# APLNext Supervisor Application Prototypes

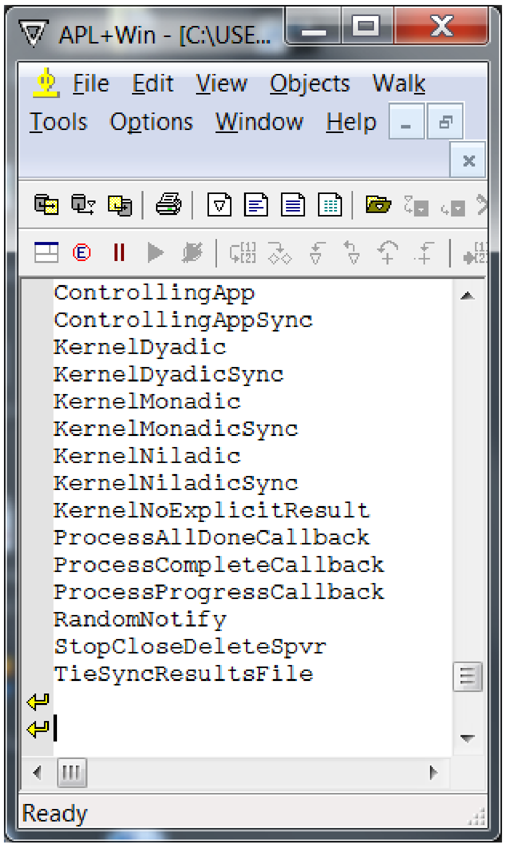
## Prototype APL+Win Workspace

The APL+Win workspace, ‘SUPERVISORTEST.w3’, associated with the APLNext Supervisor v1.9.4.0 includes prototype APL+Win functions that illustrate the operation of the APLNext Supervisor.

There are two ‘controlling application’ functions in this workspace so that the ‘async’ and ‘sync’ APLNext Supervisor operation modes can be illustrated. Refer to the APL+Win v15-compatible ‘SUPERVISORTEST.ws’ for the APL+Win source code.:

- The ‘ControllingApp’ function illustrates the asynchronous mode of the APLNext Supervisor using the ‘Start’ method.
    - This function uses the global ‘ProcessCompleteResult’ variable to contain the ‘Kernel Function’ task processing results received by the ‘ProcessCompleteCallback’ event handler function.
    - The ‘Kernel Functions’ ‘KernelDyadic’, ‘KernelMonadic’ and ‘KernelNiladic’ are used when the ‘ControllingApp’ submits tasks using the ‘BeginCall’ method.
    - The ‘ProcessAllDoneCallback’ event is subscribed in the ‘ControllingApp’ function after all tasks have been queued using the ‘BeginCall’ method. The ‘ProcessAllDoneCallback’ event handler function defers the Stop, Close and Delete of the APLNext Supervisor instance.
    - The ‘ControllingApp’ function illustrates subscribing to the ‘onXProcessProgressCallback’ event and creating a container variable to accumulate, if desired, the information provided by the ‘ProcessProgressCallback’ event.
    - The ‘ProcessProgressCallback’ event handler function illustrates recording and displaying the information provided by the ‘ProcessProgressCallback’ event.
    - The ‘KernelDyadic’ function illustrates using the APL+Win ‘Notify’ method to trigger the ‘ProcessProgressCallback’ event. With this technique the ‘KernelDyadic’ function provides processing status or other information to the domain of the ‘controlling application’

- The ‘ControllingAppSync’ function illustrates the synchronous mode of the APLNext Supervisor using the ‘StartSync’ method.
    - The ‘ControllingAppSync’ submits tasks using the ‘BeginCall’ method to The APLNext Supervisor task queue.
    - Since the APLNext Supervisor events are not used in ‘sync’ mode, the ‘kernel function’ persists its processing results in a data store shared between the domains of the ‘controlling application’ and the ‘kernel function’. This sample uses a shared APL+Win colossal component file to contain the ‘Kernel Function’ task processing results. As an alternative to an APL+Win component file, used in this workspace as an example, a commercial database such as Microsoft SQL Server could be used.
    - When the ‘sync’ processing of all tasks is complete, the ‘controlling application’ accesses the shared data store to consolidate and analyze the processing results.



```
ControllingApp
ControllingAppSync
KernelDyadic
KernelDyadicSync
KernelMonadic
KernelMonadicSync
KernelNiladic
KernelNiladicSync
KernelNoExplicitResult
ProcessAllDoneCallback
ProcessCompleteCallback
ProcessProgressCallback
RandomNotify
StopCloseDeleteSpvr
TieSyncResultsFile
```

Ready

### ControllingApp Function using Async Mode

The 'ControllingApp' function in the prototype workspace illustrates how APL+Win can:

- Create an instance of the APLNext Supervisor
- Configure that instance of the APLNext Supervisor, including setting the 'maxpool' configuration parameter according to the number of processors in the target workstation.
- Subscribe to the 'ProcessCompleteCallback' event which fires when each processing request is satisfied by the 'kernel' function.
- Subscribe to the 'ProcessProgressCallback' event which fires when the 'kernel' function uses the APL+Win 'Notify' method.
- Subscribe to the 'ProcessAllDoneCallback' event which fires when all processing requests have been satisfied. Note that the 'ProcessAllDoneCallback' event should be subscribed by the 'ControllingApp' function only until after all tasks have been queued using the 'BeginCall' method.
- Submit processing requests to the APLNext Supervisor queue using the 'BeginCall' method.

