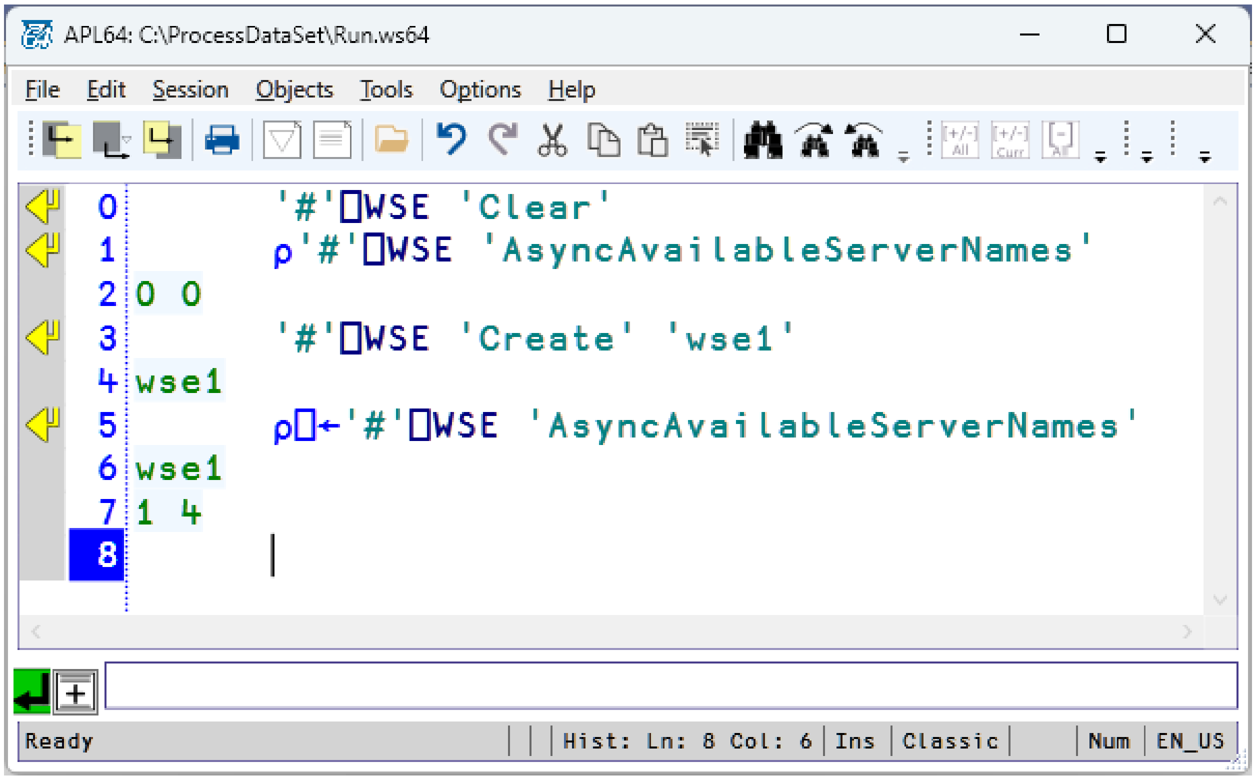
```
      '#' ⎕WSE 'AsyncAnyTaskRunning'
0
```

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'
```


APL64: C:\ProcessDataSet\Run.ws64
File Edit Session Objects Tools Options Help
Ready
Hist: Ln: 8 Col: 6
Ins
Classic
Num
EN_US

```
      '#'⎕WSE 'Clear'
      ρ'#'⎕WSE 'AsyncAvailableServerNames'
0 0
      '#'⎕WSE 'Create' 'wse1'
wse1
      ρ⎕←'#'⎕WSE 'AsyncAvailableServerNames'
wse1
1 4
```

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.

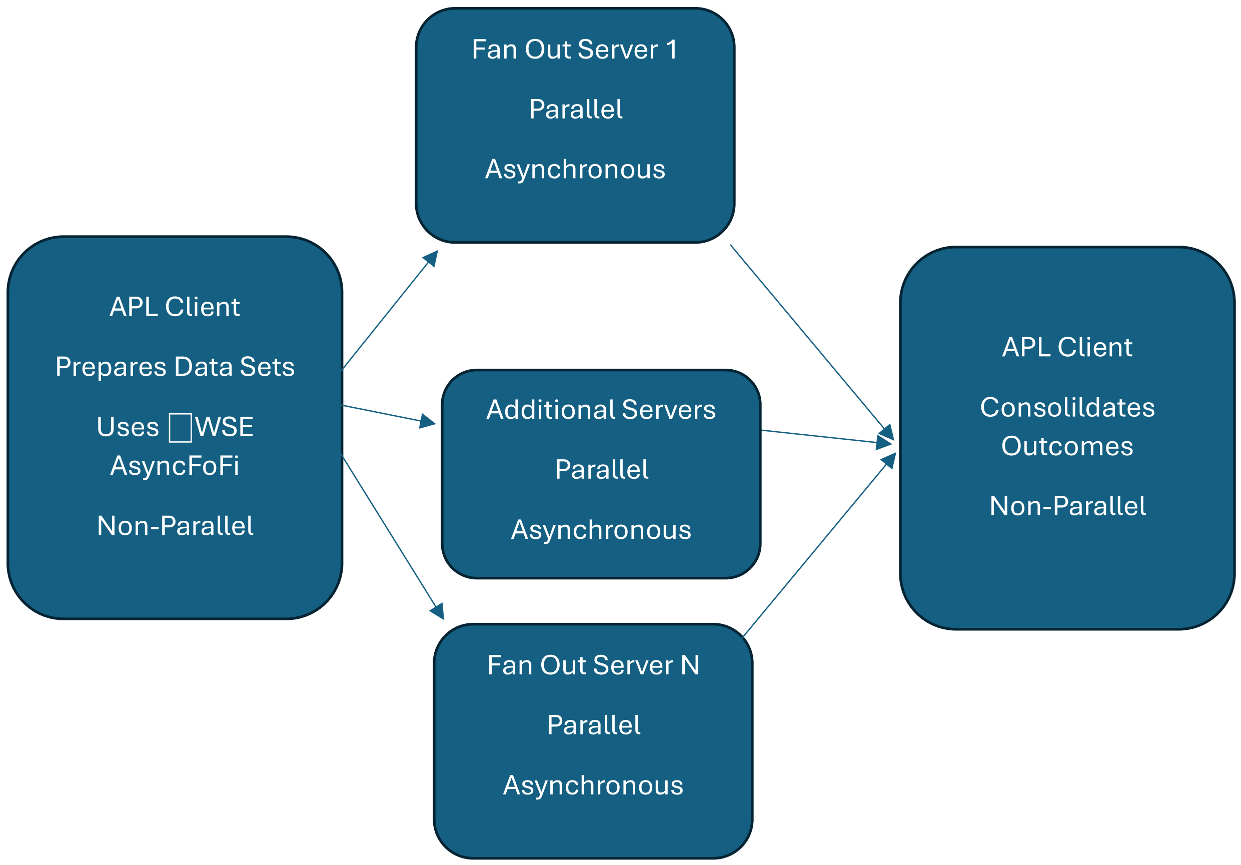
'#' ⎕WSE 'AsyncHandler'

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.


APL Client
Prepares Data Sets
Uses ⎕WSE AsyncFoFi
Non-Parallel
Fan Out Server 1
Parallel
Asynchronous
Additional Servers
Parallel
Asynchronous
Fan Out Server N
Parallel
Asynchronous
APL Client
Consolildates Outcomes
Non-Parallel

⎕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

⎕WSEAsync 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 Rquest 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.

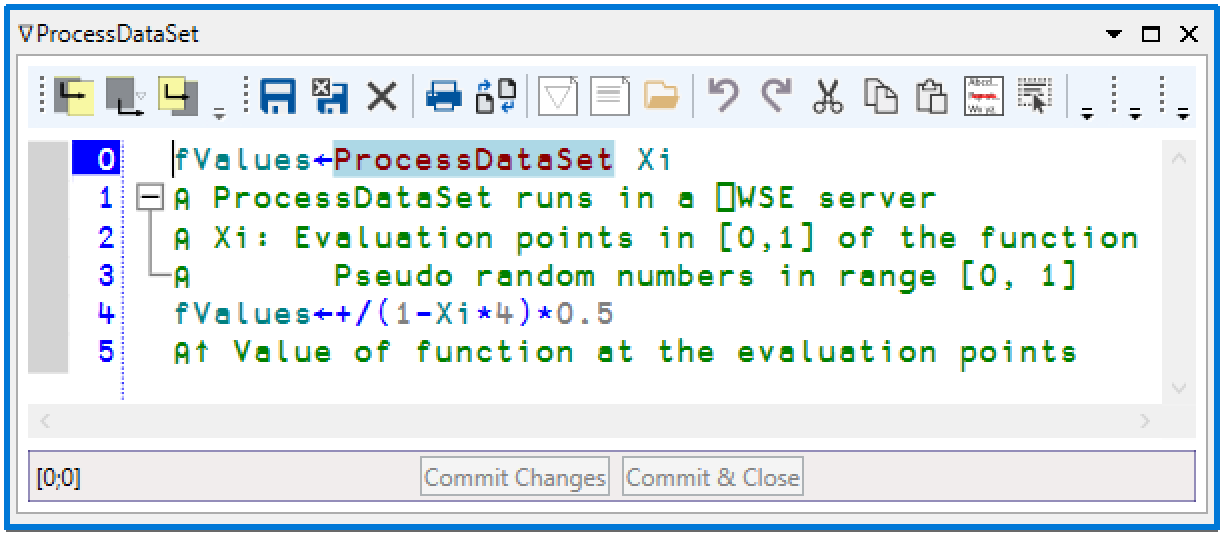
⎕WSE 'AsyncFoFi' example

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.

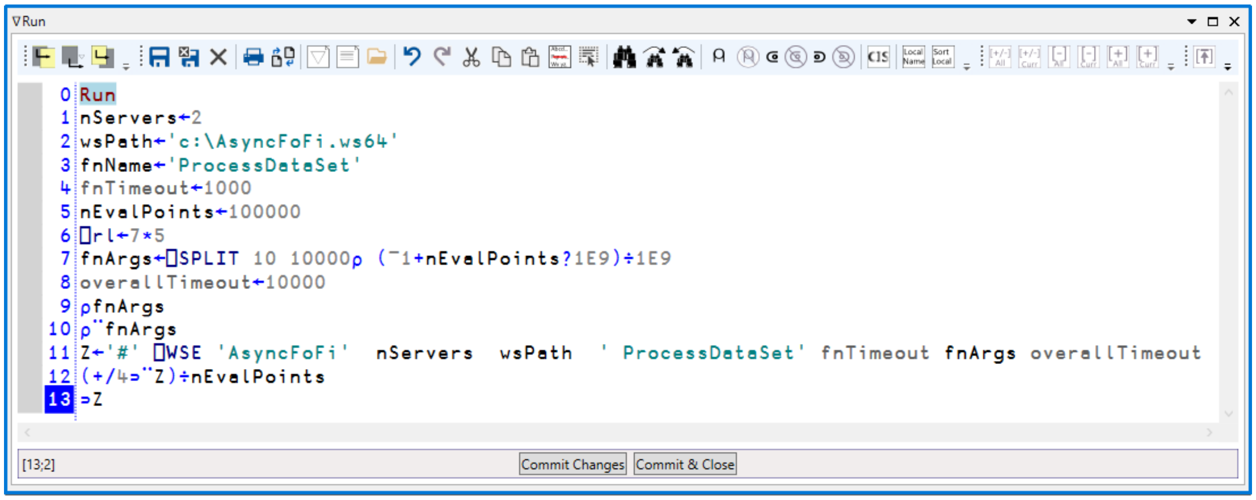
```
fValues ←ProcessDataSet Xi
⍝ ProcessDataSet runs in a ⎕WSE server
⍝ Xi: Evaluation points in [0,1] of the function
⍝      Pseudo random numbers in range [0, 1]
fValues←+/(1-Xi*4)*0.5
⍝↑ Value of function at the evaluation points
```



```
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
```

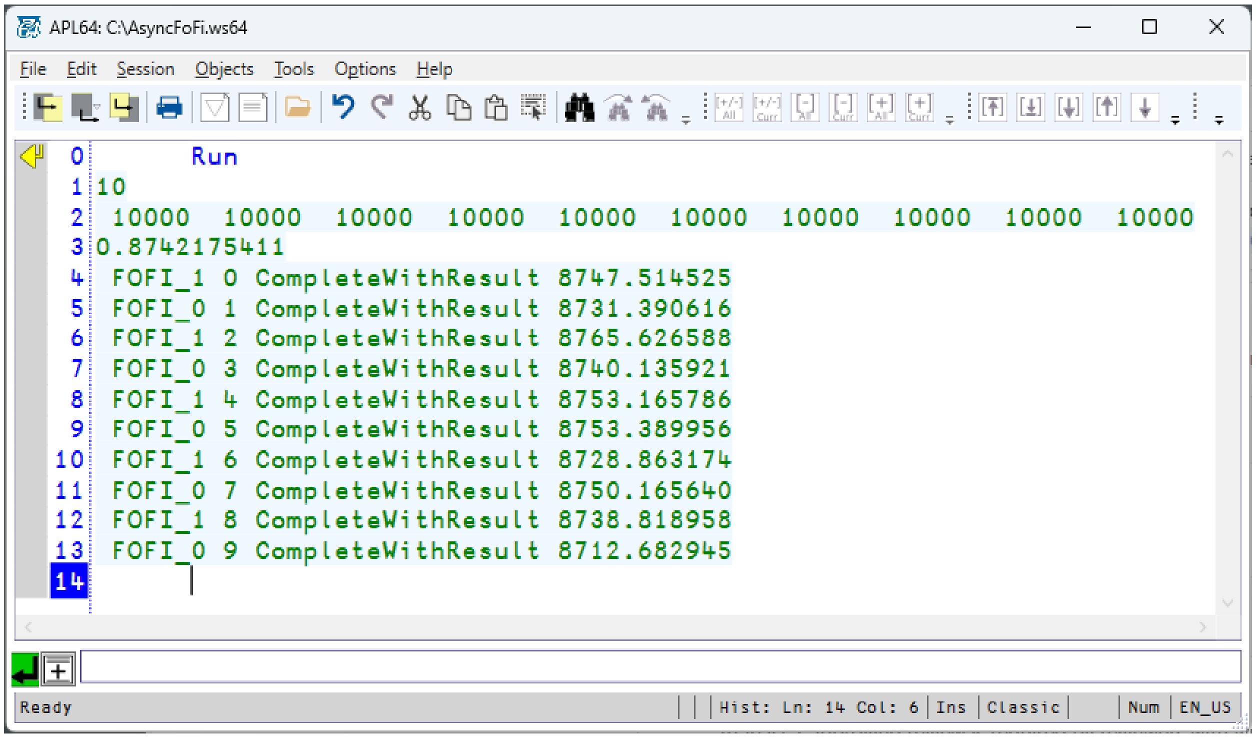
Create the Run function:

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



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

Run the example



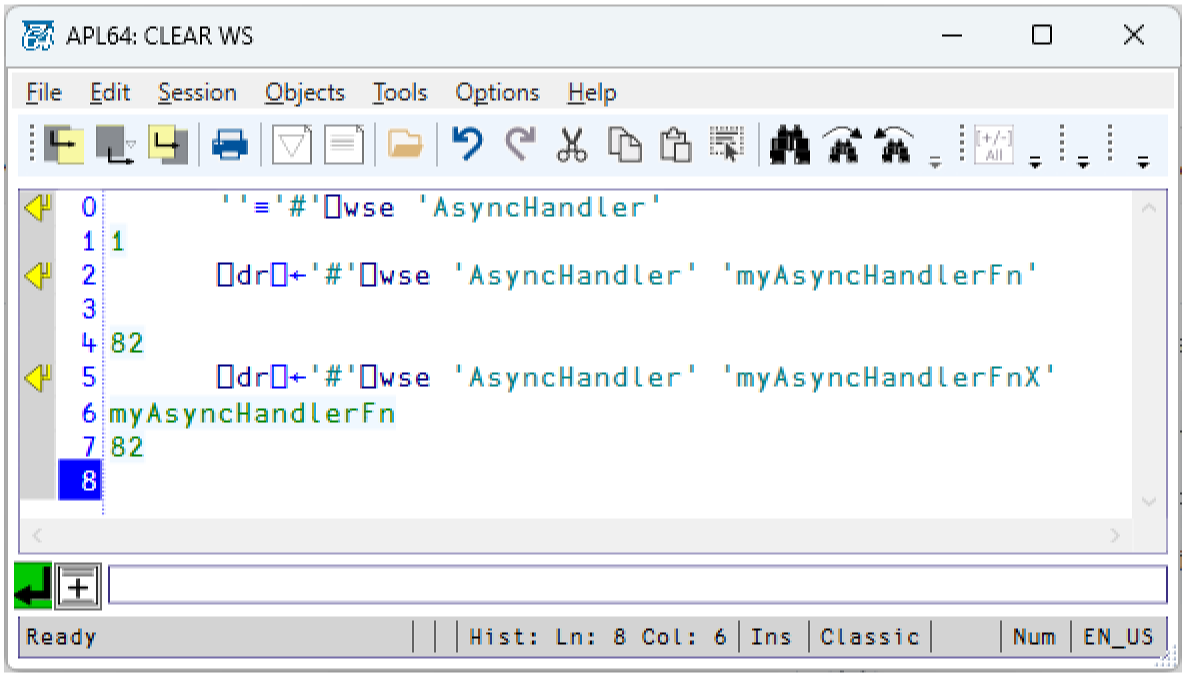
```
      Run
10
 10000  10000  10000  10000  10000  10000  10000  10000  10000  10000
0.8742175411
 FOFI_1 0 CompleteWithResult 8747.514525
 FOFI_0 1 CompleteWithResult 8731.390616
 FOFI_1 2 CompleteWithResult 8765.626588
 FOFI_0 3 CompleteWithResult 8740.135921
 FOFI_1 4 CompleteWithResult 8753.165786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

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.



```
      ''≡'#'⎕wse 'AsyncHandler'
1
      ⎕dr⎕←'#'⎕wse 'AsyncHandler' 'myAsyncHandlerFn'

82
      ⎕dr⎕←'#'⎕wse 'AsyncHandler' 'myAsyncHandlerFnX'
myAsyncHandlerFn
82
```

```
Ready | | Hist: Ln: 8 Col: 6 | Ins | Classic | | Num | EN_US
```

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.

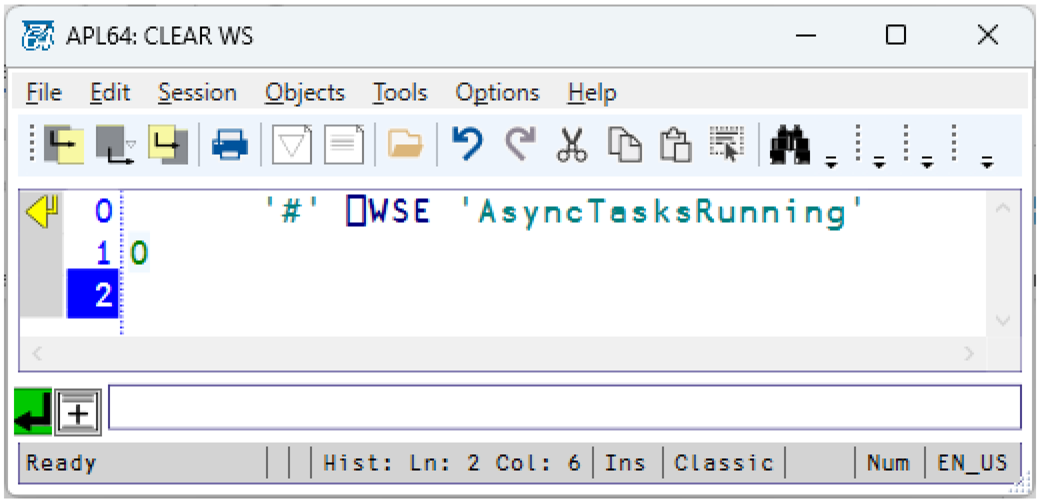
APL64 Result Variable is the APL64 programmer-defined variable which is returned by the APL64 programmer-defined function run in the ⎕WSE server APL instance.