### ProcessCompleteCallback Function

Since processing requests made to the APLNext Supervisor are satisfied asynchronously using multiple threads which the Microsoft Windows operating system assigns to multiple processors on the target workstation, the ‘Controlling App’ function receives notification that a processing request has been satisfied by subscribing to the APLNext Supervisor ‘ProcessCompleteCallBackEvent’.

The ‘ProcessCompleteCallBack’ function in the prototype workspace illustrates:

- Receiving the ⎕wevent information which contains ‘ProcessCompleteCallBackEvent’
- Receiving the ⎕warg information which contains the information about the completed APLNext Supervisor processing request:
    - The ‘processId’ is the integer value which the ‘Controlling App’ provided to the APLNext Supervisor ‘BeginCall’ method when this processing request was submitted to the APLNext Supervisor processing request queue.
    - The ‘errorCode’ is the integer value which the APLNext Supervisor provides when a processing request is satisfied with 0 indicating no error.
    - The ‘resultData’ is the APL+Win variable containing the result of the execution of the ‘kernel’ function, if any. The ‘ProcessCompleteCallback’ function uses this data to augment the application-specific result, e.g. adding it to other processing request results.

## Kernel Functions

Processing requests made to the APLNext Supervisor generally involve repeatedly executing a ‘kernel’ function with different arguments.

The ‘KernelMonadic’ function in the prototype workspace illustrates the function syntax required if a processing request made to the APLNext Supervisor will use a monadic APL+Win function. The APL+Win ⎕dl system function is used to simulate processing work done by a ‘kernel’ function and is not required when a production ‘kernel’ function is used in an APL+Win application system. The ‘KernelDyadic’ and ‘KernelNiladic’ functions are analogous to the ‘KernelMonadic’ function.

The ‘KernelDyadic’ function uses the APL+Win ‘Notify’ event to trigger the APLNext Supervisor ‘ProcessProgressCallback’. This event is useful if information from the domain of the ‘kernel’ function is to be passed to the domain of the ‘controlling application’ before the ‘kernel’ function has completed processing of the task. The ‘KernelDyadic’ function provides the programmer-specified ‘code’ and ‘data’ objects to the domain of the ‘controlling application’ where it is processed by the ‘ProcessProgressCallback’ event handler function..

## ProcessAllDoneCallback Function

When all processing requests have been submitted to the APLNext Supervisor queue and all such queued processing request have been satisfied, the APLNext Supervisor ProcessAllDoneCallback event will fire if it has been subscribed by the ‘ControllingApp’ function.
When the ‘ProcessAllDoneCallback’ event handler function runs, it:

- Prepares the final result of the application system because at this point all the processing requests have been satisfied

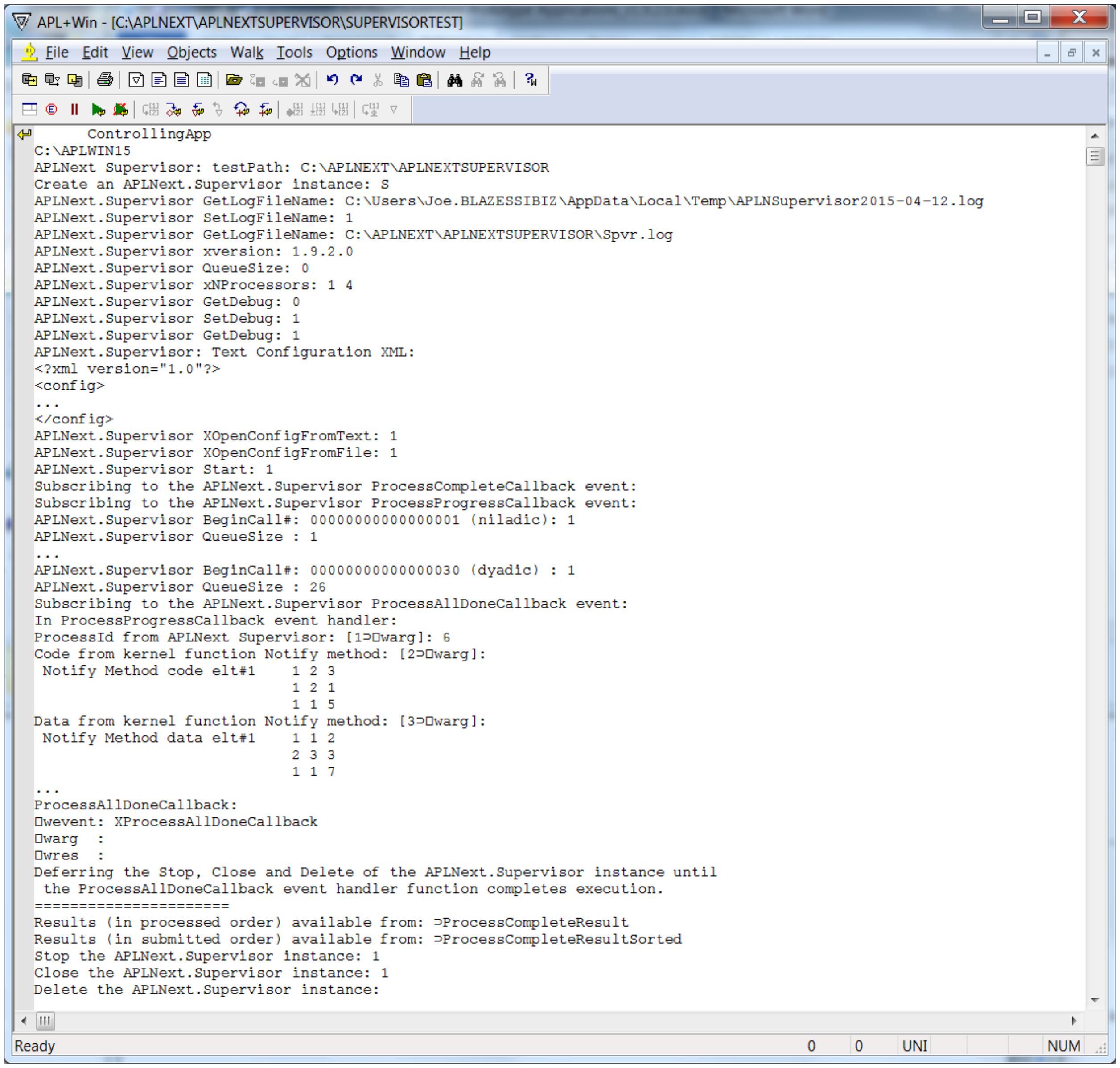
- Stops, closes and deletes the APLNext Supervisor instance which was used by the application system

## Running the ControllingApp Function

To execute the APL+Win 'ControllingApp' function:

- Put the 'SUPERVISORTEST' workspace into an accessible location on the target workstation, e.g. c:\APLNext\APLNextSupervisor\.
- Register APL+Win ActiveX engine (v15.0.1.0 or subsequent) on the target workstation
- Install the APLNext Supervisor .Net component using the 'APLNextSupervisorSetup_v1.9.1.msi' or subsequent version.
- Open the APL+Win developer session
- Load the'SUPERVISORTEST' workspace
- Execute the 'ControllingApp' function

The output from this execution should be similar to the abbreviated output illustrated below:

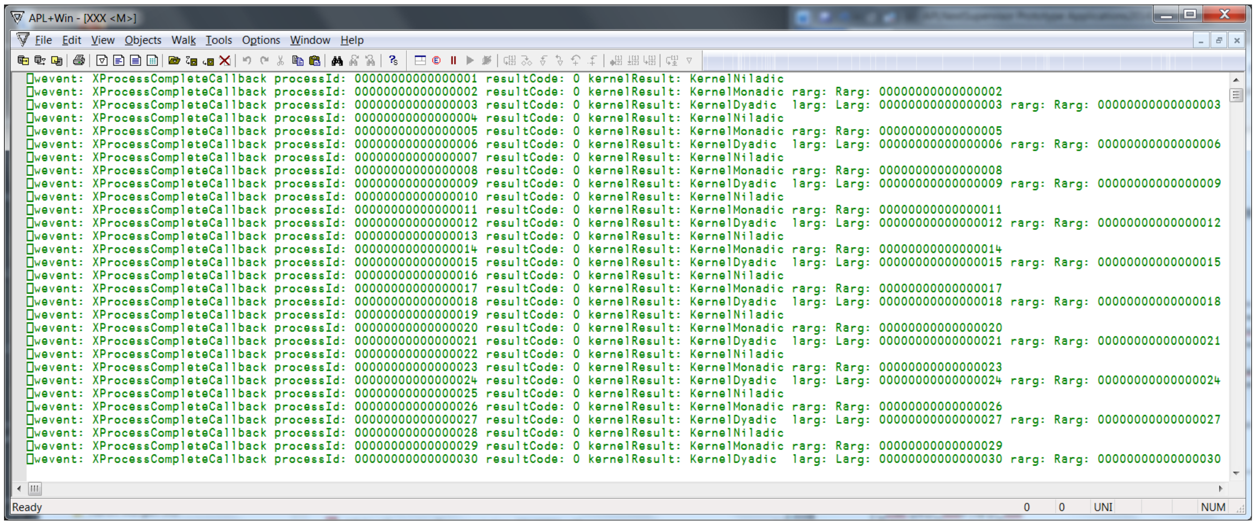

APL+Win - [C:\APLNEXT\APLNEXTSUPERVISOR\SUPERVISORTEST]
File Edit View Objects Walk Tools Options Window Help

```
      ControllingApp
C:\APLWIN15
APLNext Supervisor: testPath: C:\APLNEXT\APLNEXTSUPERVISOR
Create an APLNext.Supervisor instance: S
APLNext.Supervisor GetLogFileName: C:\Users\Joe.BLAZESSIBIZ\AppData\Local\Temp\APLNSupervisor2015-04-12.log
APLNext.Supervisor SetLogFileName: 1
APLNext.Supervisor GetLogFileName: C:\APLNEXT\APLNEXTSUPERVISOR\Spvr.log
APLNext.Supervisor xversion: 1.9.2.0
APLNext.Supervisor QueueSize: 0
APLNext.Supervisor xNProcessors: 1 4
APLNext.Supervisor GetDebug: 0
APLNext.Supervisor SetDebug: 1
APLNext.Supervisor GetDebug: 1
APLNext.Supervisor: Text Configuration XML:
<?xml version="1.0"?>
<config>
...
</config>
APLNext.Supervisor XOpenConfigFromText: 1
APLNext.Supervisor XOpenConfigFromFile: 1
APLNext.Supervisor Start: 1
Subscribing to the APLNext.Supervisor ProcessCompleteCallback event:
Subscribing to the APLNext.Supervisor ProcessProgressCallback event:
APLNext.Supervisor BeginCall#: 0000000000000001 (niladic): 1
APLNext.Supervisor QueueSize : 1
...
APLNext.Supervisor BeginCall#: 0000000000000030 (dyadic) : 1