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'


APL64: CLEAR WS
File Edit Session Objects Tools Options Help

```
      '#' ⎕WSE 'AsyncTasksRunning'
0
```

Ready | Hist: Ln: 2 Col: 6 | Ins | Classic | Num | EN_US

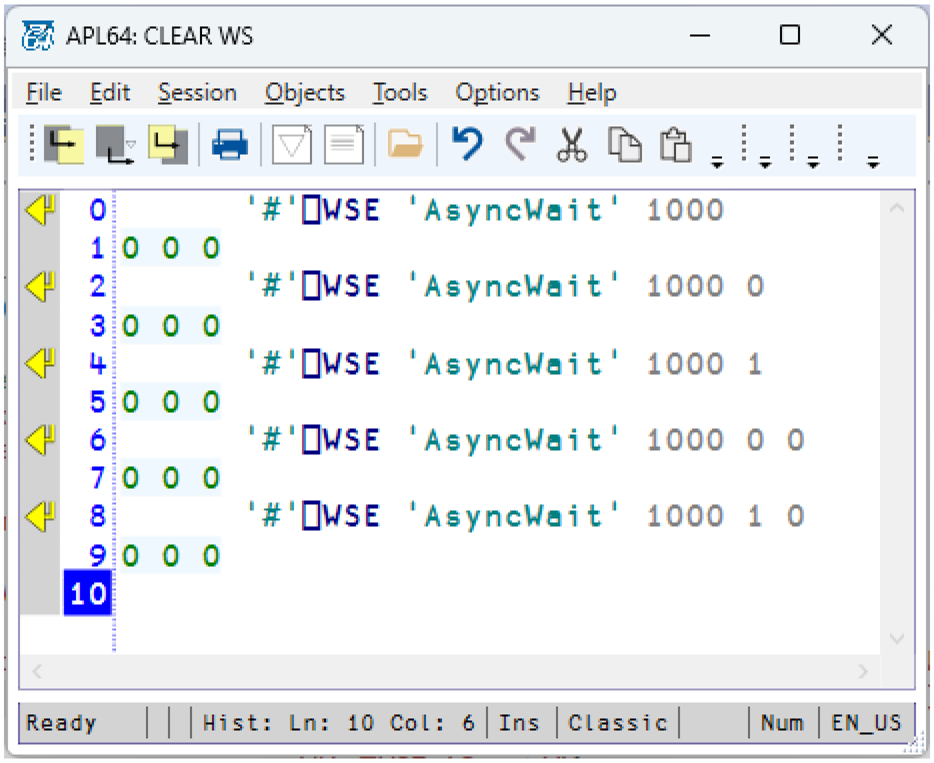
AsyncWait

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
```



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

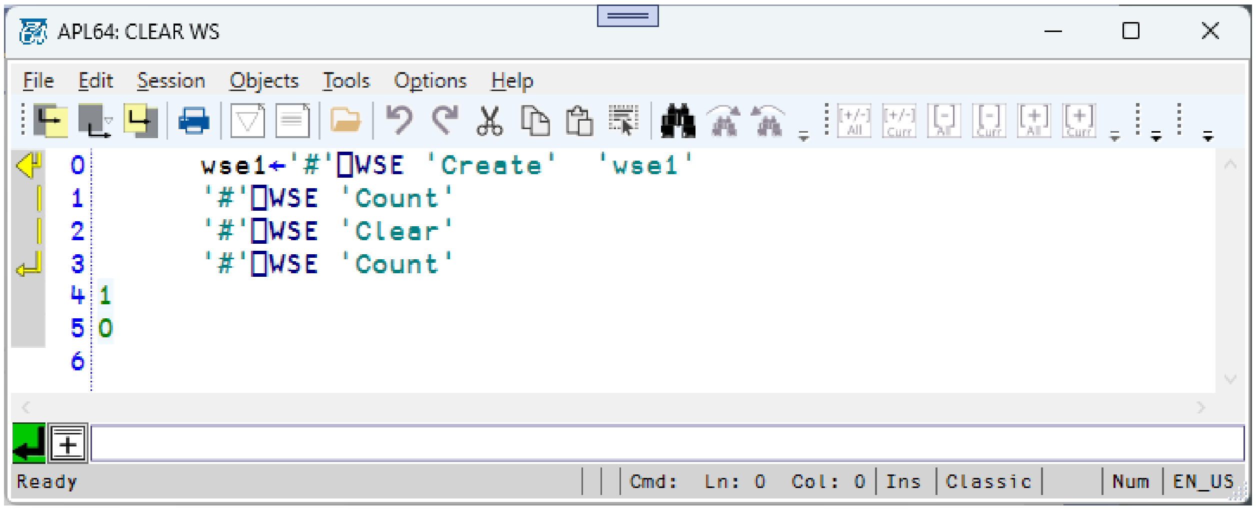
```
wse1←'#'⎕WSE 'Create'  'wse1'
'#'⎕WSE 'Count'
'#'⎕WSE 'Clear'
'#'⎕WSE 'Count'
'#'⎕WSE 'Clear'
```

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'
```



```
      wse1←'#'⎕WSE 'Create'  'wse1'
      '#'⎕WSE 'Count'
      '#'⎕WSE 'Clear'
      '#'⎕WSE 'Count'
1
0
```

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

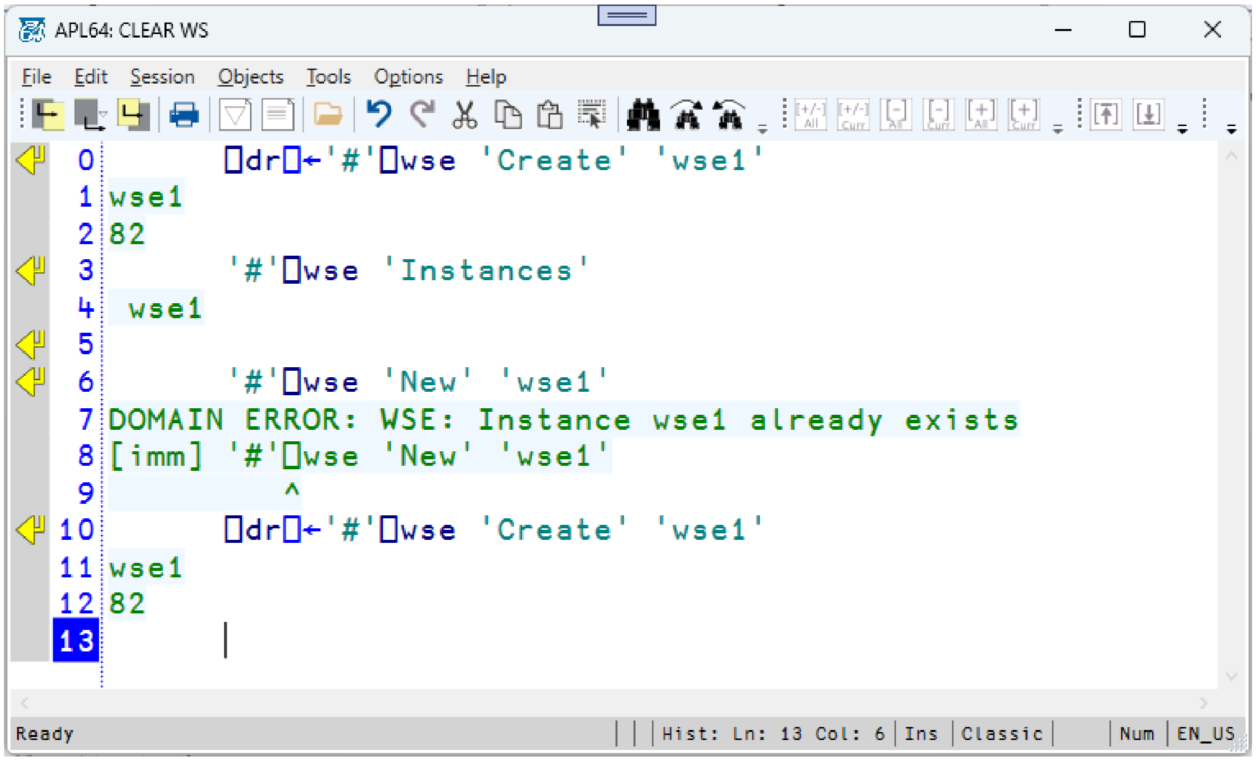
Create

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'
```

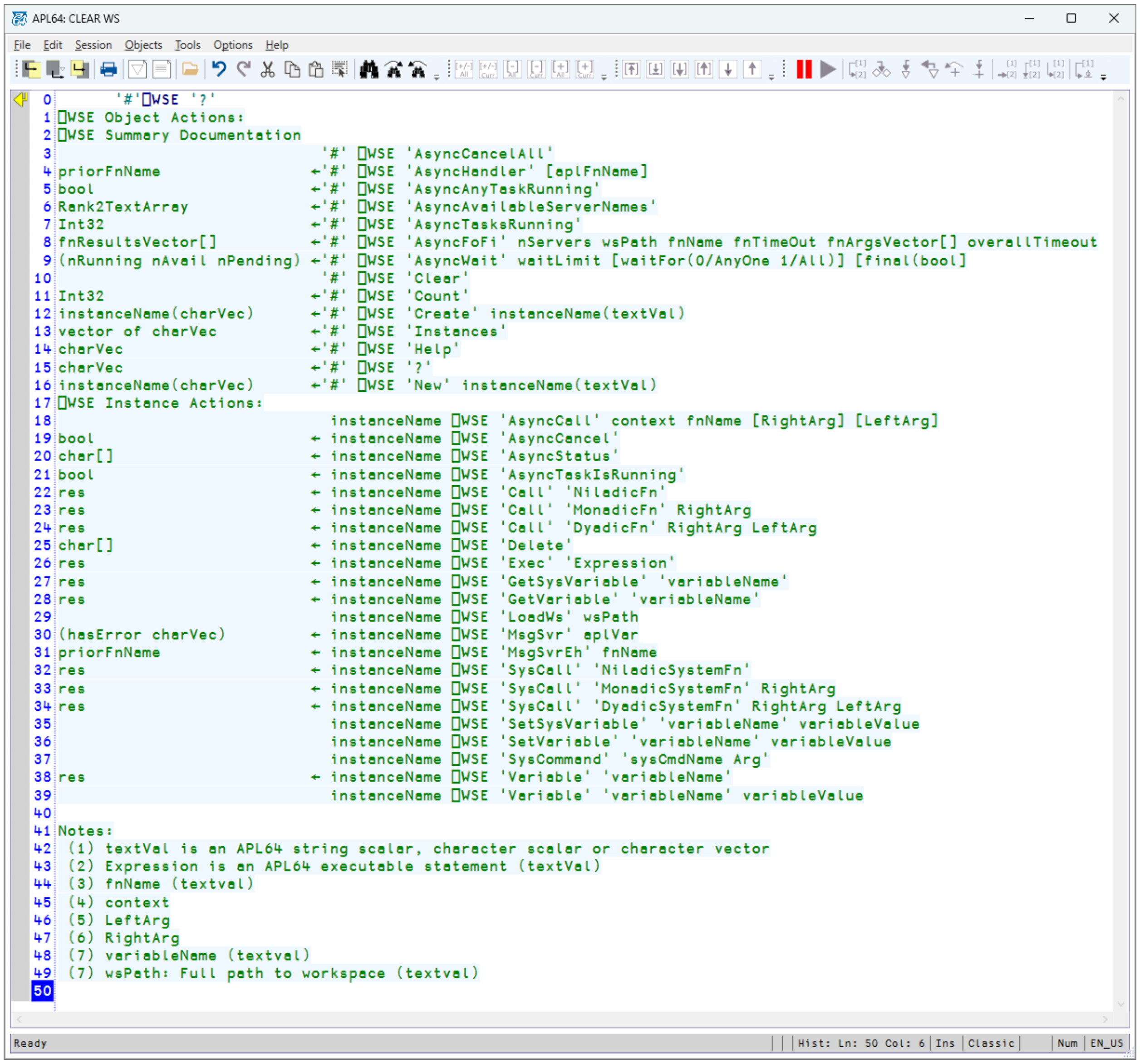


```
      ⎕dr⎕←'#'⎕wse 'Create' 'wse1'
wse1
82
      '#'⎕wse 'Instances'
 wse1

      '#'⎕wse 'New' 'wse1'
DOMAIN ERROR: WSE: Instance wse1 already exists
[imm] '#'⎕wse 'New' 'wse1'
           ^
      ⎕dr⎕←'#'⎕wse 'Create' 'wse1'
wse1
82
```

Help (?)

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

```
'#'⎕WSE '?'
```

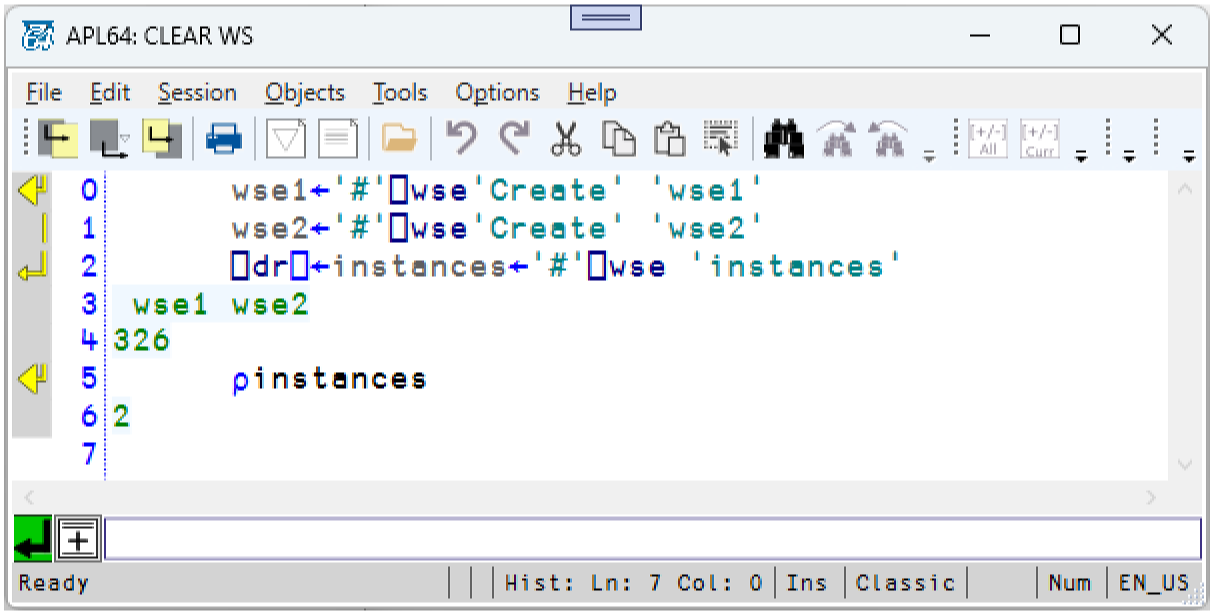


```
      '#'⎕WSE '?'
⎕WSE Object Actions:
⎕WSE Summary Documentation
                            '#' ⎕WSE 'AsyncCancelAll'
priorFnName                ←'#' ⎕WSE 'AsyncHandler' [aplFnName]
bool                       ←'#' ⎕WSE 'AsyncAnyTaskRunning'
Rank2TextArray             ←'#' ⎕WSE 'AsyncAvailableServerNames'
Int32                      ←'#' ⎕WSE 'AsyncTasksRunning'
fnResultsVector[]          ←'#' ⎕WSE 'AsyncFoFi' nServers wsPath fnName fnTimeOut fnArgsVector[] overallTimeout
(nRunning nAvail nPending) ←'#' ⎕WSE 'AsyncWait' waitLimit [waitFor(0/AnyOne 1/All)] [final(bool]
                            '#' ⎕WSE 'Clear'
Int32                      ←'#' ⎕WSE 'Count'
instanceName(charVec)      ←'#' ⎕WSE 'Create' instanceName(textVal)
vector of charVec          ←'#' ⎕WSE 'Instances'
charVec                    ←'#' ⎕WSE 'Help'
charVec                    ←'#' ⎕WSE '?'
instanceName(charVec)      ←'#' ⎕WSE 'New' instanceName(textVal)
⎕WSE Instance Actions:
                             instanceName ⎕WSE 'AsyncCall' context fnName [RightArg] [LeftArg]
bool                       ← instanceName ⎕WSE 'AsyncCancel'
char[]                     ← instanceName ⎕WSE 'AsyncStatus'
bool                       ← instanceName ⎕WSE 'AsyncTaskIsRunning'
res                        ← instanceName ⎕WSE 'Call' 'NiladicFn'
res                        ← instanceName ⎕WSE 'Call' 'MonadicFn' RightArg
res                        ← instanceName ⎕WSE 'Call' 'DyadicFn' RightArg LeftArg
char[]                     ← instanceName ⎕WSE 'Delete'
res                        ← instanceName ⎕WSE 'Exec' 'Expression'
res                        ← instanceName ⎕WSE 'GetSysVariable' 'variableName'
res                        ← instanceName ⎕WSE 'GetVariable' 'variableName'
                             instanceName ⎕WSE 'LoadWs' wsPath
(hasError charVec)         ← instanceName ⎕WSE 'MsgSvr' aplVar
priorFnName                ← instanceName ⎕WSE 'MsgSvrEh' fnName
res                        ← instanceName ⎕WSE 'SysCall' 'NiladicSystemFn'
res                        ← instanceName ⎕WSE 'SysCall' 'MonadicSystemFn' RightArg
res                        ← instanceName ⎕WSE 'SysCall' 'DyadicSystemFn' RightArg LeftArg
                             instanceName ⎕WSE 'SetSysVariable' 'variableName' variableValue
                             instanceName ⎕WSE 'SetVariable' 'variableName' variableValue
                             instanceName ⎕WSE 'SysCommand' 'sysCmdName Arg'
res                        ← instanceName ⎕WSE 'Variable' 'variableName'
                             instanceName ⎕WSE 'Variable' 'variableName' variableValue

Notes:
 (1) textVal is an APL64 string scalar, character scalar or character vector
 (2) Expression is an APL64 executable statement (textVal)
 (3) fnName (textval)
 (4) context
 (5) LeftArg
 (6) RightArg
 (7) variableName (textval)
 (7) wsPath: Full path to workspace (textval)
```

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
```


APL64: CLEAR WS
File Edit Session Objects Tools Options Help

```
      wse1←'#'⎕wse'Create' 'wse1'
      wse2←'#'⎕wse'Create' 'wse2'
      ⎕dr⎕←instances←'#'⎕wse 'instances'
 wse1 wse2
326
      ⍴instances
2
```

Ready | Hist: Ln: 7 Col: 0 | Ins | Classic | Num | EN_US

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

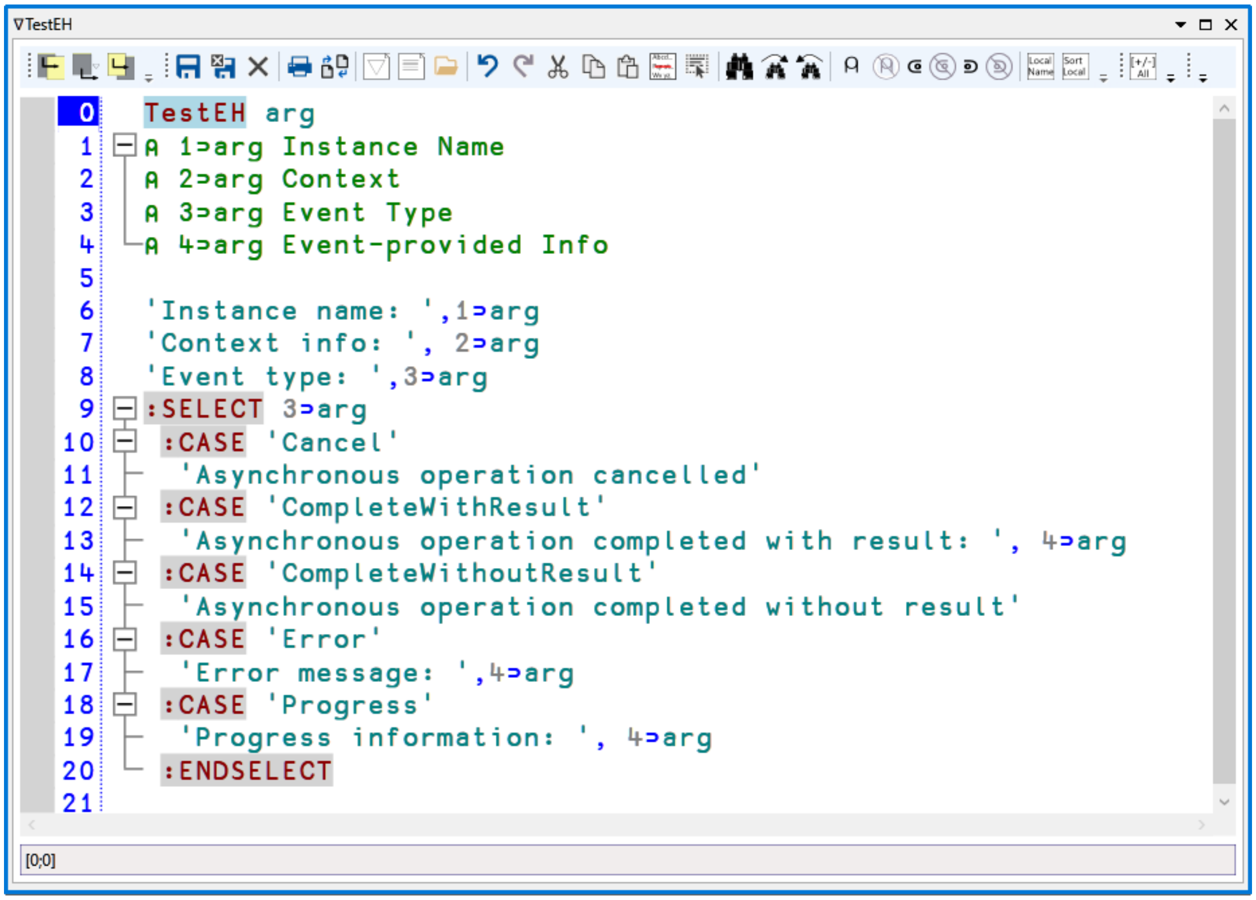
Example: AsyncCall: Using One ⎕WSE Server

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
```

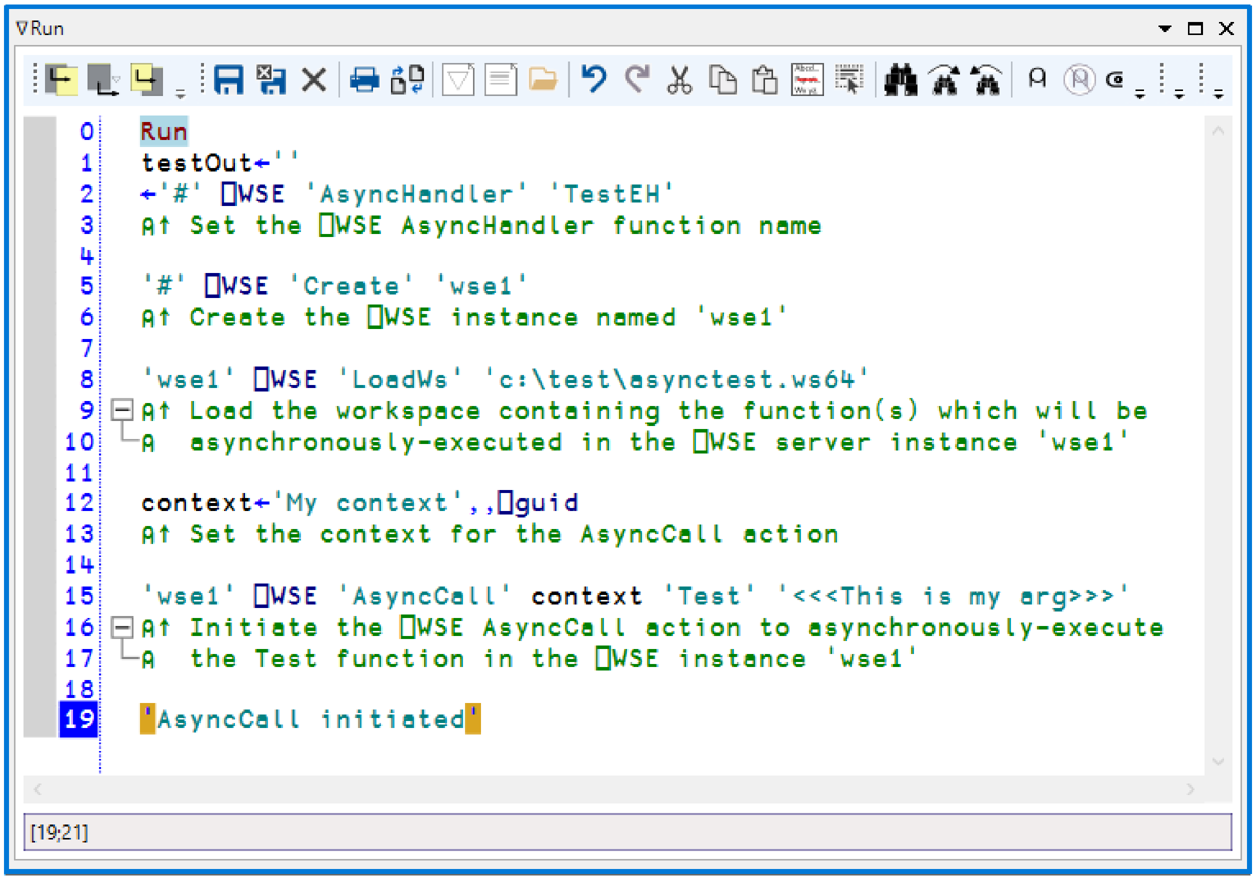


```
∇TestEH
 TestEH arg
⍝ 1⊃arg Instance Name
⍝ 2⊃arg Context
⍝ 3⊃arg Event Type
⍝ 4⊃arg Event-provided Info

'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

[0;0]
```

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'
```



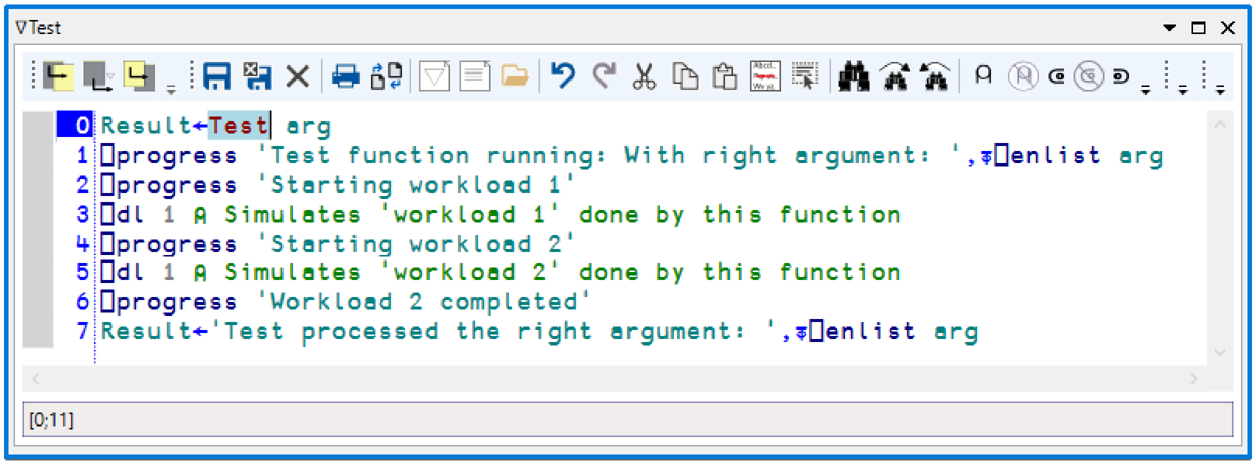
```
Run
testOut←''
←'#' ⎕WSE 'AsyncHandler' 'TestEH'
⍝↑ Set the ⎕WSE AsyncHandler function name

'#' ⎕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'
```

[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
```

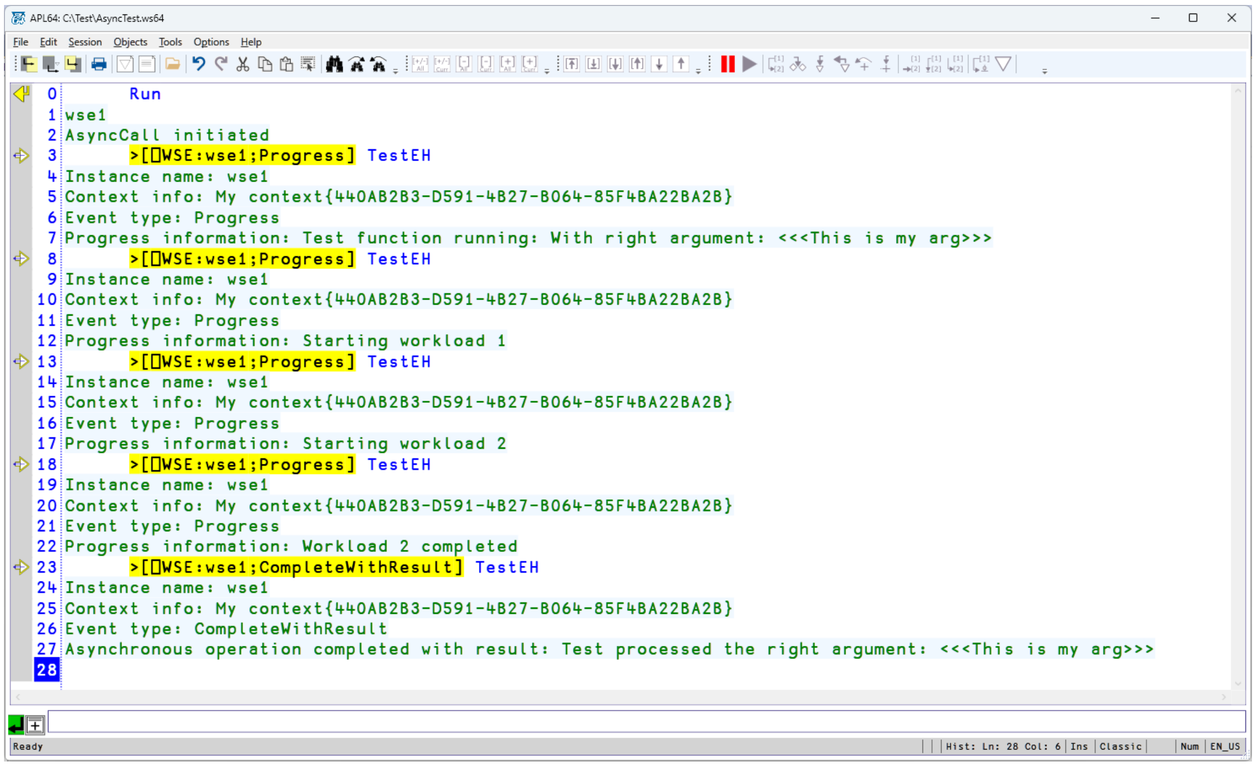


```
Result←Test arg
⎕progress 'Test function running: With right argument: ',⍕⎕enlist arg
⎕progress 'Starting workload 1'
⎕dl 1 ⍝ Simulates 'workload 1' done by this function
⎕progress 'Starting workload 2'
⎕dl 1 ⍝ Simulates 'workload 2' done by this function
⎕progress 'Workload 2 completed'
Result←'Test processed the right argument: ',⍕⎕enlist arg
```

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.


APL64: C:\Test\AsyncTest.ws64
File Edit Session Objects Tools Options Help
Ready
Hist: Ln: 28 Col: 6 | Ins | Classic | Num | EN_US

```
      Run
wse1
AsyncCall initiated
      >[⎕WSE:wse1;Progress] TestEH
Instance name: wse1
Context info: My context{440AB2B3-D591-4B27-B064-85F4BA22BA2B}
Event type: Progress
Progress information: Test function running: With right argument: <<<This is my arg>>>
      >[⎕WSE:wse1;Progress] TestEH
Instance name: wse1
Context info: My context{440AB2B3-D591-4B27-B064-85F4BA22BA2B}
Event type: Progress
Progress information: Starting workload 1
      >[⎕WSE:wse1;Progress] TestEH
Instance name: wse1
Context info: My context{440AB2B3-D591-4B27-B064-85F4BA22BA2B}
Event type: Progress
Progress information: Starting workload 2
      >[⎕WSE:wse1;Progress] TestEH
Instance name: wse1
Context info: My context{440AB2B3-D591-4B27-B064-85F4BA22BA2B}
Event type: Progress
Progress information: Workload 2 completed
      >[⎕WSE:wse1;CompleteWithResult] TestEH
Instance name: wse1
Context info: My context{440AB2B3-D591-4B27-B064-85F4BA22BA2B}
Event type: CompleteWithResult
Asynchronous operation completed with result: Test processed the right argument: <<<This is my arg>>>
```

Example: AsyncCall: Using Multiple ⎕WSE Servers

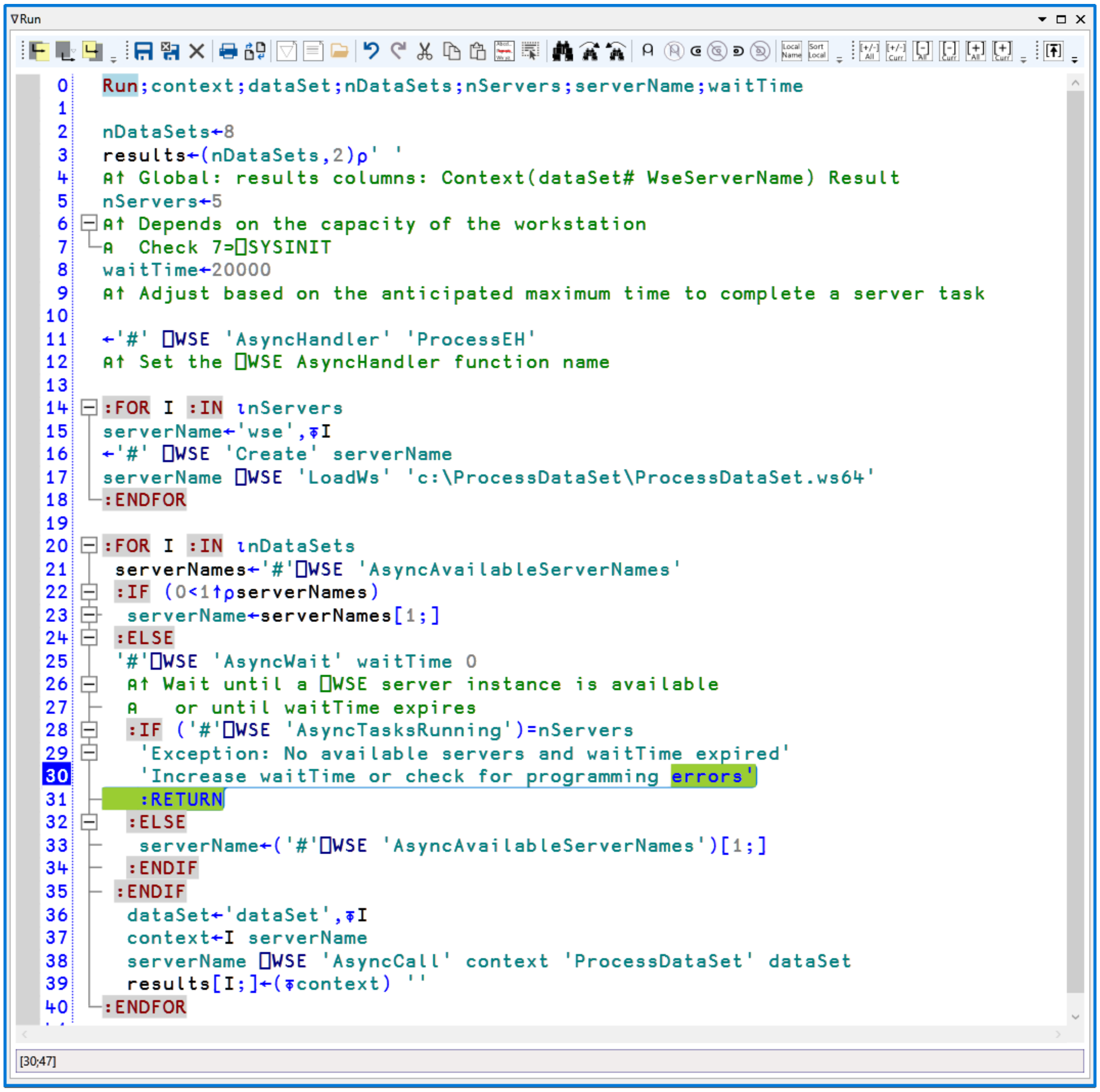
Sometimes an application needs to process multiple data sets of the same format, but with each data set containing different values. If the processing of a data set is an independent calculation, the APL64 client can create multiple ⎕WSE servers, and each APL64 server can process a data set.

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)⍴' '
⍝↑ 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 ⍳nDataSets
 serverNames←'#'⎕WSE 'AsyncAvailableServerNames'
 :IF (0<1↑⍴serverNames)
  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
```

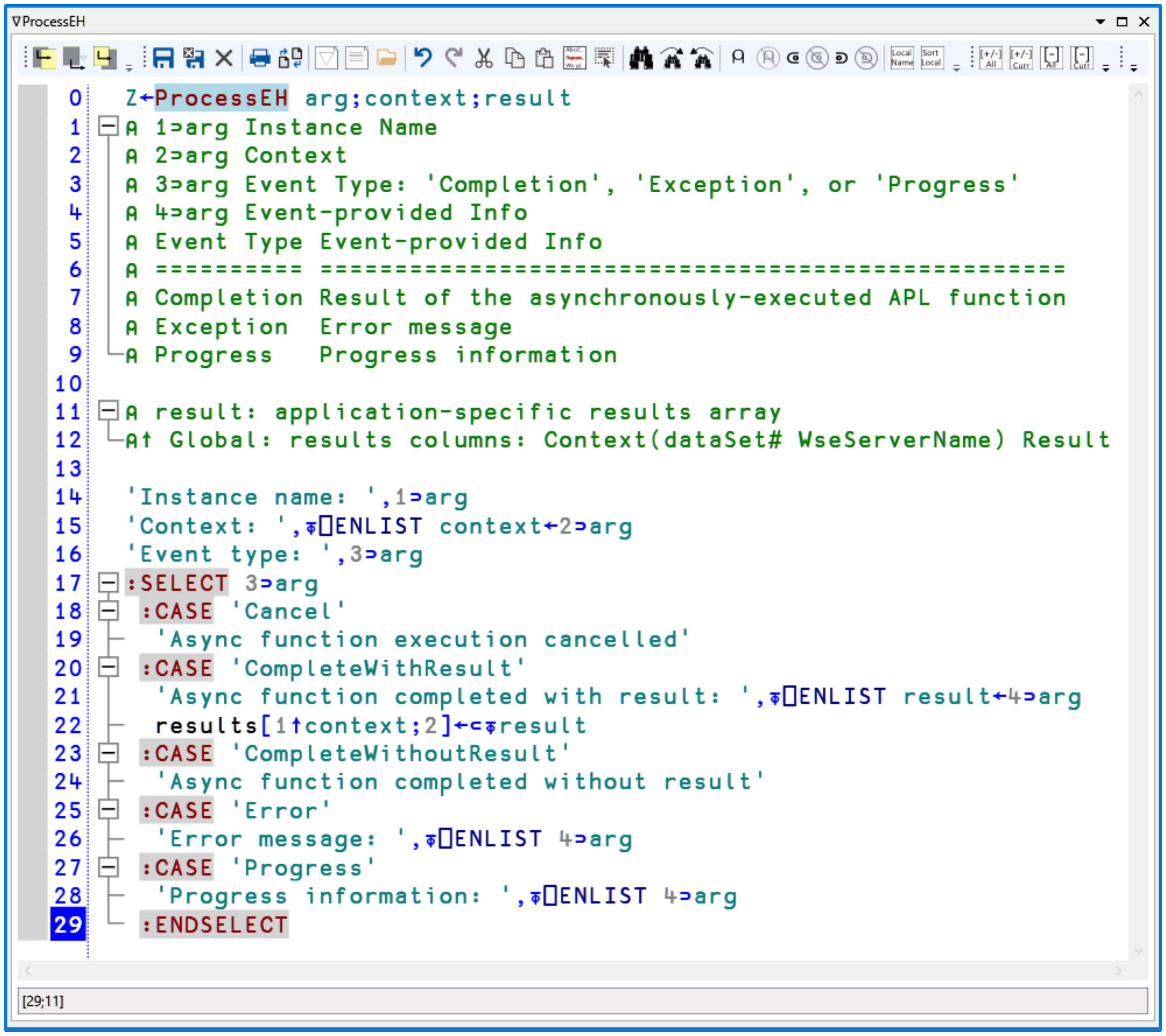


```
∇Run
 Run;context;dataSet;nDataSets;nServers;serverName;waitTime

 nDataSets←8
 results←(nDataSets,2)⍴' '
 ⍝↑ 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 ⍳nDataSets
  serverNames←'#'⎕WSE 'AsyncAvailableServerNames'
  :IF (0<1↑⍴serverNames)
   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
```

[30;47]

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
```

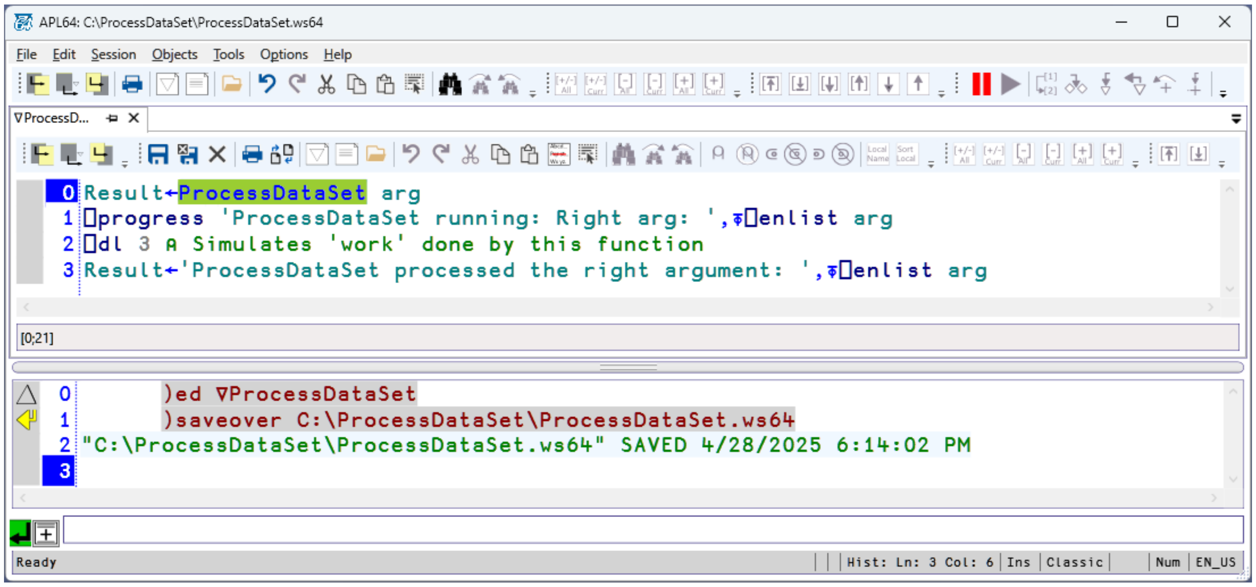


```
Z←ProcessEH arg;context;result
⍝ 1⊃arg Instance Name
⍝ 2⊃arg Context
⍝ 3⊃arg Event Type: 'Completion', 'Exception', or 'Progress'
⍝ 4⊃arg Event-provided Info
⍝ Event Type Event-provided Info
⍝ ========== ==================================================
⍝ Completion Result of the asynchronously-executed 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
```

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
```