APLNext.Supervisor QueueSize : 26
Subscribing to the APLNext.Supervisor ProcessAllDoneCallback event:
In ProcessProgressCallback event handler:
ProcessId from APLNext Supervisor: [1⊃⎕warg]: 6
Code from kernel function Notify method: [2⊃⎕warg]:
 Notify Method code elt#1    1 2 3
                             1 2 1
                             1 1 5
Data from kernel function Notify method: [3⊃⎕warg]:
 Notify Method data elt#1    1 1 2
                             2 3 3
                             1 1 7
...
ProcessAllDoneCallback:
⎕wevent: XProcessAllDoneCallback
⎕warg  :
⎕wres  :
Deferring the Stop, Close and Delete of the APLNext.Supervisor instance until
 the ProcessAllDoneCallback event handler function completes execution.
======================
Results (in processed order) available from: ⊃ProcessCompleteResult
Results (in submitted order) available from: ⊃ProcessCompleteResultSorted
Stop the APLNext.Supervisor instance: 1
Close the APLNext.Supervisor instance: 1
Delete the APLNext.Supervisor instance:
```

In the above example the 30 processing requests were submitted to the APLNext Supervisor queue using the 'BeginCall' method in numerical order (#1 – 30), but those processing requests were satisfied and reported to the 'ProcessCompleteCallback' function in the order in which processing was completed, since processing was done asynchronously using independent APL+Win ActiveX engine instances started by the APLNext Supervisor.

The 'ProcessProgressCallback' event handler function displayed data provided by the 'kernel' function via the 'Notify' method.

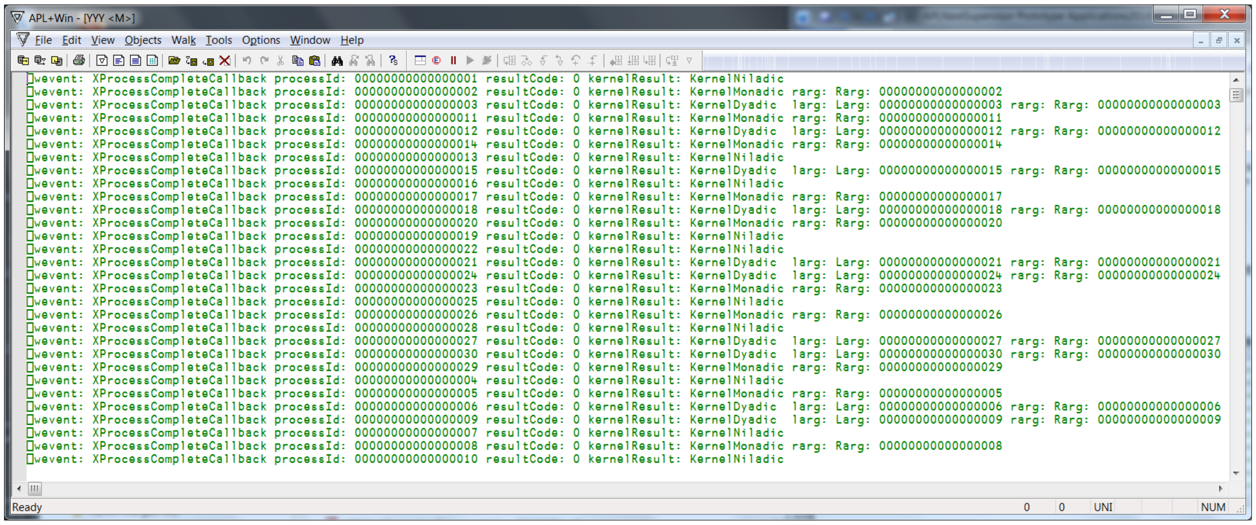
The 'ProcessCompleteSorted variable contains the list of tasks which were processed in the order submitted. Sorting is possible due to the special formatting of the results of the sample 'kernel' function.