APL64: C:\ProcessDataSet\ProcessDataSet.ws64

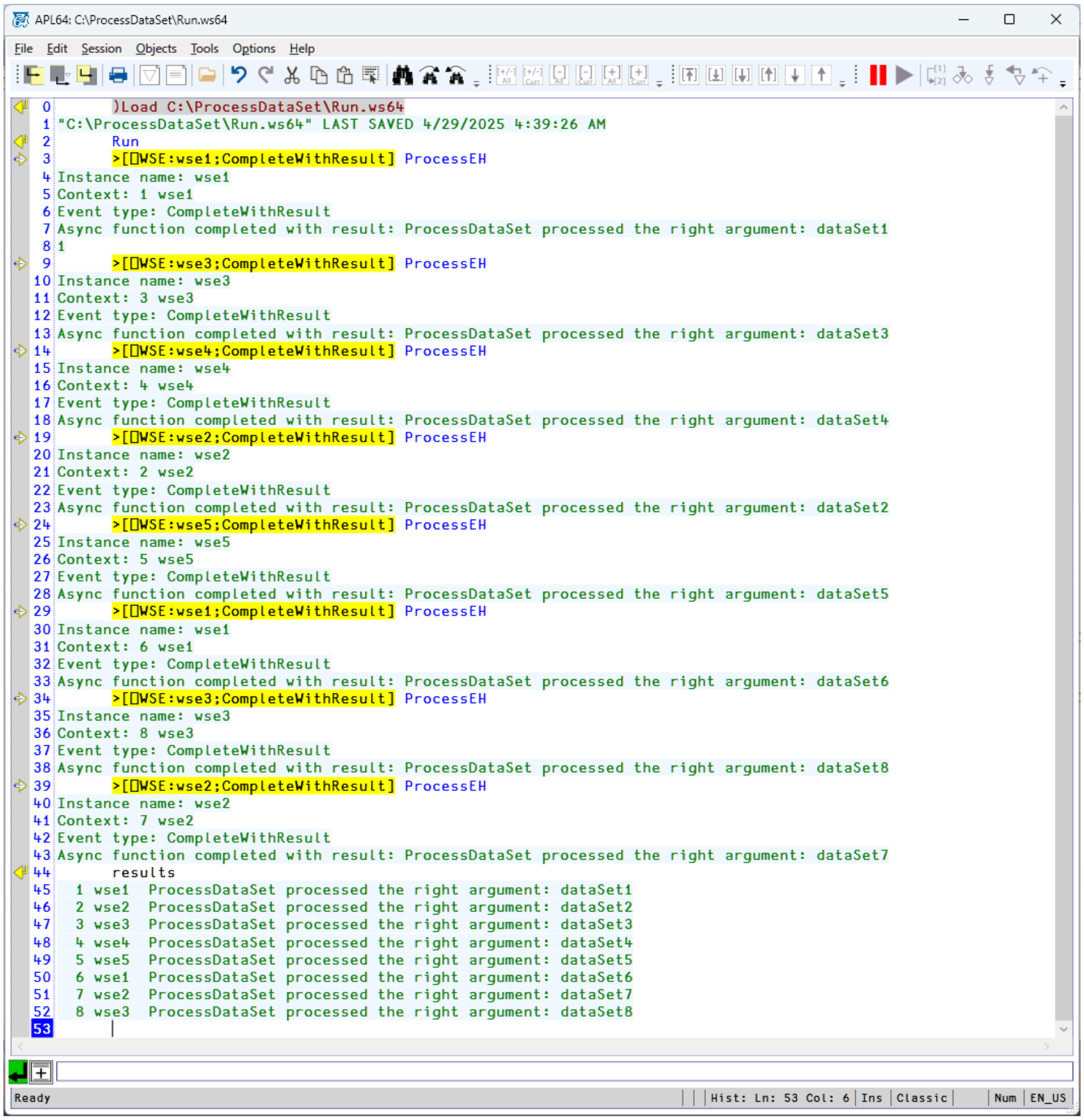
```
Result←ProcessDataSet arg
⎕progress 'ProcessDataSet running: Right arg: ',⍕⎕enlist arg
⎕dl 3 ⍝ Simulates 'work' done by this function
Result←'ProcessDataSet processed the right argument: ',⍕⎕enlist arg
```

[0;21]

```
      )ed ∇ProcessDataSet
      )saveover C:\ProcessDataSet\ProcessDataSet.ws64
"C:\ProcessDataSet\ProcessDataSet.ws64" SAVED 4/28/2025 6:14:02 PM
```

Ready | Hist: Ln: 3 Col: 6 | Ins | Classic | Num | EN_US

Test the Example



APL64: C:\ProcessDataSet\Run.ws64

```
      )Load C:\ProcessDataSet\Run.ws64
"C:\ProcessDataSet\Run.ws64" LAST SAVED 4/29/2025 4:39:26 AM
      Run
      >[⎕WSE:wse1;CompleteWithResult] ProcessEH
Instance name: wse1
Context: 1 wse1
Event type: CompleteWithResult
Async function completed with result: ProcessDataSet processed the right argument: dataSet1
1
      >[⎕WSE:wse3;CompleteWithResult] ProcessEH
Instance name: wse3
Context: 3 wse3
Event type: CompleteWithResult
Async function completed with result: ProcessDataSet processed the right argument: dataSet3
      >[⎕WSE:wse4;CompleteWithResult] ProcessEH
Instance name: wse4
Context: 4 wse4
Event type: CompleteWithResult
Async function completed with result: ProcessDataSet processed the right argument: dataSet4
      >[⎕WSE:wse2;CompleteWithResult] ProcessEH
Instance name: wse2
Context: 2 wse2
Event type: CompleteWithResult
Async function completed with result: ProcessDataSet processed the right argument: dataSet2
      >[⎕WSE:wse5;CompleteWithResult] ProcessEH
Instance name: wse5
Context: 5 wse5
Event type: CompleteWithResult
Async function completed with result: ProcessDataSet processed the right argument: dataSet5
      >[⎕WSE:wse1;CompleteWithResult] ProcessEH
Instance name: wse1
Context: 6 wse1
Event type: CompleteWithResult
Async function completed with result: ProcessDataSet processed the right argument: dataSet6
      >[⎕WSE:wse3;CompleteWithResult] ProcessEH
Instance name: wse3
Context: 8 wse3
Event type: CompleteWithResult
Async function completed with result: ProcessDataSet processed the right argument: dataSet8
      >[⎕WSE:wse2;CompleteWithResult] ProcessEH
Instance name: wse2
Context: 7 wse2
Event type: CompleteWithResult
Async function completed with result: ProcessDataSet processed the right argument: dataSet7
      results
   1 wse1  ProcessDataSet processed the right argument: dataSet1
   2 wse2  ProcessDataSet processed the right argument: dataSet2
   3 wse3  ProcessDataSet processed the right argument: dataSet3
   4 wse4  ProcessDataSet processed the right argument: dataSet4
   5 wse5  ProcessDataSet processed the right argument: dataSet5
   6 wse1  ProcessDataSet processed the right argument: dataSet6
   7 wse2  ProcessDataSet processed the right argument: dataSet7
   8 wse3  ProcessDataSet processed the right argument: dataSet8
```

Ready | Hist: Ln: 53 Col: 6 | Ins | Classic | Num | EN_US

Example: Stochastic Numerical Integration using Multiple ⎕WSE Instances

Math Details

In this example a simple form of numerical integration is implemented using multiple ⎕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:

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

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 $a \leq b$:

$a+(b-a)\times r_i$, 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 as 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 Xi 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.

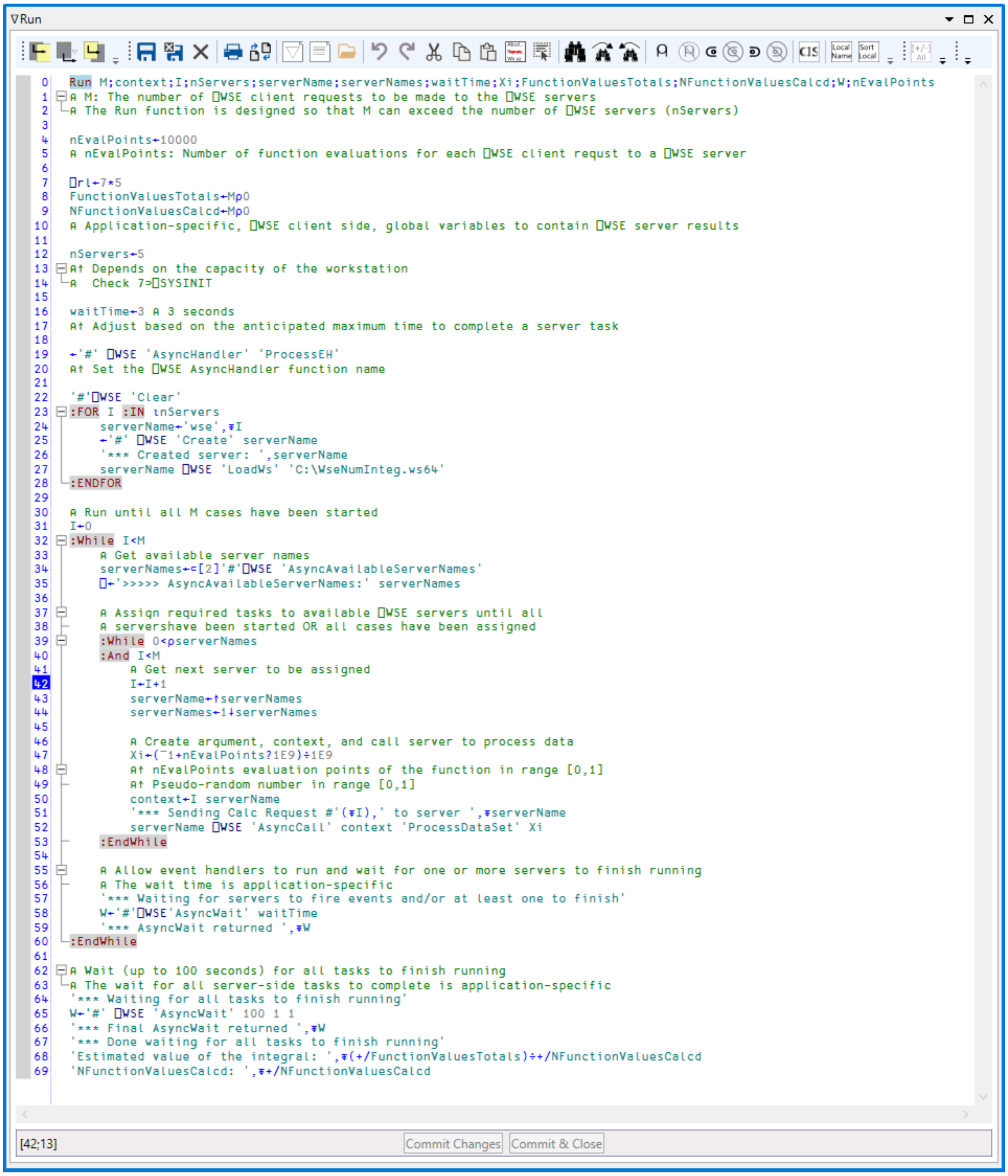
This example assumes that all servers are tasked with processing requests necessary for this example. If additional pools of ⎕WSE servers are tasked with unrelated processing requests, the example APL functions will need modification to carefully consider the ⎕WSE server instance names. For example, 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.

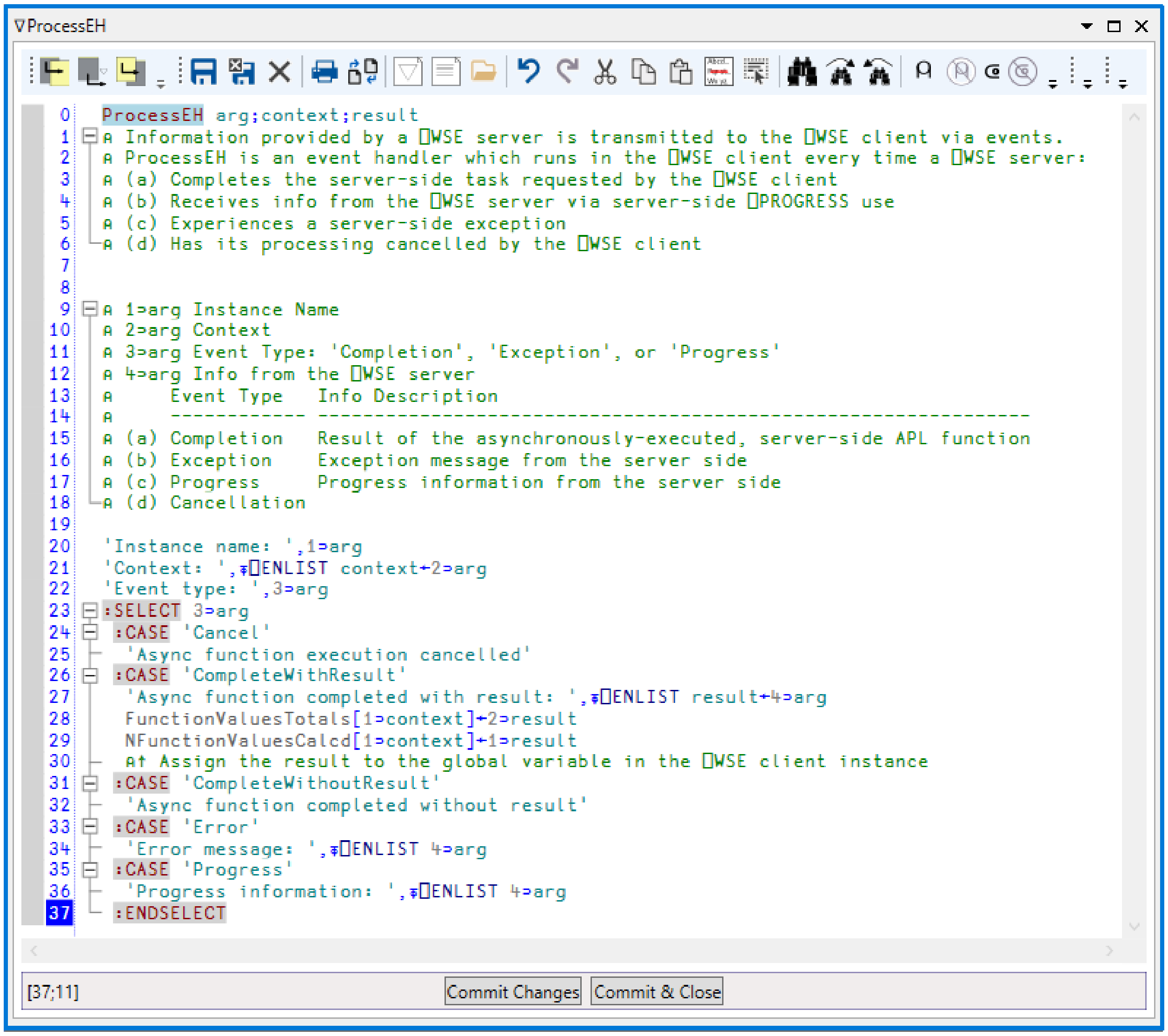
```
Run
M;context;I;nServers;serverName;serverNames;waitTime;Xi;FunctionValuesTotals;NFunctionVal
uesCalcd;W;nEvalPoints
⍝ M: The number of ⎕WSE client requests to be made to the ⎕WSE servers
⍝ The Run function is designed so that M can exceed the number of ⎕WSE servers (nServers)

nEvalPoints←10000
⍝ nEvalPoints: Number of function evaluations for each ⎕WSE client requst to a ⎕WSE server

⎕rl←7*5
FunctionValuesTotals←Mρ0
NFunctionValuesCalcd←Mρ0
⍝ 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←⊂[2]'#'⎕WSE 'AsyncAvailableServerNames'
  ⎕←'>>>>> AsyncAvailableServerNames:' serverNames

  ⍝ Assign required tasks to available ⎕WSE servers until all
  ⍝ servershave been started OR all cases have been assigned
  :While 0<ρserverNames
  :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
```



```
Run M;context;I;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 requst to a ⎕WSE server

⎕rl←7*5
FunctionValuesTotals←M⍴0
NFunctionValuesCalcd←M⍴0
⍝ 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←⊂[2]'#'⎕WSE 'AsyncAvailableServerNames'
    ⎕←'>>>>> AsyncAvailableServerNames:' serverNames

    ⍝ Assign required tasks to available ⎕WSE servers until all
    ⍝ servershave been started OR all cases have been assigned
    :While 0<⍴serverNames
    :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
```

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

```
ProcessEH arg;context;result
⍝ Information provided by a ⎕WSE server is transmitted to the ⎕WSE client via events.
⍝ ProcessEH is an event handler which runs in the ⎕WSE client every time a ⎕WSE server:
⍝ (a) Completes the server-side task requested by the ⎕WSE 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
```



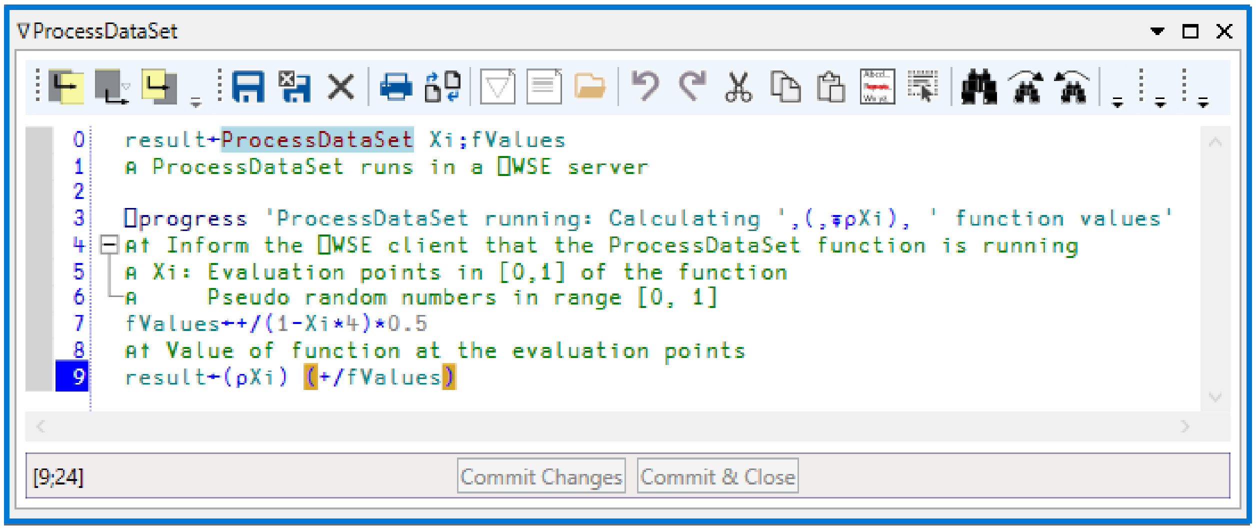
```
ProcessEH arg;context;result
⍝ Information provided by a ⎕WSE server is transmitted to the ⎕WSE client via events.
⍝ ProcessEH is an event handler which runs in the ⎕WSE client every time a ⎕WSE server:
⍝ (a) Completes the server-side task requested by the ⎕WSE client
⍝ (b) Receives info from the ⎕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
```

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 ⎕Progress 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 ',(,⍕ρXi), ' 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←(ρXi) (+/fValues)
```



∇ProcessDataSet

```
result←ProcessDataSet Xi;fValues
⍝ ProcessDataSet runs in a ⎕WSE server

⎕progress 'ProcessDataSet running: Calculating ',(⍕ρXi), ' 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←(ρXi) (+/fValues)
```

[9;24]

Test the Implementation

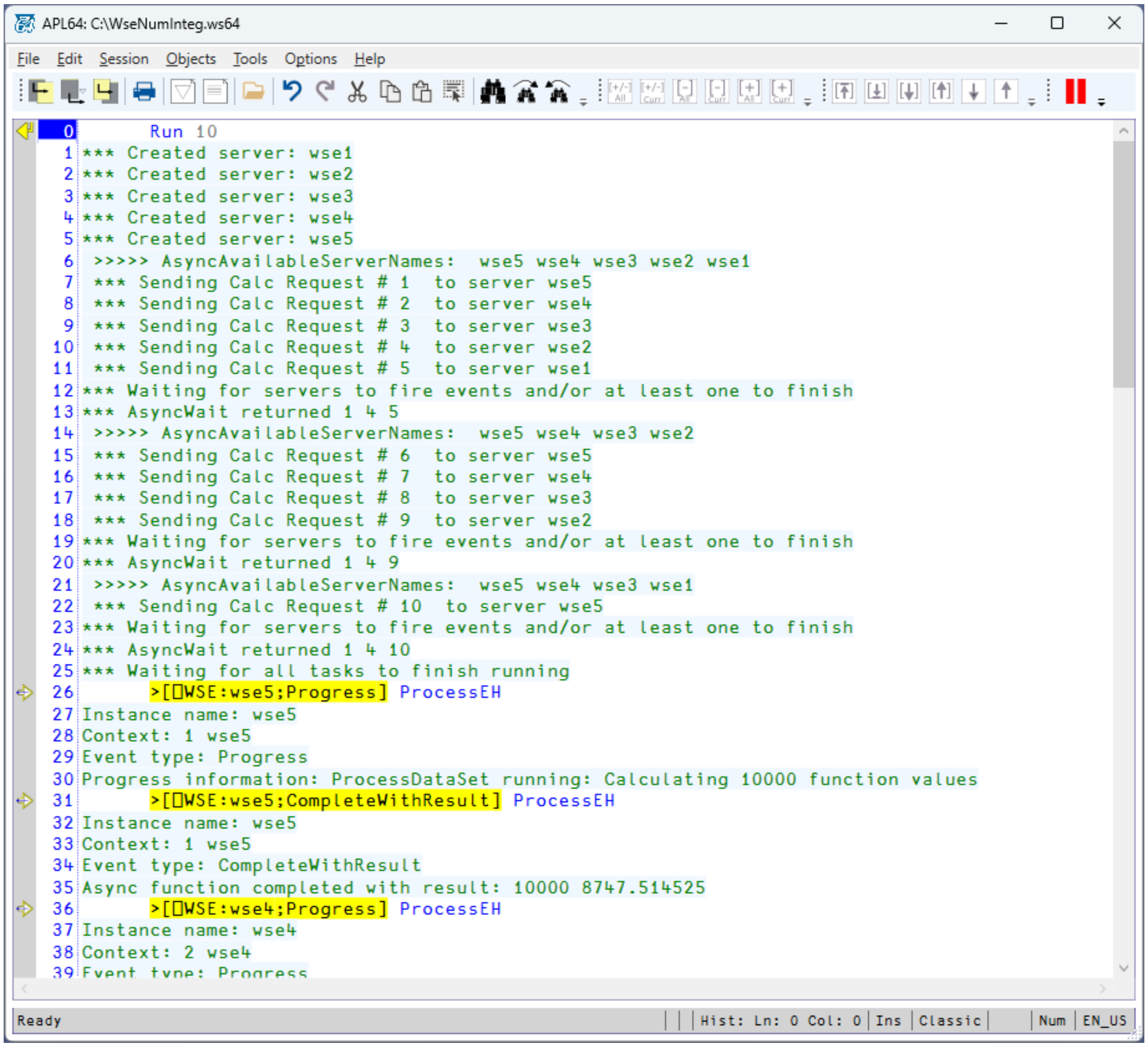
In this example:

- 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.

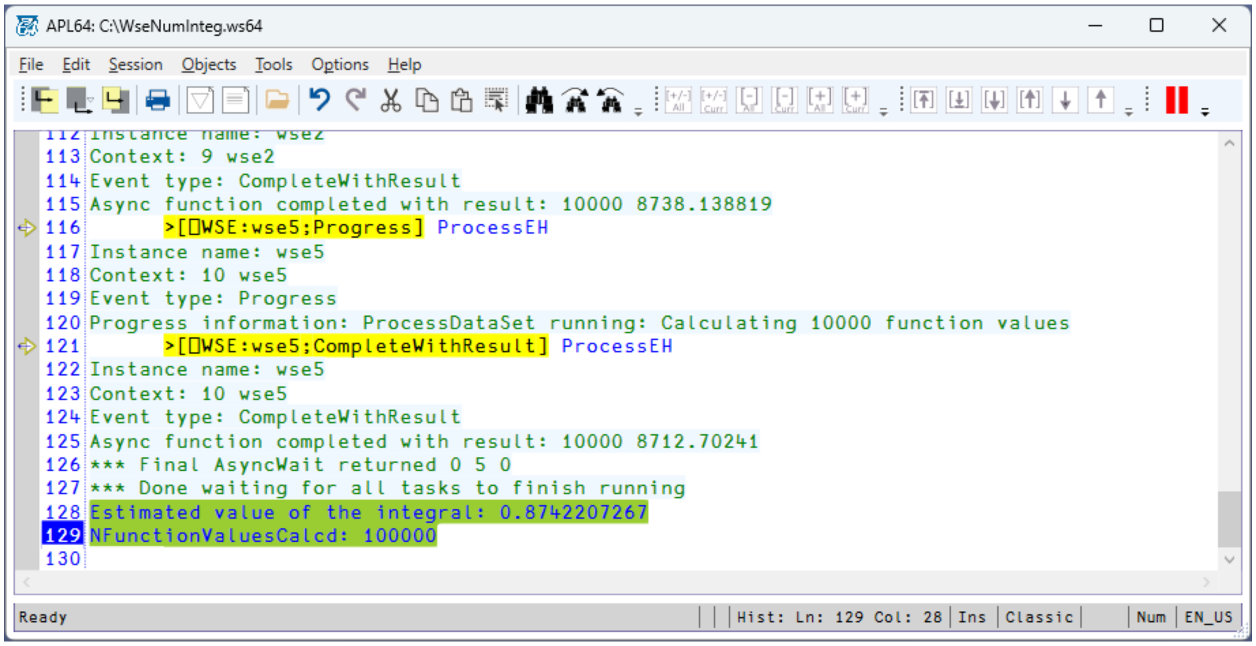
- The ⎕WSE client runs the Run function with a right argument of 10, indicating that 10 data sets will be processed asynchronously by the pool of ⎕WSE servers. When all the server-side processing is complete, the ⎕WSE client presents the result.

Here an excerpt of the output illustrating the result:


APL64: C:\WseNumInteg.ws64
File Edit Session Objects Tools Options Help
Ready
Hist: Ln: 0 Col: 0 Ins Classic Num EN_US

```
      Run 10
*** Created server: wse1
*** Created server: wse2
*** Created server: wse3
*** Created server: wse4
*** Created server: wse5
 >>>>> AsyncAvailableServerNames:  wse5 wse4 wse3 wse2 wse1
 *** Sending Calc Request # 1  to server wse5
 *** Sending Calc Request # 2  to server wse4
 *** Sending Calc Request # 3  to server wse3
 *** Sending Calc Request # 4  to server wse2
 *** Sending Calc Request # 5  to server wse1
*** Waiting for servers to fire events and/or at least one to finish
*** AsyncWait returned 1 4 5
 >>>>> AsyncAvailableServerNames:  wse5 wse4 wse3 wse2
 *** Sending Calc Request # 6  to server wse5
 *** Sending Calc Request # 7  to server wse4
 *** Sending Calc Request # 8  to server wse3
 *** Sending Calc Request # 9  to server wse2
*** Waiting for servers to fire events and/or at least one to finish
*** AsyncWait returned 1 4 9
 >>>>> AsyncAvailableServerNames:  wse5 wse4 wse3 wse1
 *** Sending Calc Request # 10  to server wse5
*** Waiting for servers to fire events and/or at least one to finish
*** AsyncWait returned 1 4 10
*** Waiting for all tasks to finish running
      >[⎕WSE:wse5;Progress] ProcessEH
Instance name: wse5
Context: 1 wse5
Event type: Progress
Progress information: ProcessDataSet running: Calculating 10000 function values
      >[⎕WSE:wse5;CompleteWithResult] ProcessEH
Instance name: wse5
Context: 1 wse5
Event type: CompleteWithResult
Async function completed with result: 10000 8747.514525
      >[⎕WSE:wse4;Progress] ProcessEH
Instance name: wse4
Context: 2 wse4
Event type: Progress
```

…

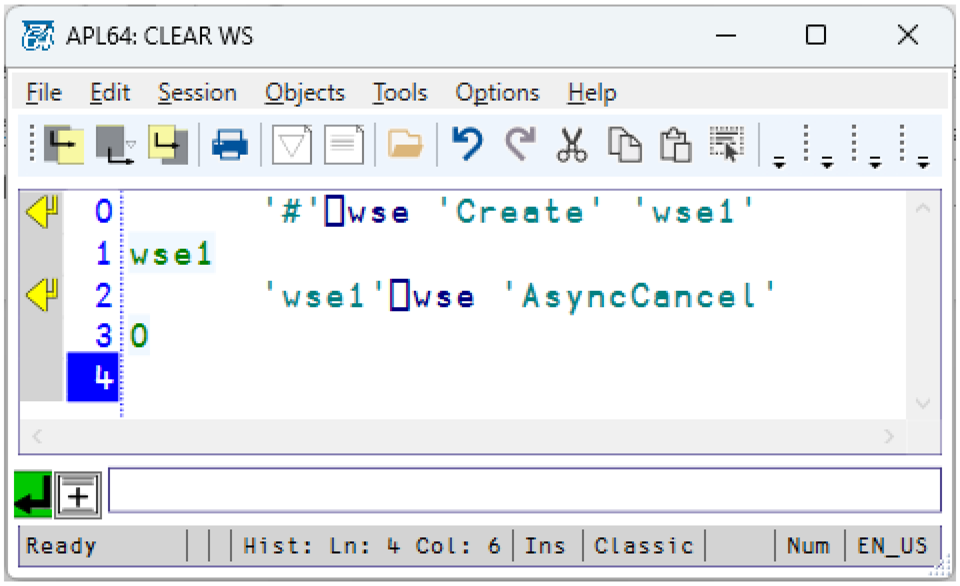


```
Instance name: wse2
Context: 9 wse2
Event type: CompleteWithResult
Async function completed with result: 10000 8738.138819
      >[⎕WSE:wse5;Progress] ProcessEH
Instance name: wse5
Context: 10 wse5
Event type: Progress
Progress information: ProcessDataSet running: Calculating 10000 function values
      >[⎕WSE:wse5;CompleteWithResult] ProcessEH
Instance name: wse5
Context: 10 wse5
Event type: CompleteWithResult
Async function completed with result: 10000 8712.70241
*** Final AsyncWait returned 0 5 0
*** Done waiting for all tasks to finish running
Estimated value of the integral: 0.8742207267
NFunctionValuesCalcd: 100000
```

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'



```
      '#'⎕wse 'Create' 'wse1'
wse1
      'wse1'⎕wse 'AsyncCancel'
0
```

AsyncStatus

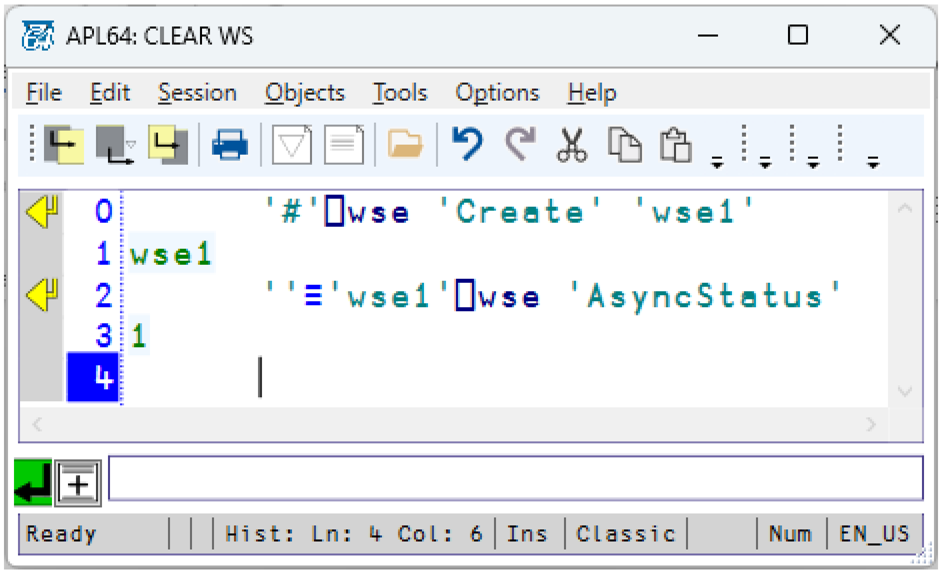
Obtain the status of the asynchronous operation, if any, currently in progress in this ⎕WSE instance. This information is transient, as requesting this information does not modify the running state of an asynchronous operation.

res ← instanceName ⎕WSE 'AsyncStatus'

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

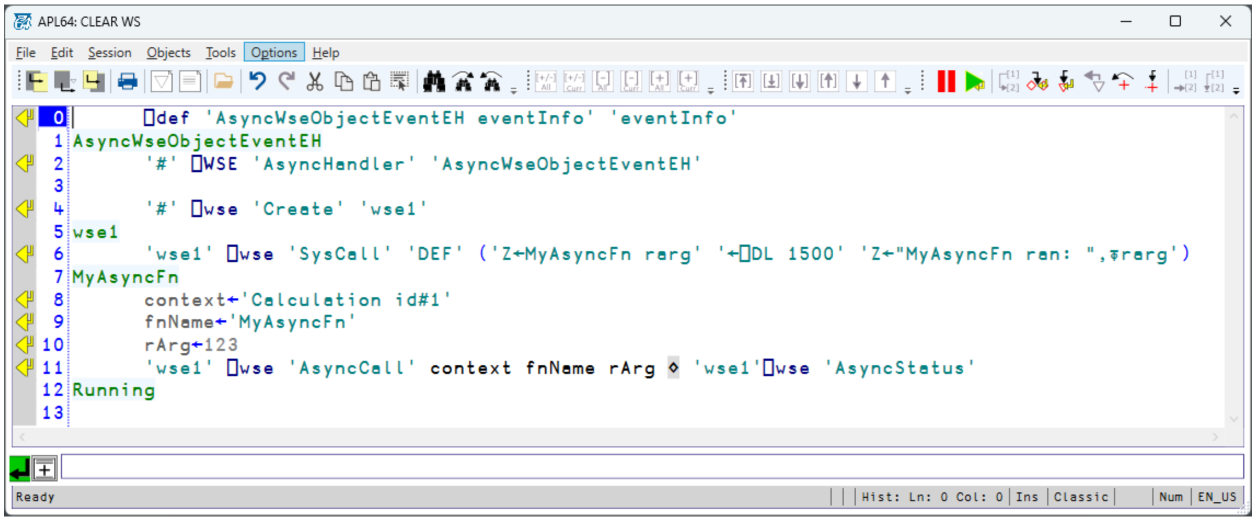
Status values for an existing .Net asynchronous operation: Created, WaitingForActivation, WaitingToRun, Running, WaitingForChildrenToComplete, RanToCompletion, Cancelled, Faulted. Not all status states are possible in APL64.

```
'#' ⎕wse 'Create' 'wse1'
'wse1'⎕wse 'AsyncStatus'
```



```
      '#'⎕wse 'Create' 'wse1'
wse1
      ''≡'wse1'⎕wse 'AsyncStatus'
1
      
```

```
⎕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'
```

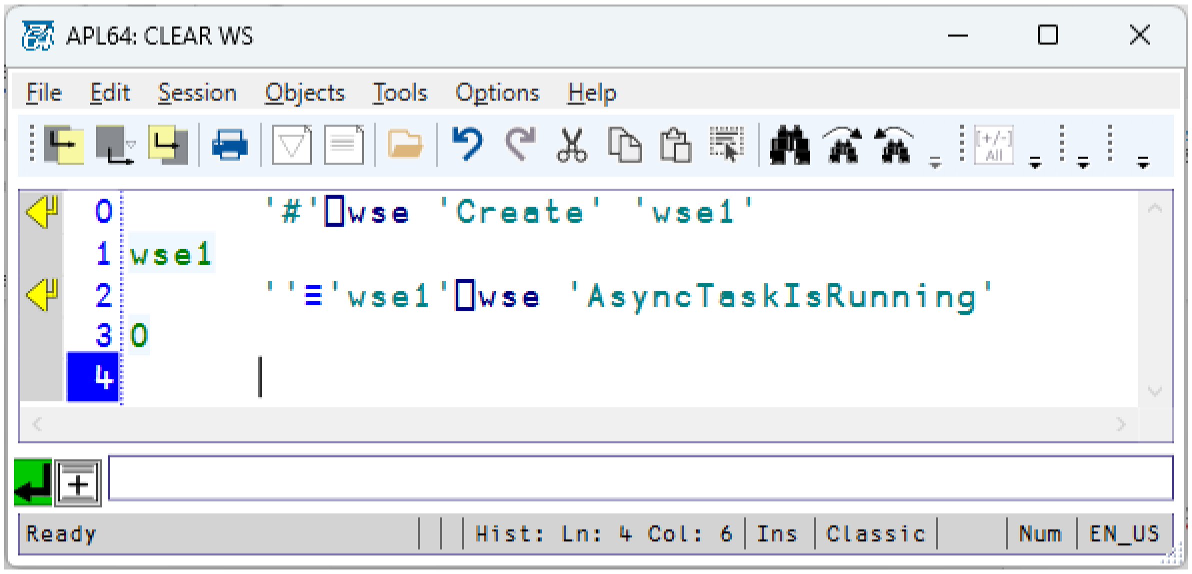


```
      ⎕def 'AsyncWseObjectEventEH eventInfo' 'eventInfo'
AsyncWseObjectEventEH
      '#' ⎕WSE 'AsyncHandler' 'AsyncWseObjectEventEH'

      '#' ⎕wse 'Create' 'wse1'
wse1
      'wse1' ⎕wse 'SysCall' 'DEF' ('Z←MyAsyncFn rarg' '←⎕DL 1500' 'Z←"MyAsyncFn ran: ",⍕rarg')
MyAsyncFn
      context←'Calculation id#1'
      fnName←'MyAsyncFn'
      rArg←123
      'wse1' ⎕wse 'AsyncCall' context fnName rArg ⋄ 'wse1'⎕wse 'AsyncStatus'
Running
```

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'



```
      '#'⎕wse 'Create' 'wse1'
wse1
      ''≡'wse1'⎕wse 'AsyncTaskIsRunning'
0
```

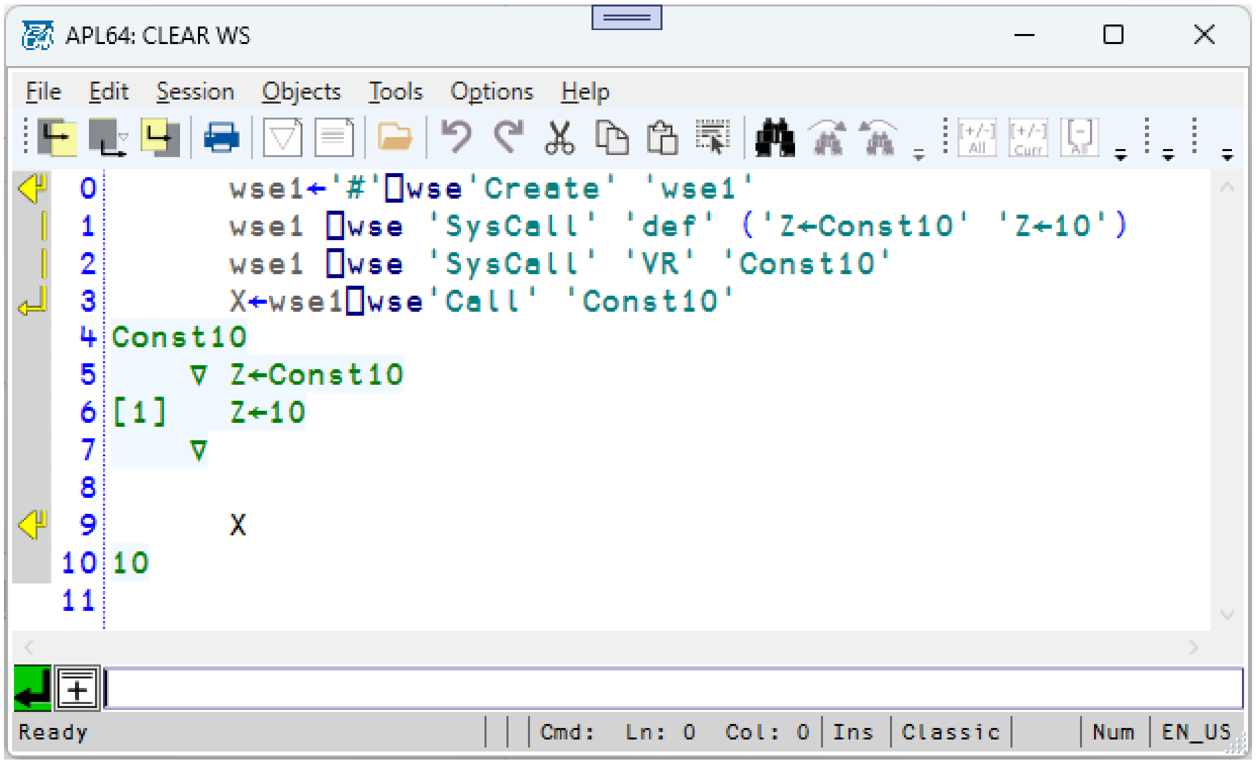
Call

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'
```


APL64: CLEAR WS
File Edit Session Objects Tools Options Help

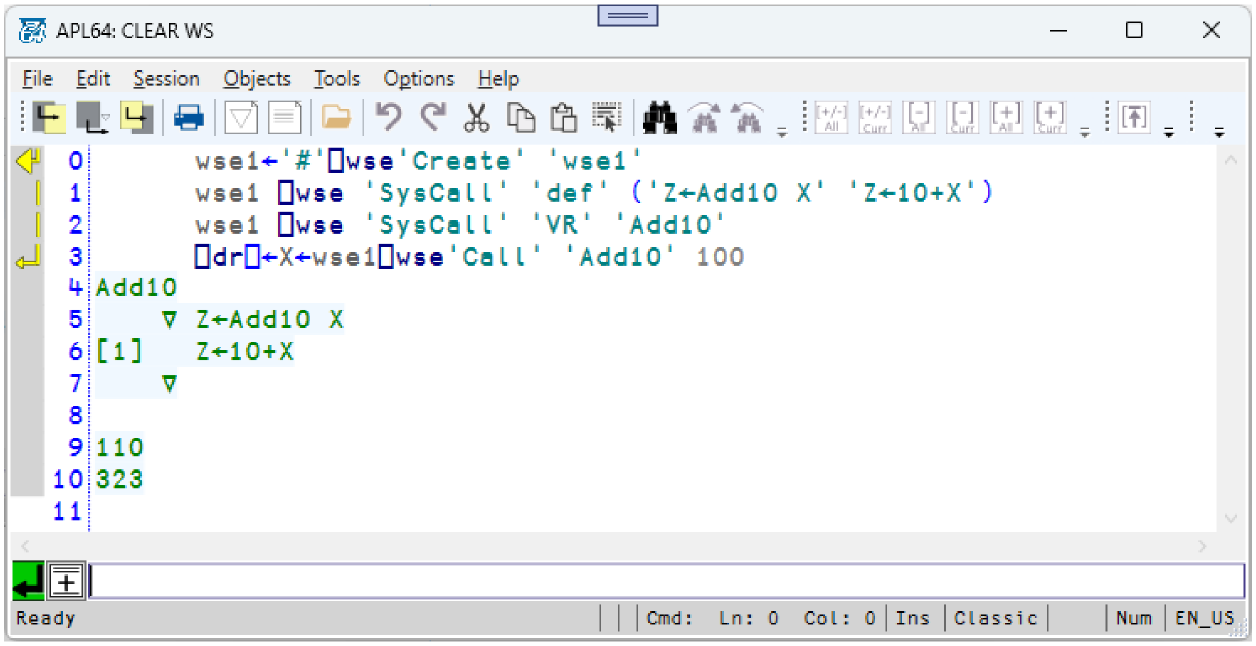
```
      wse1←'#'⎕wse'Create' 'wse1'
      wse1 ⎕wse 'SysCall' 'def' ('Z←Const10' 'Z←10')
      wse1 ⎕wse 'SysCall' 'VR' 'Const10'
      X←wse1⎕wse'Call' 'Const10'
Const10
    ∇ Z←Const10
[1]   Z←10
    ∇

      X
10
```