APL+Win - [XXX <M>]

```
⎕wevent: XProcessCompleteCallback processId: 0000000000000001 resultCode: 0 kernelResult: KernelNiladic
⎕wevent: XProcessCompleteCallback processId: 0000000000000002 resultCode: 0 kernelResult: KernelMonadic rarg: Rarg: 0000000000000002
⎕wevent: XProcessCompleteCallback processId: 0000000000000003 resultCode: 0 kernelResult: KernelDyadic  larg: Larg: 0000000000000003 rarg: Rarg: 0000000000000003
⎕wevent: XProcessCompleteCallback processId: 0000000000000004 resultCode: 0 kernelResult: KernelNiladic
⎕wevent: XProcessCompleteCallback processId: 0000000000000005 resultCode: 0 kernelResult: KernelMonadic rarg: Rarg: 0000000000000005
⎕wevent: XProcessCompleteCallback processId: 0000000000000006 resultCode: 0 kernelResult: KernelDyadic  larg: Larg: 0000000000000006 rarg: Rarg: 0000000000000006
⎕wevent: XProcessCompleteCallback processId: 0000000000000007 resultCode: 0 kernelResult: KernelNiladic
⎕wevent: XProcessCompleteCallback processId: 0000000000000008 resultCode: 0 kernelResult: KernelMonadic rarg: Rarg: 0000000000000008
⎕wevent: XProcessCompleteCallback processId: 0000000000000009 resultCode: 0 kernelResult: KernelDyadic  larg: Larg: 0000000000000009 rarg: Rarg: 0000000000000009
⎕wevent: XProcessCompleteCallback processId: 0000000000000010 resultCode: 0 kernelResult: KernelNiladic
⎕wevent: XProcessCompleteCallback processId: 0000000000000011 resultCode: 0 kernelResult: KernelMonadic rarg: Rarg: 0000000000000011
⎕wevent: XProcessCompleteCallback processId: 0000000000000012 resultCode: 0 kernelResult: KernelDyadic  larg: Larg: 0000000000000012 rarg: Rarg: 0000000000000012
⎕wevent: XProcessCompleteCallback processId: 0000000000000013 resultCode: 0 kernelResult: KernelNiladic
⎕wevent: XProcessCompleteCallback processId: 0000000000000014 resultCode: 0 kernelResult: KernelMonadic rarg: Rarg: 0000000000000014
⎕wevent: XProcessCompleteCallback processId: 0000000000000015 resultCode: 0 kernelResult: KernelDyadic  larg: Larg: 0000000000000015 rarg: Rarg: 0000000000000015
⎕wevent: XProcessCompleteCallback processId: 0000000000000016 resultCode: 0 kernelResult: KernelNiladic
⎕wevent: XProcessCompleteCallback processId: 0000000000000017 resultCode: 0 kernelResult: KernelMonadic rarg: Rarg: 0000000000000017
⎕wevent: XProcessCompleteCallback processId: 0000000000000018 resultCode: 0 kernelResult: KernelDyadic  larg: Larg: 0000000000000018 rarg: Rarg: 0000000000000018
⎕wevent: XProcessCompleteCallback processId: 0000000000000019 resultCode: 0 kernelResult: KernelNiladic
⎕wevent: XProcessCompleteCallback processId: 0000000000000020 resultCode: 0 kernelResult: KernelMonadic rarg: Rarg: 0000000000000020
⎕wevent: XProcessCompleteCallback processId: 0000000000000021 resultCode: 0 kernelResult: KernelDyadic  larg: Larg: 0000000000000021 rarg: Rarg: 0000000000000021
⎕wevent: XProcessCompleteCallback processId: 0000000000000022 resultCode: 0 kernelResult: KernelNiladic
⎕wevent: XProcessCompleteCallback processId: 0000000000000023 resultCode: 0 kernelResult: KernelMonadic rarg: Rarg: 0000000000000023
⎕wevent: XProcessCompleteCallback processId: 0000000000000024 resultCode: 0 kernelResult: KernelDyadic  larg: Larg: 0000000000000024 rarg: Rarg: 0000000000000024
⎕wevent: XProcessCompleteCallback processId: 0000000000000025 resultCode: 0 kernelResult: KernelNiladic
⎕wevent: XProcessCompleteCallback processId: 0000000000000026 resultCode: 0 kernelResult: KernelMonadic rarg: Rarg: 0000000000000026
⎕wevent: XProcessCompleteCallback processId: 0000000000000027 resultCode: 0 kernelResult: KernelDyadic  larg: Larg: 0000000000000027 rarg: Rarg: 0000000000000027
⎕wevent: XProcessCompleteCallback processId: 0000000000000028 resultCode: 0 kernelResult: KernelNiladic
⎕wevent: XProcessCompleteCallback processId: 0000000000000029 resultCode: 0 kernelResult: KernelMonadic rarg: Rarg: 0000000000000029
⎕wevent: XProcessCompleteCallback processId: 0000000000000030 resultCode: 0 kernelResult: KernelDyadic  larg: Larg: 0000000000000030 rarg: Rarg: 0000000000000030
```

The 'ProcessComplete' variable contains the list of tasks where were processed in the order that their processing was completed.