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

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
```



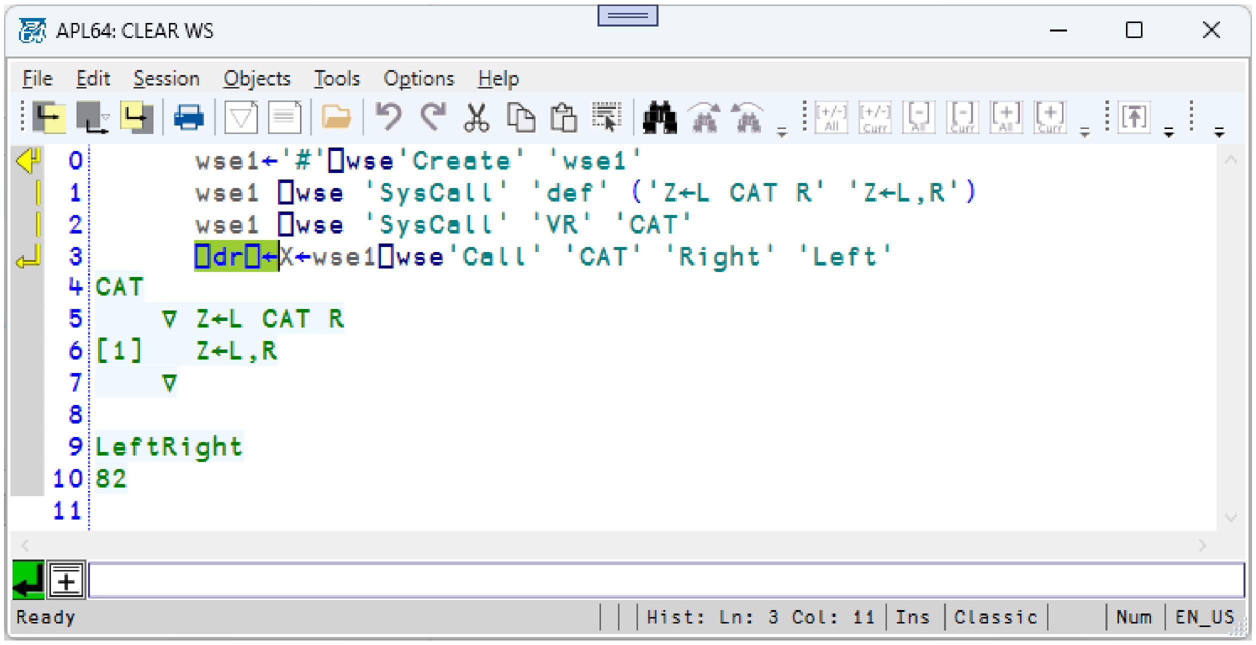
```
        wse1←'#'⎕wse'Create' 'wse1'
        wse1 ⎕wse 'SysCall' 'def' ('Z←Add10 X' 'Z←10+X')
        wse1 ⎕wse 'SysCall' 'VR' 'Add10'
        ⎕dr⎕←X←wse1⎕wse'Call' 'Add10' 100
Add10
     ∇ Z←Add10 X
[1]    Z←10+X
     ∇

110
323
```

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'
```



```
        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'
CAT
     ∇ Z←L CAT R
[1]    Z←L,R
     ∇

LeftRight
82
```

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.

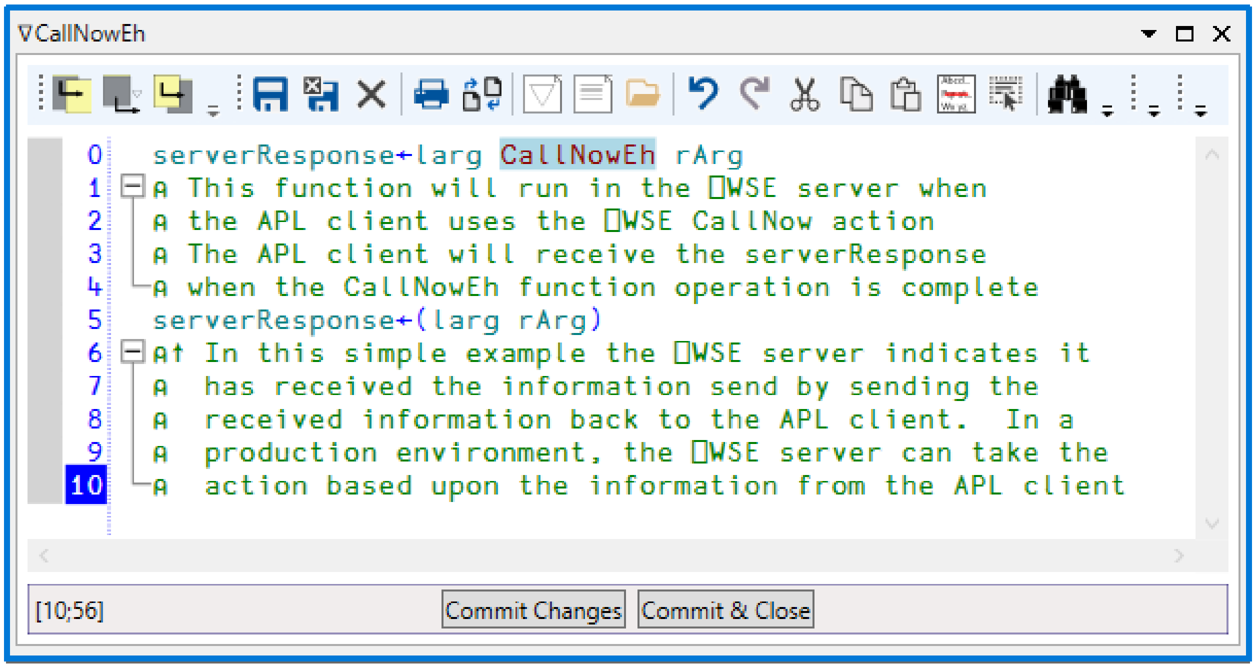
Example: ⎕WSE CallNow synchronous environment

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:

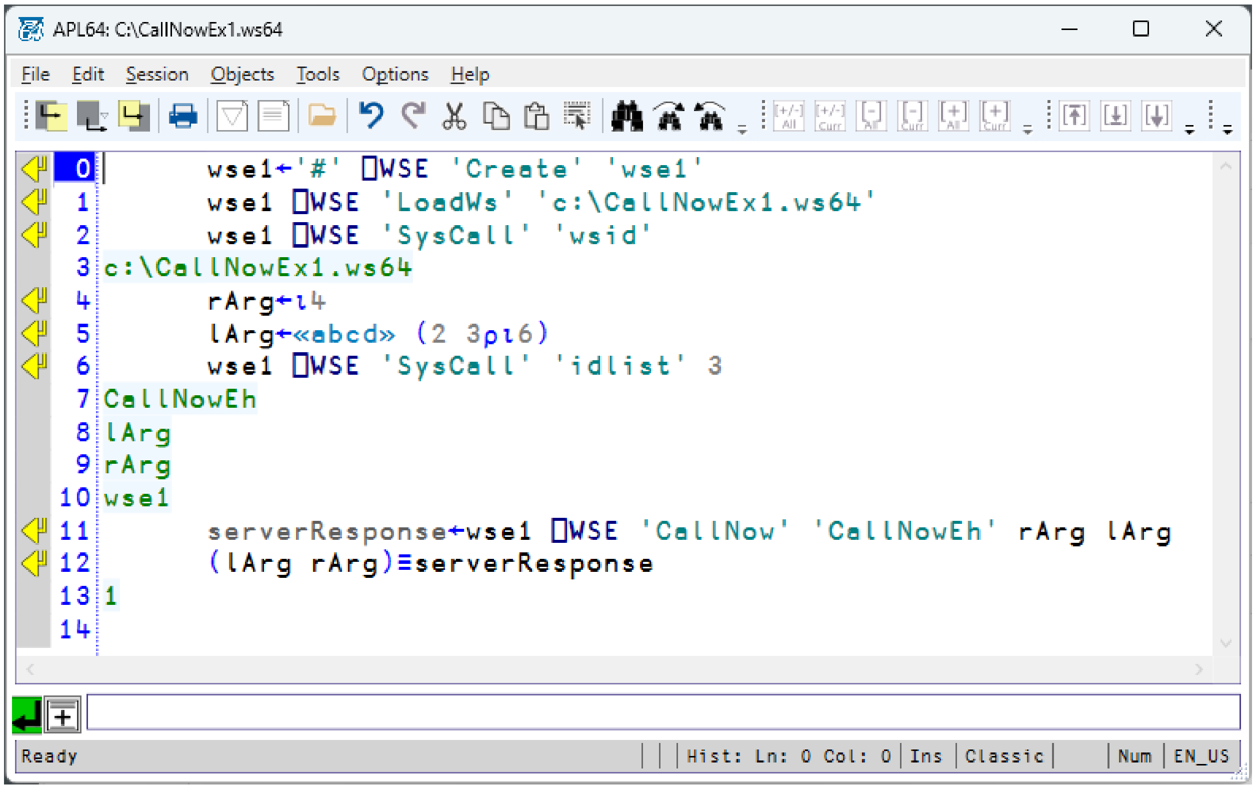
```
serverResponse←larg CallNowEh rArg
⍝ This function will run in the ⎕WSE server when
⍝ the APL client uses the ⎕WSE CallNow action
⍝ The APL client will receive the serverResponse
⍝ when the CallNowEh function operation is complete
serverResponse←(larg rArg)
⍝↑ In this simple example the ⎕WSE server 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
```



```
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←⍳4
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

```
      wse1←'#' ⎕WSE 'Create' 'wse1'
      wse1 ⎕WSE 'LoadWs' 'c:\CallNowEx1.ws64'
      wse1 ⎕WSE 'SysCall' 'wsid'
c:\CallNowEx1.ws64
      rArg←⍳4
      lArg←«abcd» (2 3⍴⍳6)
      wse1 ⎕WSE 'SysCall' 'idlist' 3
CallNowEh
lArg
rArg
wse1
      serverResponse←wse1 ⎕WSE 'CallNow' 'CallNowEh' rArg lArg
      (lArg rArg)≡serverResponse
1
```

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

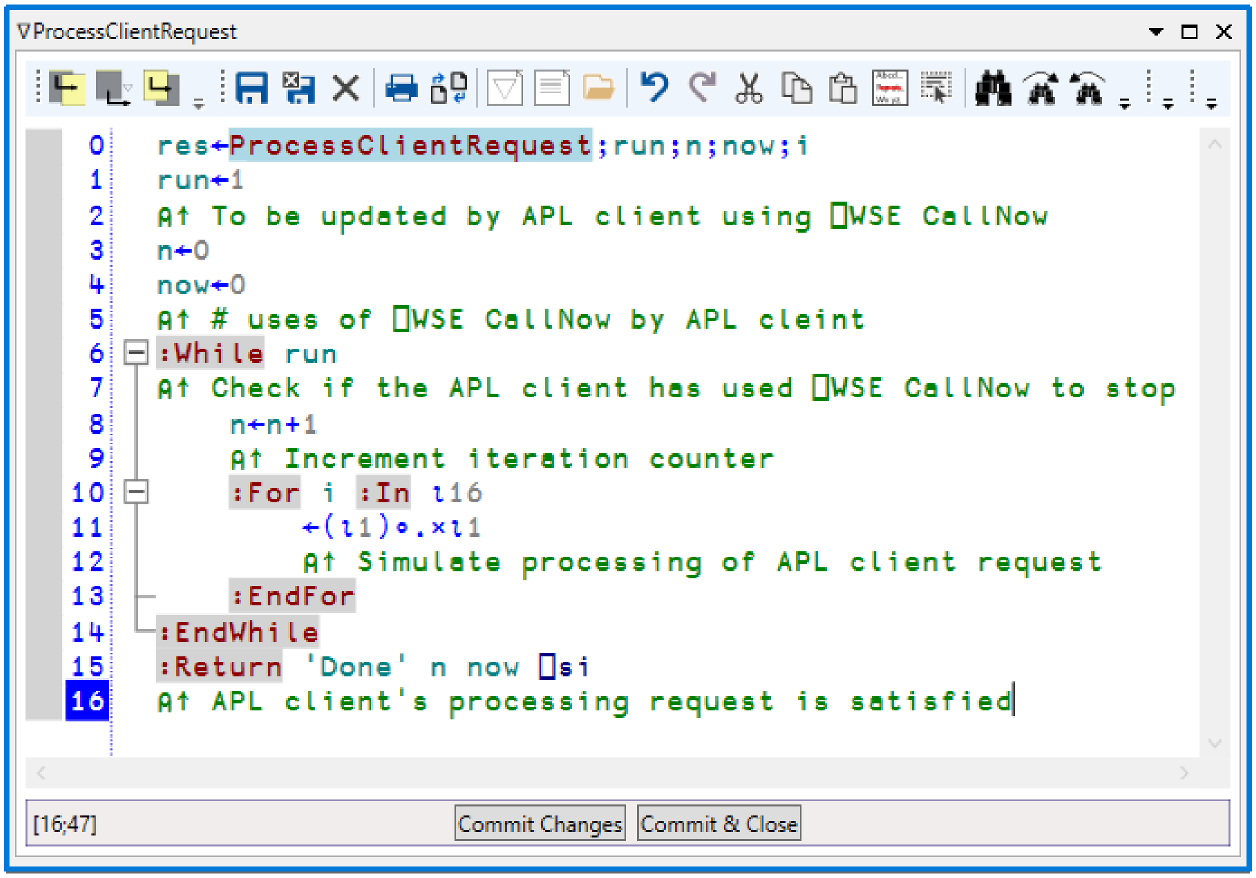
Example: ⎕WSE CallNow asynchronous parallel environment

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.

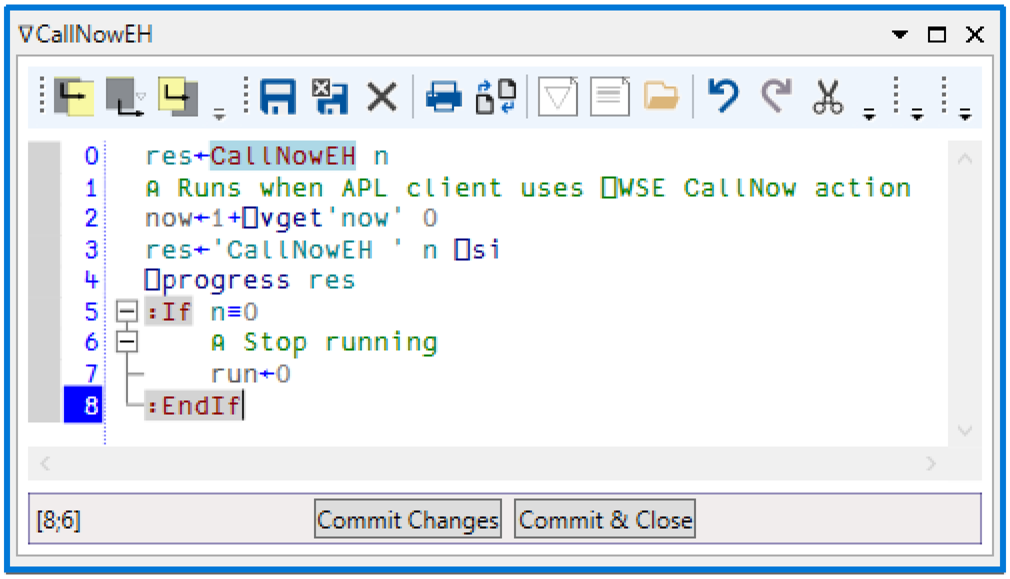
```
res←ProcessClientRequest;run;n;now;i
run←1
⍝↑ To be updated by APL client using ⎕WSE CallNow
n←0
now←0
⍝↑ # uses of ⎕WSE CallNow by APL cleint
:While run
⍝↑ Check if the APL client has used ⎕WSE CallNow to stop
  n←n+1
  ⍝↑ Increment iteration counter
  :For i :In ⍳16
    ←(⍳1)∘.×⍳1
    ⍝↑ Simulate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


∇ProcessClientRequest

```
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 cleint
:While run
⍝ Check if the APL client has used ⎕WSE CallNow to stop
    n←n+1
    ⍝ Increment iteration counter
    :For i :In ⍳16
        ←(⍳1)∘.×⍳1
        ⍝ Simulate processing of APL client request
    :EndFor
:EndWhile
:Return 'Done' n now ⎕si
⍝ APL client's processing request is satisfied
```

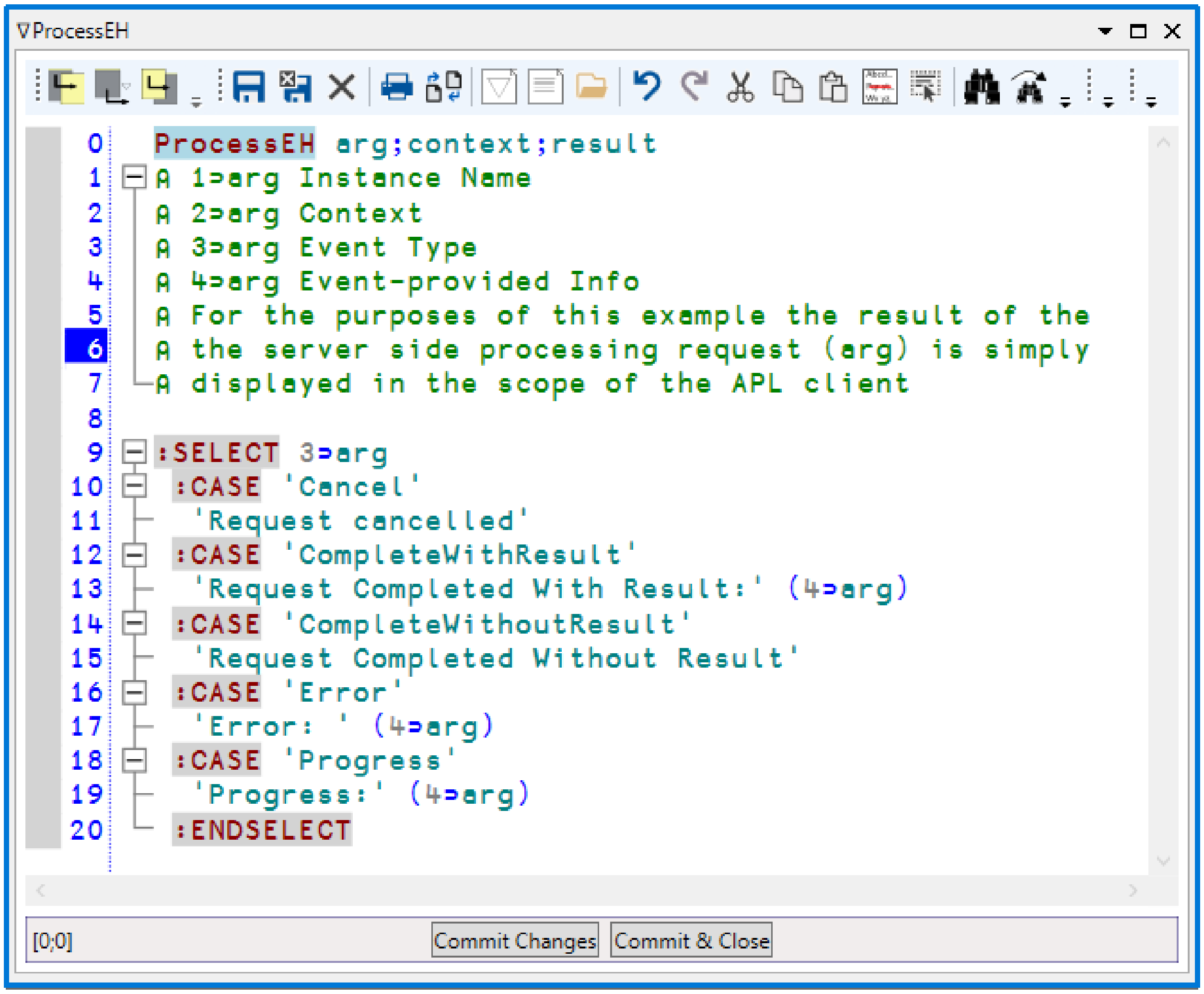
[16;47] Commit Changes Commit & Close

```
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
```



```
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
```

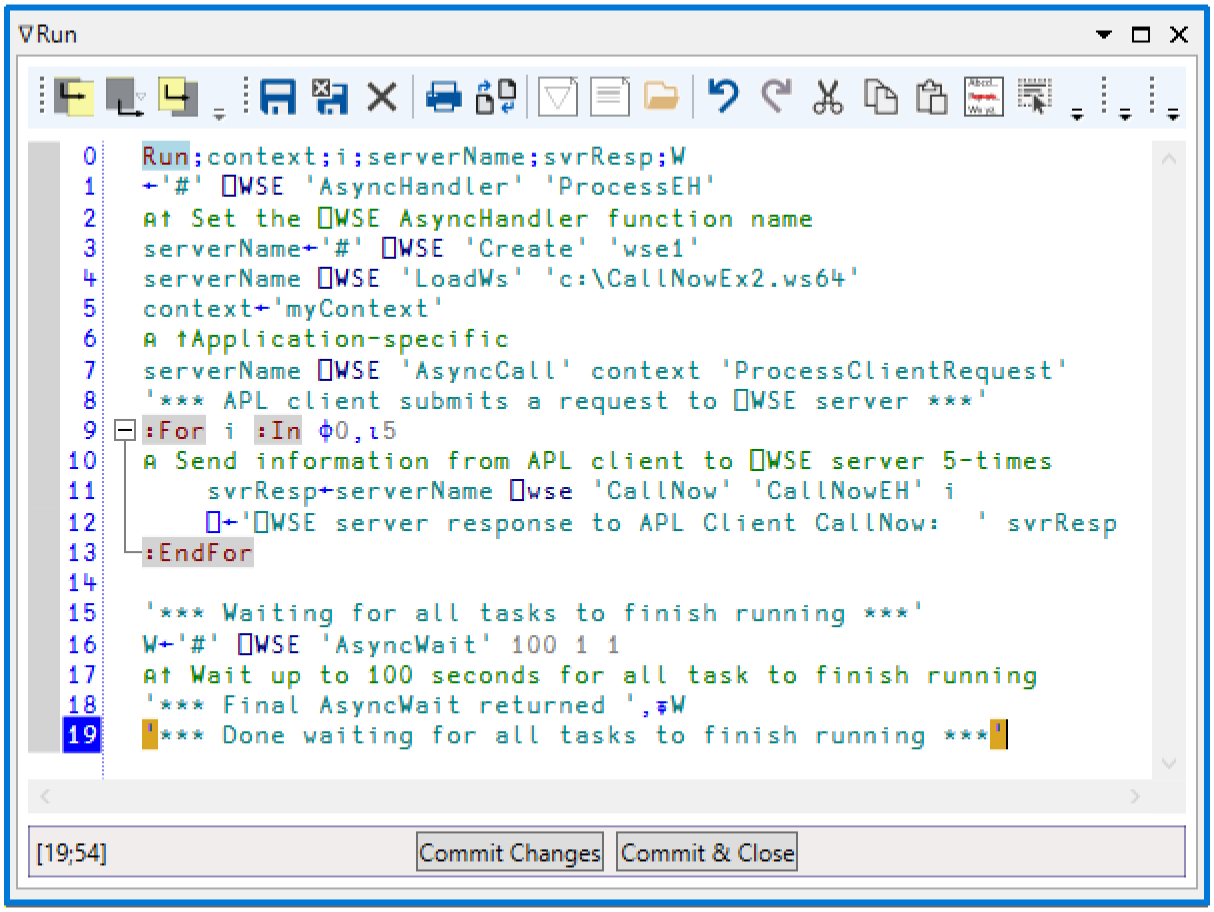


```
∇ProcessEH
 ProcessEH arg;context;result
 ⍝ 1⊃arg Instance Name
 ⍝ 2⊃arg Context
 ⍝ 3⊃arg Event Type
 ⍝ 4⊃arg Event-provided Info
 ⍝ For the purposes of this example the result of the
 ⍝ the server side processing request (arg) is simply
 ⍝ displayed in the scope of the APL client

 :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;0] Commit Changes Commit & Close

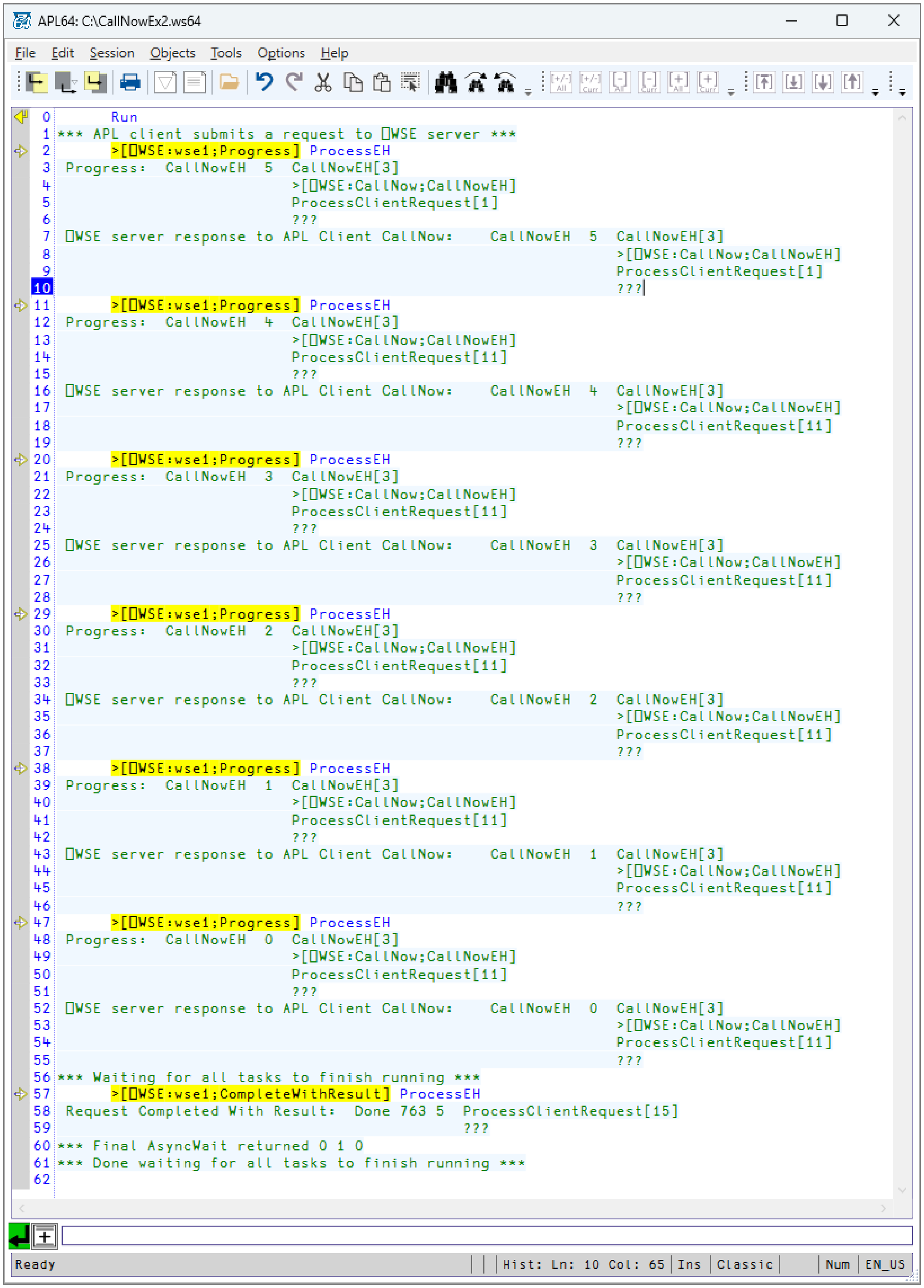
```
Run;context;i;serverName;svrResp;W
←'#' ⎕WSE 'AsyncHandler' 'ProcessEH'
⍝↑ Set the ⎕WSE AsyncHandler function name
serverName←'#' ⎕WSE 'Create' 'wse1'
serverName ⎕WSE 'LoadWs' 'c:\CallNowEx2.ws64'
context←'myContext'
⍝ ↑Application-specific
serverName ⎕WSE 'AsyncCall' context 'ProcessClientRequest'
'*** APL client submits a request to ⎕WSE server ***'
:For i :In ⌽0,⍳5
⍝ Send information from APL client to ⎕WSE server 5-times
   svrResp←serverName ⎕wse 'CallNow' 'CallNowEH' i
   ⎕←'⎕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 ',⍕W
'*** Done waiting for all tasks to finish running ***'
```



```
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,⍳5
⍝ Send information from APL client to ⎕WSE server 5-times
    svrResp←serverName ⎕wse 'CallNow' 'CallNowEH' i
    ⎕←'⎕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 ',⍕W
'*** Done waiting for all tasks to finish running ***'
```

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
Ready
Hist: Ln: 10 Col: 65 Ins Classic Num EN_US

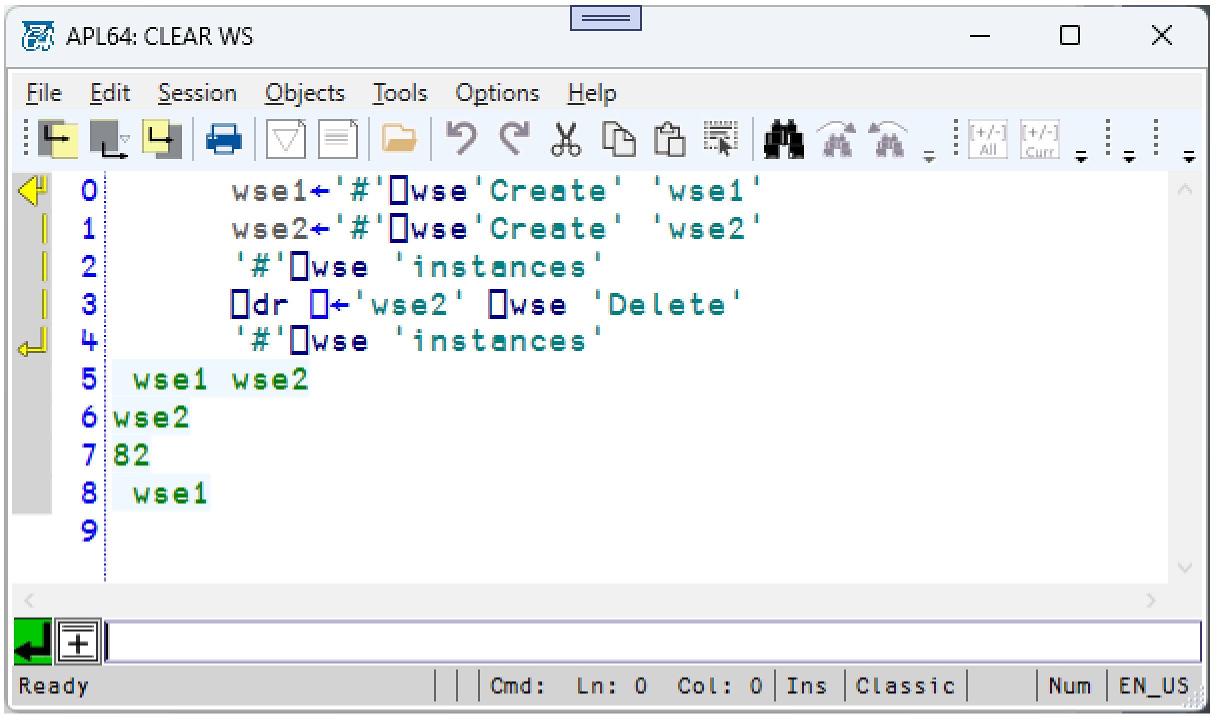
```
      Run
*** APL client submits a request to ⎕WSE server ***
      >[⎕WSE:wse1;Progress] ProcessEH
 Progress:  CallNowEH  5  CallNowEH[3]
                          >[⎕WSE:CallNow;CallNowEH]
                          ProcessClientRequest[1]
                          ???
 ⎕WSE server response to APL Client CallNow:    CallNowEH  5  CallNowEH[3]
                                                              >[⎕WSE:CallNow;CallNowEH]
                                                              ProcessClientRequest[1]
                                                              ???
      >[⎕WSE:wse1;Progress] ProcessEH
 Progress:  CallNowEH  4  CallNowEH[3]
                          >[⎕WSE:CallNow;CallNowEH]
                          ProcessClientRequest[11]
                          ???
 ⎕WSE server response to APL Client CallNow:    CallNowEH  4  CallNowEH[3]
                                                              >[⎕WSE:CallNow;CallNowEH]
                                                              ProcessClientRequest[11]
                                                              ???
      >[⎕WSE:wse1;Progress] ProcessEH
 Progress:  CallNowEH  3  CallNowEH[3]
                          >[⎕WSE:CallNow;CallNowEH]
                          ProcessClientRequest[11]
                          ???
 ⎕WSE server response to APL Client CallNow:    CallNowEH  3  CallNowEH[3]
                                                              >[⎕WSE:CallNow;CallNowEH]
                                                              ProcessClientRequest[11]
                                                              ???
      >[⎕WSE:wse1;Progress] ProcessEH
 Progress:  CallNowEH  2  CallNowEH[3]
                          >[⎕WSE:CallNow;CallNowEH]
                          ProcessClientRequest[11]
                          ???
 ⎕WSE server response to APL Client CallNow:    CallNowEH  2  CallNowEH[3]
                                                              >[⎕WSE:CallNow;CallNowEH]
                                                              ProcessClientRequest[11]
                                                              ???
      >[⎕WSE:wse1;Progress] ProcessEH
 Progress:  CallNowEH  1  CallNowEH[3]
                          >[⎕WSE:CallNow;CallNowEH]
                          ProcessClientRequest[11]
                          ???
 ⎕WSE server response to APL Client CallNow:    CallNowEH  1  CallNowEH[3]
                                                              >[⎕WSE:CallNow;CallNowEH]
                                                              ProcessClientRequest[11]
                                                              ???
      >[⎕WSE:wse1;Progress] ProcessEH
 Progress:  CallNowEH  0  CallNowEH[3]
                          >[⎕WSE:CallNow;CallNowEH]
                          ProcessClientRequest[11]
                          ???
 ⎕WSE server response to APL Client CallNow:    CallNowEH  0  CallNowEH[3]
                                                              >[⎕WSE:CallNow;CallNowEH]
                                                              ProcessClientRequest[11]
                                                              ???
*** Waiting for all tasks to finish running ***
      >[⎕WSE:wse1;CompleteWithResult] ProcessEH
 Request Completed With Result:  Done 763 5  ProcessClientRequest[15]
                                             ???
*** Final AsyncWait returned 0 1 0
*** Done waiting for all tasks to finish running ***
```

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'
'#'⎕wse 'instances'
⎕dr ⎕←'wse2' ⎕wse 'Delete'
'#'⎕wse 'instances'
```


APL64: CLEAR WS
File Edit Session Objects Tools Options Help

```
      wse1←'#'⎕wse'Create' 'wse1'
      wse2←'#'⎕wse'Create' 'wse2'
      '#'⎕wse 'instances'
      ⎕dr ⎕←'wse2' ⎕wse 'Delete'
      '#'⎕wse 'instances'
 wse1 wse2
wse2
82
 wse1
```

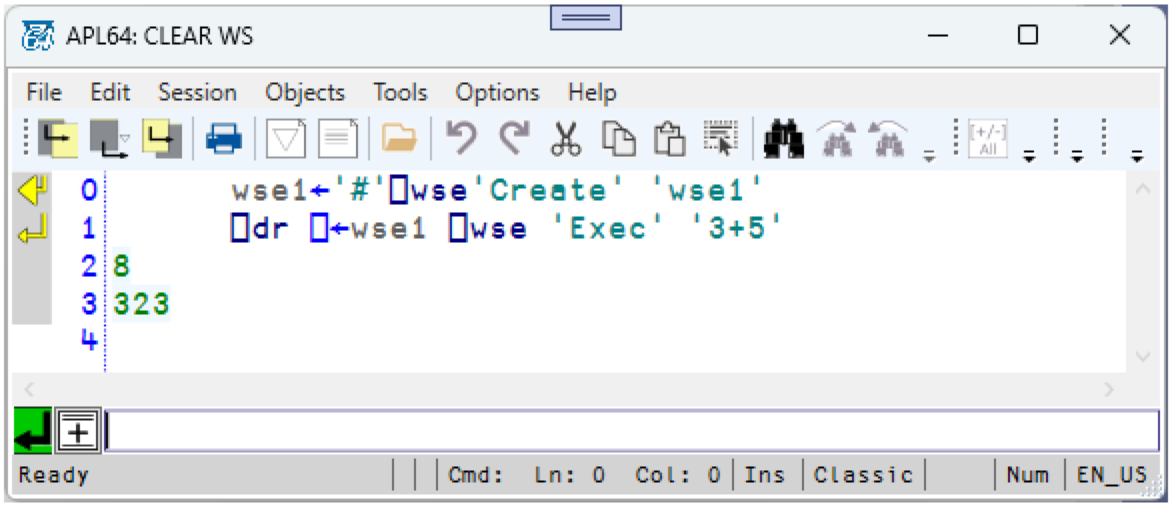
Ready
Cmd: Ln: 0 Col: 0 Ins Classic Num EN_US

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'
```



```
      wse1←'#'⎕wse'Create' 'wse1'
      ⎕dr ⎕←wse1 ⎕wse 'Exec' '3+5'
8
323
```

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

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.

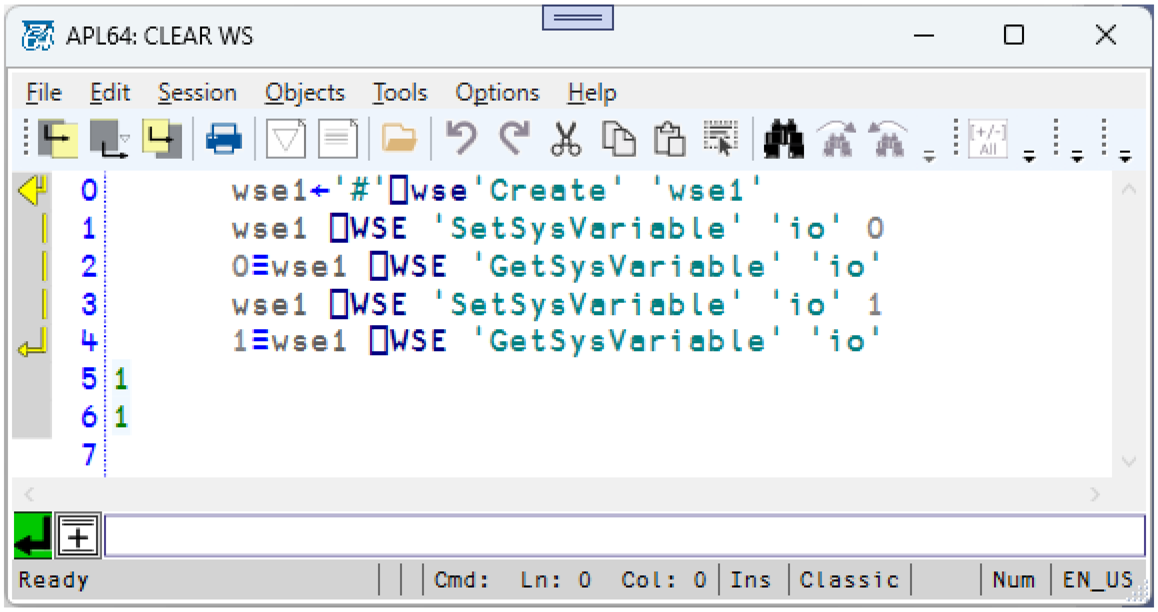
SetSysVariable

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'
```



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

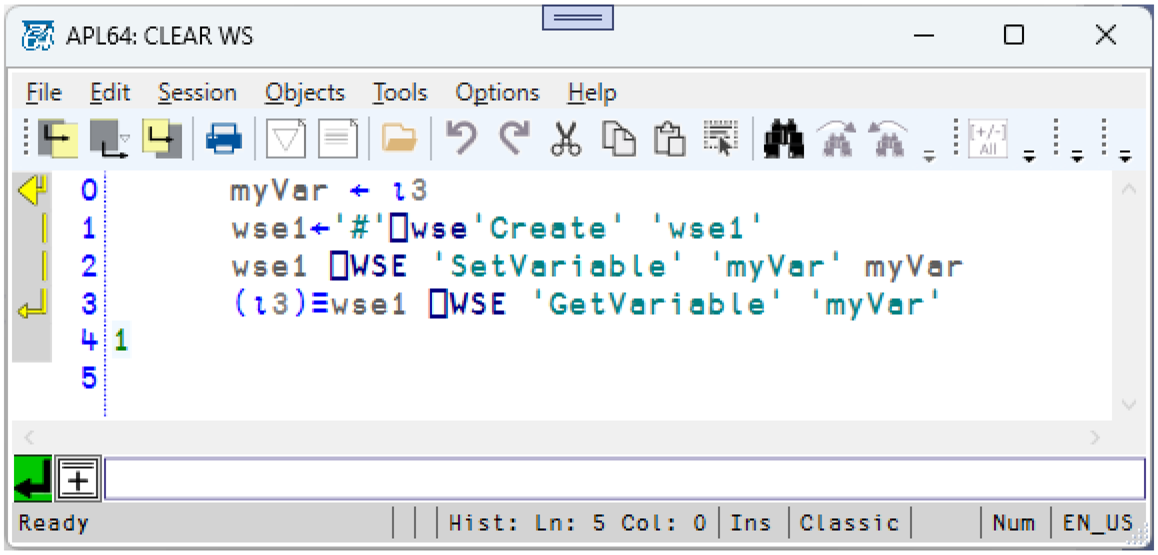
SetVariable

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'
```


APL64: CLEAR WS
File Edit Session Objects Tools Options Help