APL+Win - [YYY <M>]

```
⎕wevent: XProcessCompleteCallback processId: 0000000000000001 resultCode: 0 kernelResult: KernelNiladic
⎕wevent: XProcessCompleteCallback processId: 0000000000000002 resultCode: 0 kernelResult: KernelMonadic rarg: Rarg: 0000000000000002
⎕wevent: XProcessCompleteCallback processId: 0000000000000003 resultCode: 0 kernelResult: KernelDyadic  larg: Larg: 0000000000000003 rarg: Rarg: 0000000000000003
⎕wevent: XProcessCompleteCallback processId: 0000000000000011 resultCode: 0 kernelResult: KernelMonadic rarg: Rarg: 0000000000000011
⎕wevent: XProcessCompleteCallback processId: 0000000000000012 resultCode: 0 kernelResult: KernelDyadic  larg: Larg: 0000000000000012 rarg: Rarg: 0000000000000012
⎕wevent: XProcessCompleteCallback processId: 0000000000000014 resultCode: 0 kernelResult: KernelMonadic rarg: Rarg: 0000000000000014
⎕wevent: XProcessCompleteCallback processId: 0000000000000013 resultCode: 0 kernelResult: KernelNiladic
⎕wevent: XProcessCompleteCallback processId: 0000000000000015 resultCode: 0 kernelResult: KernelDyadic  larg: Larg: 0000000000000015 rarg: Rarg: 0000000000000015
⎕wevent: XProcessCompleteCallback processId: 0000000000000016 resultCode: 0 kernelResult: KernelNiladic
⎕wevent: XProcessCompleteCallback processId: 0000000000000017 resultCode: 0 kernelResult: KernelMonadic rarg: Rarg: 0000000000000017
⎕wevent: XProcessCompleteCallback processId: 0000000000000018 resultCode: 0 kernelResult: KernelDyadic  larg: Larg: 0000000000000018 rarg: Rarg: 0000000000000018
⎕wevent: XProcessCompleteCallback processId: 0000000000000020 resultCode: 0 kernelResult: KernelMonadic rarg: Rarg: 0000000000000020
⎕wevent: XProcessCompleteCallback processId: 0000000000000019 resultCode: 0 kernelResult: KernelNiladic
⎕wevent: XProcessCompleteCallback processId: 0000000000000022 resultCode: 0 kernelResult: KernelNiladic
⎕wevent: XProcessCompleteCallback processId: 0000000000000021 resultCode: 0 kernelResult: KernelDyadic  larg: Larg: 0000000000000021 rarg: Rarg: 0000000000000021
⎕wevent: XProcessCompleteCallback processId: 0000000000000024 resultCode: 0 kernelResult: KernelDyadic  larg: Larg: 0000000000000024 rarg: Rarg: 0000000000000024
⎕wevent: XProcessCompleteCallback processId: 0000000000000023 resultCode: 0 kernelResult: KernelMonadic rarg: Rarg: 0000000000000023
⎕wevent: XProcessCompleteCallback processId: 0000000000000025 resultCode: 0 kernelResult: KernelNiladic
⎕wevent: XProcessCompleteCallback processId: 0000000000000026 resultCode: 0 kernelResult: KernelMonadic rarg: Rarg: 0000000000000026
⎕wevent: XProcessCompleteCallback processId: 0000000000000028 resultCode: 0 kernelResult: KernelNiladic
⎕wevent: XProcessCompleteCallback processId: 0000000000000027 resultCode: 0 kernelResult: KernelDyadic  larg: Larg: 0000000000000027 rarg: Rarg: 0000000000000027
⎕wevent: XProcessCompleteCallback processId: 0000000000000030 resultCode: 0 kernelResult: KernelDyadic  larg: Larg: 0000000000000030 rarg: Rarg: 0000000000000030
⎕wevent: XProcessCompleteCallback processId: 0000000000000029 resultCode: 0 kernelResult: KernelMonadic rarg: Rarg: 0000000000000029
⎕wevent: XProcessCompleteCallback processId: 0000000000000004 resultCode: 0 kernelResult: KernelNiladic
⎕wevent: XProcessCompleteCallback processId: 0000000000000005 resultCode: 0 kernelResult: KernelMonadic rarg: Rarg: 0000000000000005
⎕wevent: XProcessCompleteCallback processId: 0000000000000006 resultCode: 0 kernelResult: KernelDyadic  larg: Larg: 0000000000000006 rarg: Rarg: 0000000000000006
⎕wevent: XProcessCompleteCallback processId: 0000000000000009 resultCode: 0 kernelResult: KernelDyadic  larg: Larg: 0000000000000009 rarg: Rarg: 0000000000000009
⎕wevent: XProcessCompleteCallback processId: 0000000000000007 resultCode: 0 kernelResult: KernelNiladic
⎕wevent: XProcessCompleteCallback processId: 0000000000000008 resultCode: 0 kernelResult: KernelMonadic rarg: Rarg: 0000000000000008
⎕wevent: XProcessCompleteCallback processId: 0000000000000010 resultCode: 0 kernelResult: KernelNiladic
```