```
myVar ← ⍳3
wse1←'#'⎕wse'Create' 'wse1'
wse1 ⎕WSE 'SetVariable' 'myVar' myVar
(⍳3)≡wse1 ⎕WSE 'GetVariable' 'myVar'
1
```

Ready | Hist: Ln: 5 Col: 0 | Ins | Classic | Num | EN_US

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

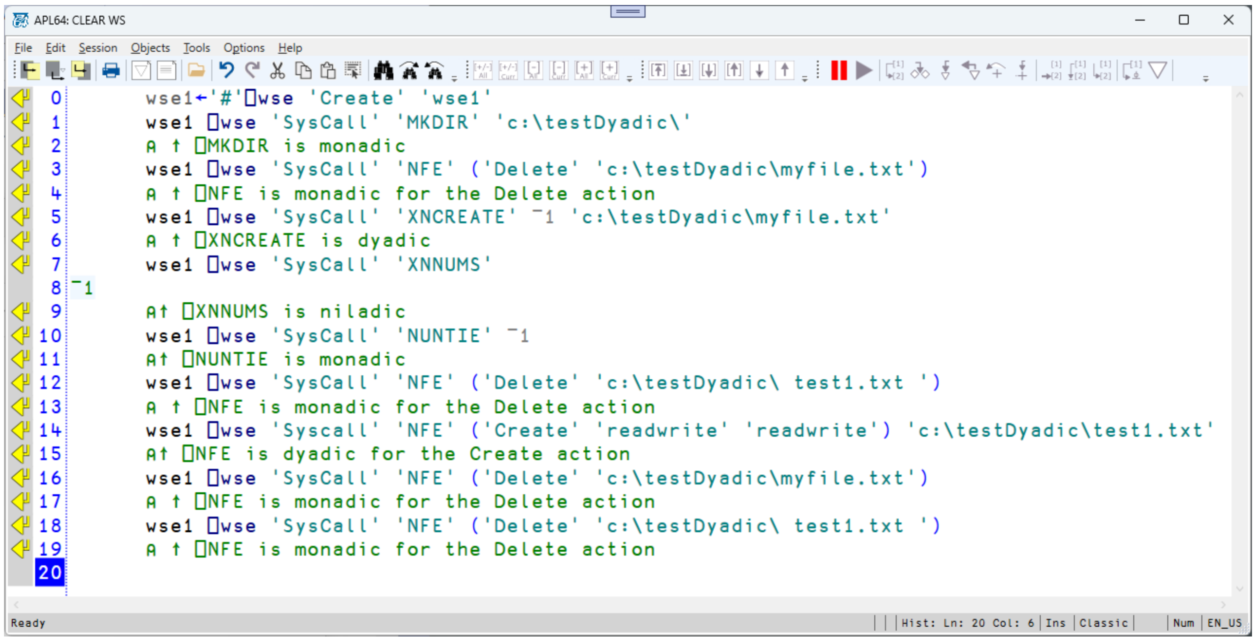
Refer to the SysCall dyadic action documentation for an example of the SysCall monadic action action.

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
```

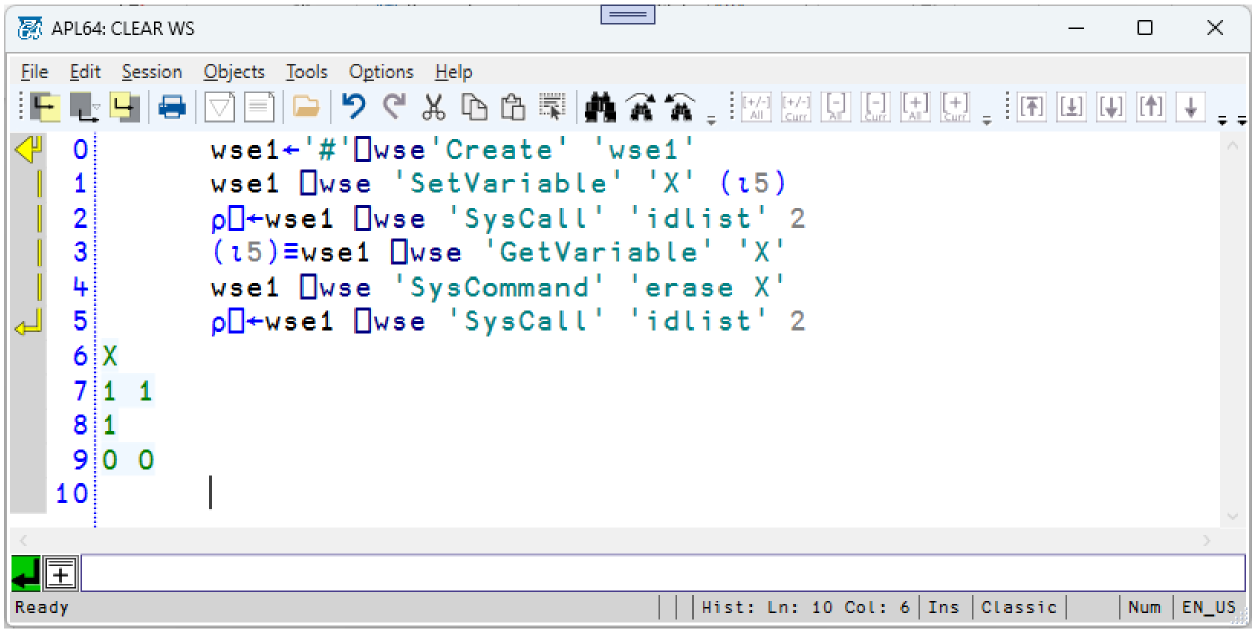

APL64: CLEAR WS
File Edit Session Objects Tools Options Help
Ready
Hist: Ln: 20 Col: 6 Ins Classic Num EN_US

```
      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'
¯1
      ⍝↑ ⎕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
```

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' (⍳5)
ρ⎕←wse1 ⎕wse 'SysCall' 'idlist' 2
(⍳5)≡wse1 ⎕wse
'GetVariable' 'X'
wse1 ⎕wse 'SysCommand' 'erase X'
ρ⎕←wse1 ⎕wse 'SysCall' 'idlist' 2
```

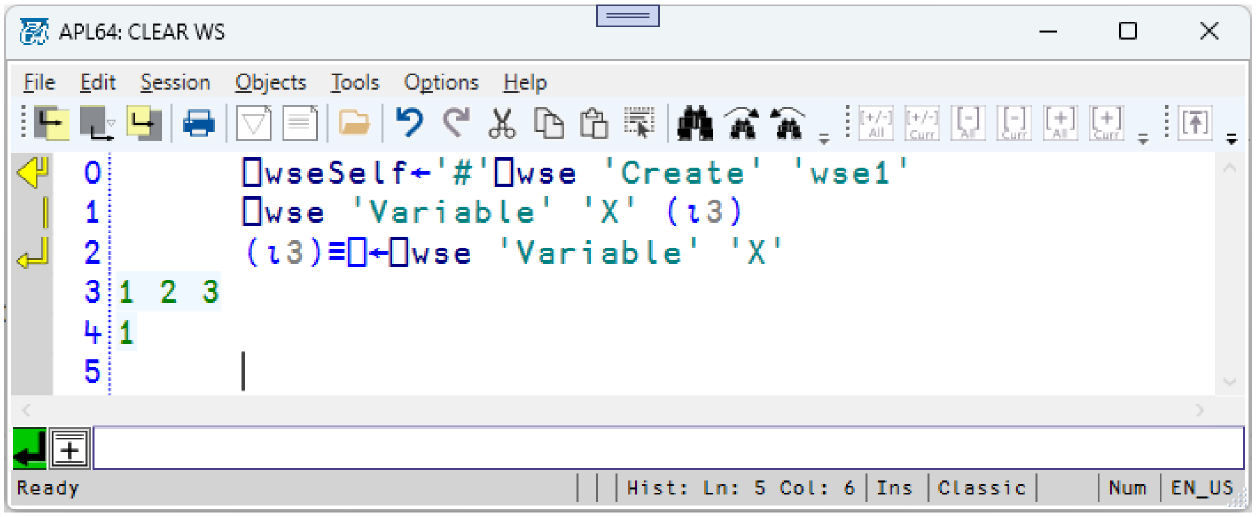


```
      wse1←'#'⎕wse'Create' 'wse1'
      wse1 ⎕wse 'SetVariable' 'X' (⍳5)
      ρ⎕←wse1 ⎕wse 'SysCall' 'idlist' 2
      (⍳5)≡wse1 ⎕wse 'GetVariable' 'X'
      wse1 ⎕wse 'SysCommand' 'erase X'
      ρ⎕←wse1 ⎕wse 'SysCall' 'idlist' 2
X
1 1
1
0 0
```

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 SetVariable actions.

```
⎕wseSelf←'#'⎕wse 'Create' 'wse1'
⎕wse 'Variable' 'X' (⍳3)
(⍳3)≡⎕←⎕wse 'Variable' 'X'
```



```
      ⎕wseSelf←'#'⎕wse 'Create' 'wse1'
      ⎕wse 'Variable' 'X' (⍳3)
      (⍳3)≡⎕←⎕wse 'Variable' 'X'
1 2 3
1
```

⎕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

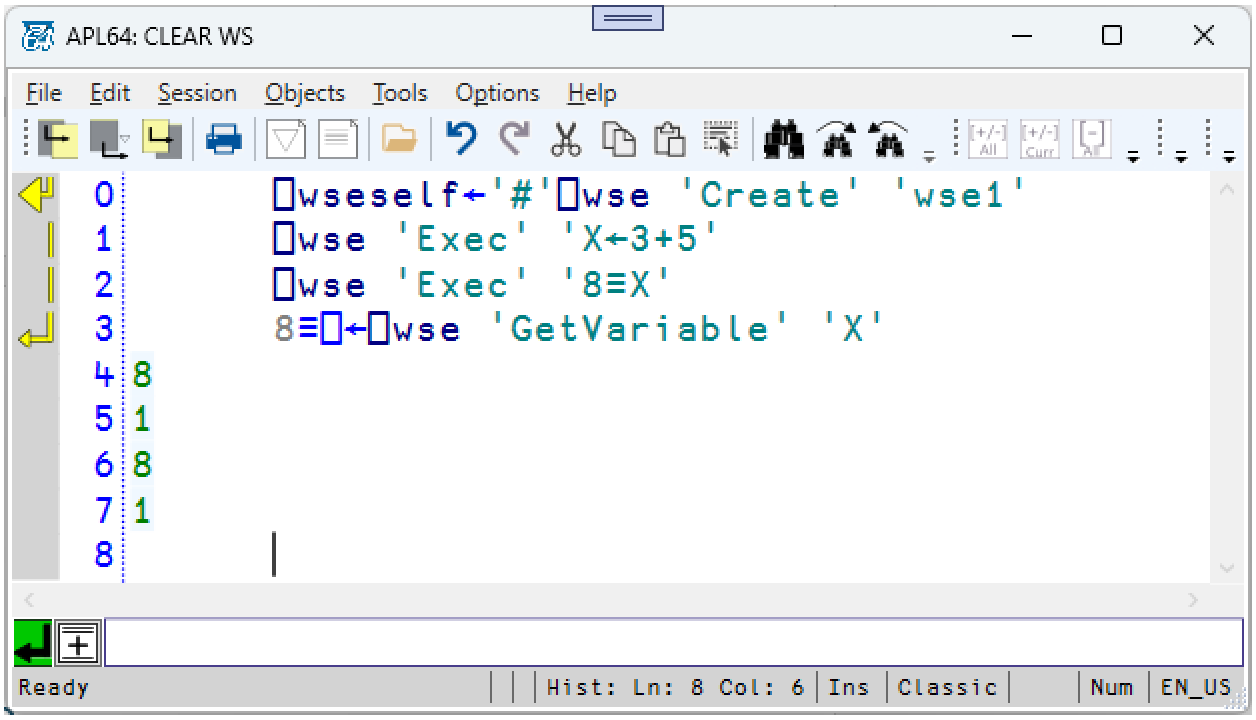
Instance Actions	**Permitted while Asynchronous Action Is Running**	**Permitted when Synchronous Action is Running**
AsyncCall	N, if targeting a ⎕WSE instance already running an AsyncCall, otherwise Y	N
AsyncCancel	Y	N
AsyncStatus	Y	N
AsyncTaskIsRunning	Y	N
Call	N	N
Delete	Y	N
Exec	N	N
GetSysVariable	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'
```



```
      ⎕wseself←'#'⎕wse 'Create' 'wse1'
      ⎕wse 'Exec' 'X←3+5'
      ⎕wse 'Exec' '8≡X'
      8≡⎕←⎕wse 'GetVariable' 'X'
8
1
8
1
```

Ready | Hist: Ln: 8 Col: 6 | Ins | Classic | Num | EN_US

⎕WseInfo System Variable

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

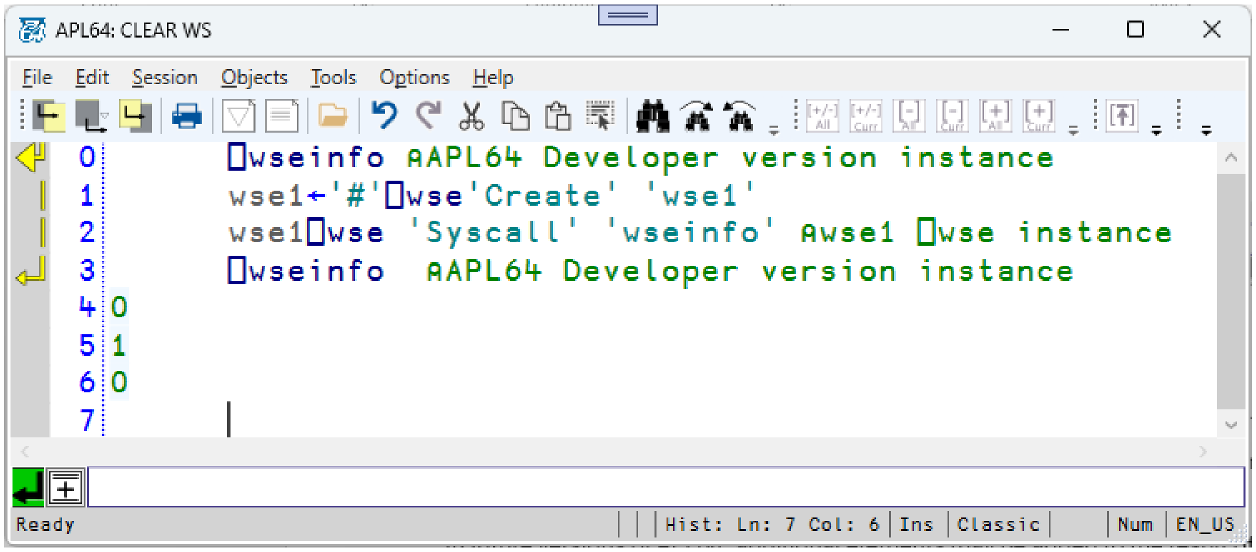
The first element of the ⎕wseinfo system variable is a Boolean scalar which indicates if the APL64 instance is running within a ⎕wse workspace engine instance.

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

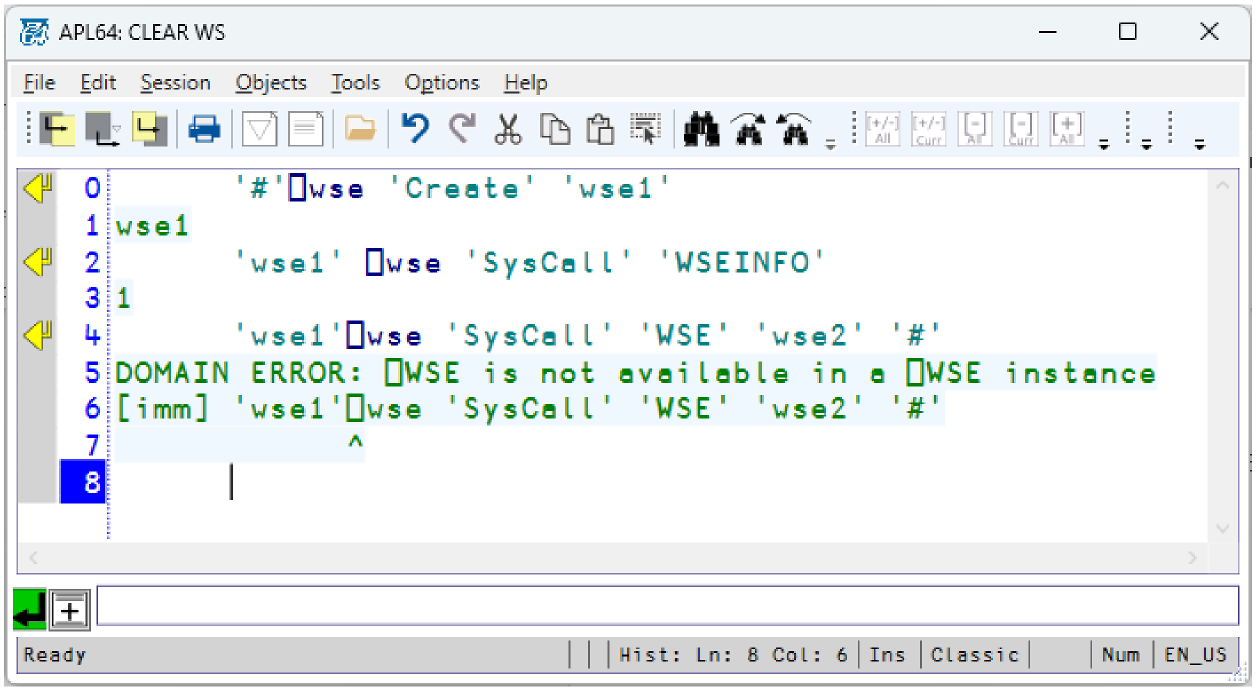
```
      ⎕wseinfo ⍝APL64 Developer version instance
      wse1←'#'⎕wse'Create' 'wse1'
      wse1⎕wse 'Syscall' 'wseinfo' ⍝wse1 ⎕wse instance
      ⎕wseinfo  ⍝APL64 Developer version instance
0
1
0
```

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

⎕WSE Comparision 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:



```
      '#'⎕wse 'Create' 'wse1'
wse1
      'wse1' ⎕wse 'SysCall' 'WSEINFO'
1
      'wse1'⎕wse 'SysCall' 'WSE' 'wse2' '#'
DOMAIN ERROR: ⎕WSE is not available in a ⎕WSE instance
[imm] 'wse1'⎕wse 'SysCall' 'WSE' 'wse2' '#'
            ^
```

⎕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 ⎕Progress when an AsyncCall action is underway

Refer to the ⎕WSE AsyncCall action example to see how the ⎕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.

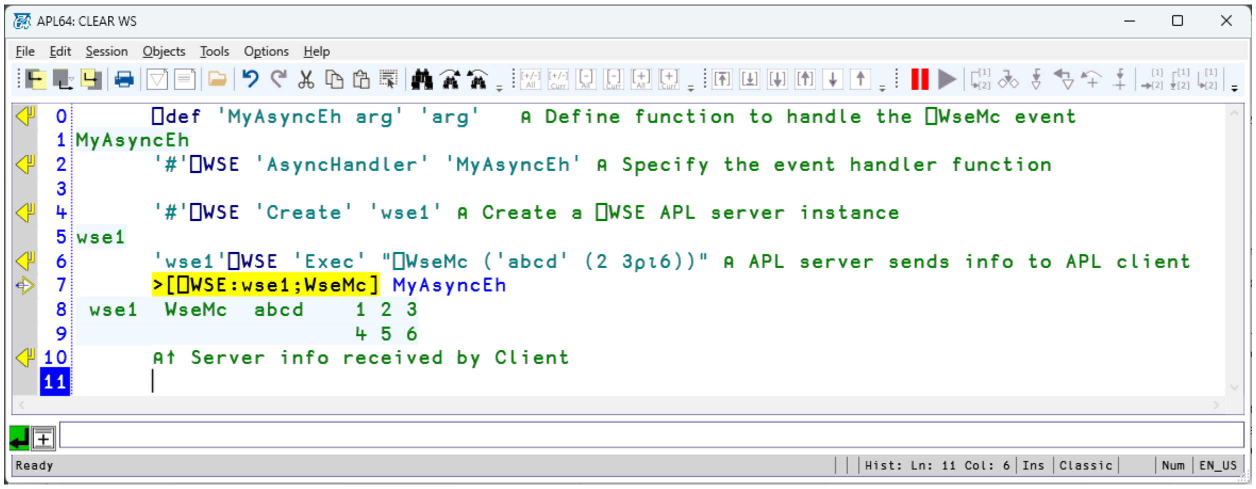
When the ⎕WseMc system function is used in a ⎕WSE server APL instance, an event is fired in the scope of the ⎕WSE APL client instance. The ⎕WSE APL client instance must subscribe to this event using the ⎕WSE AsyncHandler action.

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 3⍴⍳6))" ⍝ APL server sends info to APL client
⍝↑ Server info received by Client
```


APL64: CLEAR WS
File Edit Session Objects Tools Options Help
Ready
Hist: Ln: 11 Col: 6 Ins Classic Num EN_US

```
      ⎕def 'MyAsyncEh arg' 'arg'    ⍝ Define function to handle the ⎕WseMc event
MyAsyncEh
      '#'⎕WSE 'AsyncHandler' 'MyAsyncEh' ⍝ Specify the event handler function

      '#'⎕WSE 'Create' 'wse1' ⍝ Create a ⎕WSE APL server instance
wse1
      'wse1'⎕WSE 'Exec' "⎕WseMc ('abcd' (2 3ρι6))" ⍝ APL server sends info to APL client
      >[⎕WSE:wse1;WseMc] MyAsyncEh
 wse1  WseMc  abcd     1 2 3
                       4 5 6
      ⍝↑ Server info received by Client
```

In this ‘artificial’ example, the statement: 'wse1'⎕WSE 'Exec' "⎕WseMc ('abcd' (2 3ρι6)) 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.