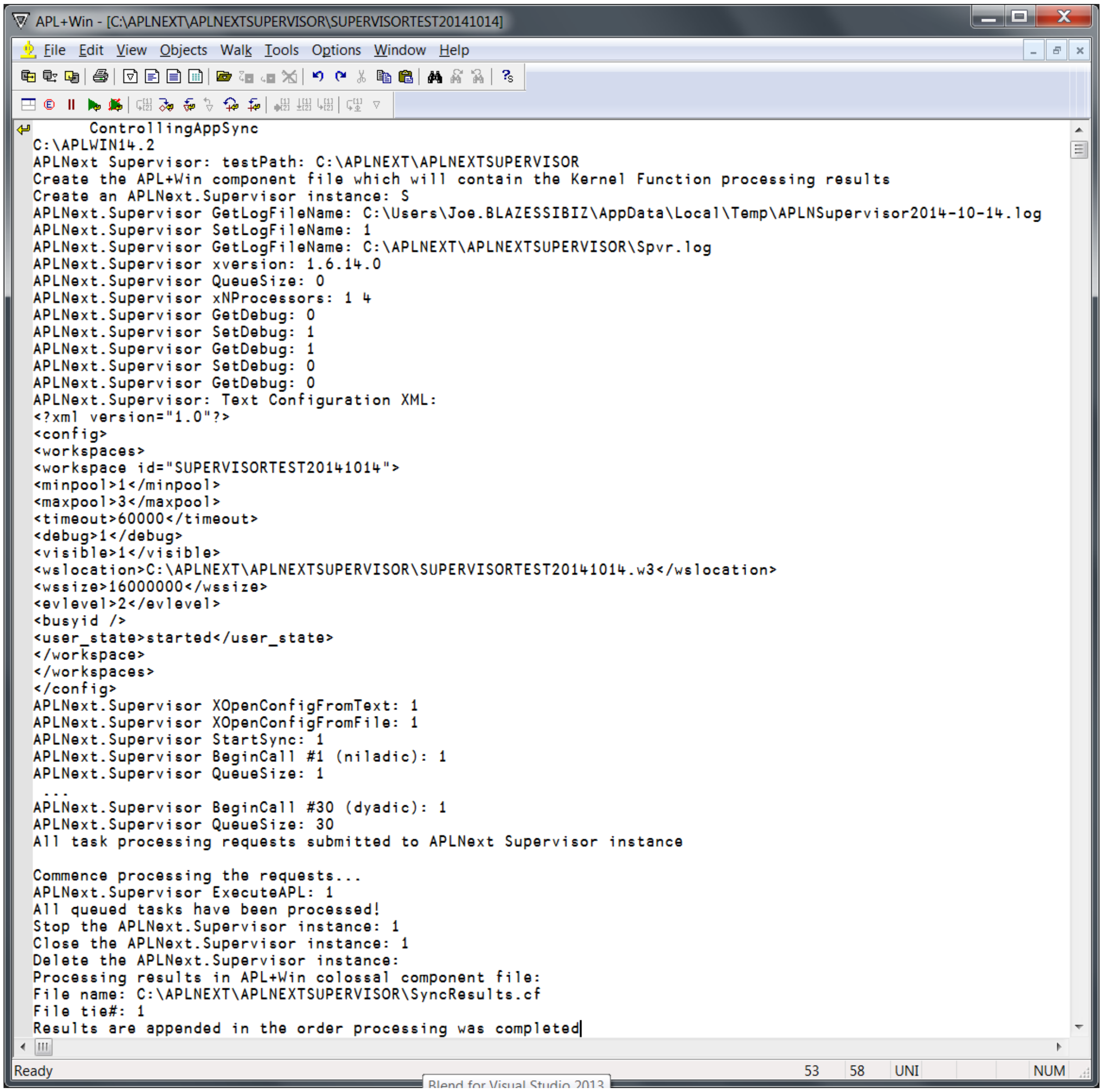
## ControllingAppSync Function using Sync Mode

The workspace contains the ‘ControllingAppSync’ function which uses the APLNext Supervisor in sync mode.  The APLNext Supervisor ‘StartSync’ method is used instead of the ‘Start’ method.  This means that the processing of the tasks submitted by the ‘ControllingAppSync’ function using the ‘BeginCall’ method will not commence until the ‘ControllingAppSync’ function uses the ‘ExecuteAPL’ method.

Since in sync mode the APLNext Supervisor events do not fire, the kernel function(s), KernelDyadicSync’, ‘KernelMonadicSync’ and ‘KernelNiladicSync’, save their task processing results to a shared repository, in this case an APL+Win colossal component file.

The execution of the ‘ControllingAppSync’ function is suspended after the ‘ExecuteAPL’ method is used and resumes after all the queued tasks have been processed.

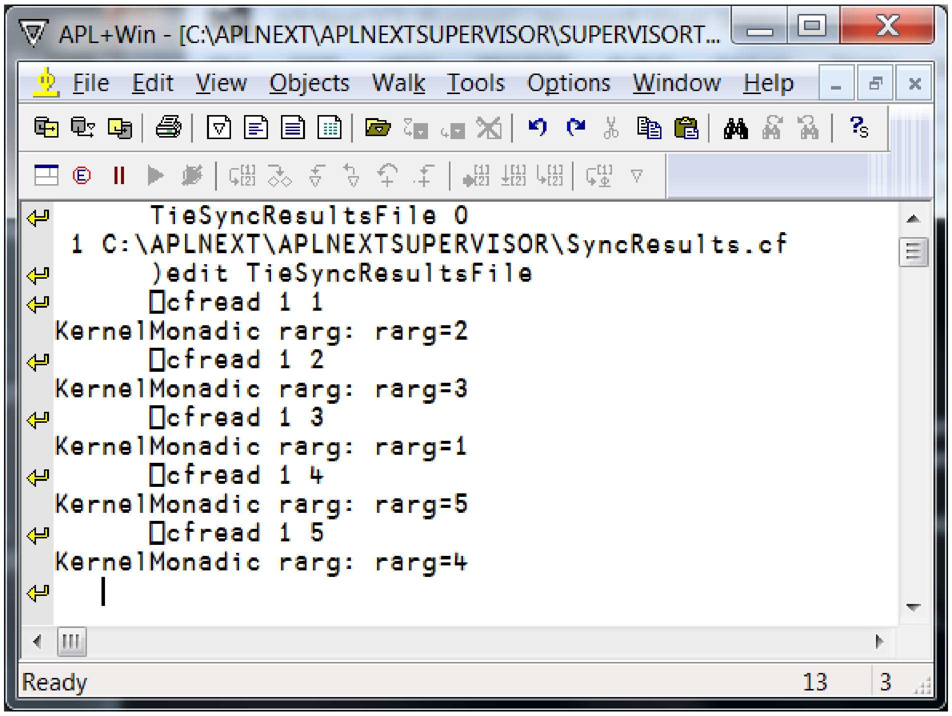
When the 'ControllingAppSync' function is run the output should be similar to:


APL+Win - [C:\APLNEXT\APLNEXTSUPERVISOR\SUPERVISORTEST20141014]
File Edit View Objects Walk Tools Options Window Help
Ready 53 58 UNI NUM

```
      ControllingAppSync
C:\APLWIN14.2
APLNext Supervisor: testPath: C:\APLNEXT\APLNEXTSUPERVISOR
Create the APL+Win component file which will contain the Kernel Function processing results
Create an APLNext.Supervisor instance: S
APLNext.Supervisor GetLogFileName: C:\Users\Joe.BLAZESSIBIZ\AppData\Local\Temp\APLNSupervisor2014-10-14.log
APLNext.Supervisor SetLogFileName: 1
APLNext.Supervisor GetLogFileName: C:\APLNEXT\APLNEXTSUPERVISOR\Spvr.log
APLNext.Supervisor xversion: 1.6.14.0
APLNext.Supervisor QueueSize: 0
APLNext.Supervisor xNProcessors: 1 4
APLNext.Supervisor GetDebug: 0
APLNext.Supervisor SetDebug: 1
APLNext.Supervisor GetDebug: 1
APLNext.Supervisor SetDebug: 0
APLNext.Supervisor GetDebug: 0
APLNext.Supervisor: Text Configuration XML:
<?xml version="1.0"?>
<config>
<workspaces>
<workspace id="SUPERVISORTEST20141014">
<minpool>1</minpool>
<maxpool>3</maxpool>
<timeout>60000</timeout>
<debug>1</debug>
<visible>1</visible>
<wslocation>C:\APLNEXT\APLNEXTSUPERVISOR\SUPERVISORTEST20141014.w3</wslocation>
<wssize>16000000</wssize>
<evlevel>2</evlevel>
<busyid />
<user_state>started</user_state>
</workspace>
</workspaces>
</config>
APLNext.Supervisor XOpenConfigFromText: 1
APLNext.Supervisor XOpenConfigFromFile: 1
APLNext.Supervisor StartSync: 1
APLNext.Supervisor BeginCall #1 (niladic): 1
APLNext.Supervisor QueueSize: 1
 ...
APLNext.Supervisor BeginCall #30 (dyadic): 1
APLNext.Supervisor QueueSize: 30
All task processing requests submitted to APLNext Supervisor instance

Commence processing the requests...
APLNext.Supervisor ExecuteAPL: 1
All queued tasks have been processed!
Stop the APLNext.Supervisor instance: 1
Close the APLNext.Supervisor instance: 1
Delete the APLNext.Supervisor instance:
Processing results in APL+Win colossal component file:
File name: C:\APLNEXT\APLNEXTSUPERVISOR\SyncResults.cf
File tie#: 1
Results are appended in the order processing was completed
```

After all the queued tasks are processed the contents of the shared result repository, i.e. an APL+Win colossal component file, can be examined to see the results of the processing:


APL+Win - [C:\APLNEXT\APLNEXTSUPERVISOR\SUPERVISORT...
File Edit View Objects Walk Tools Options Window Help

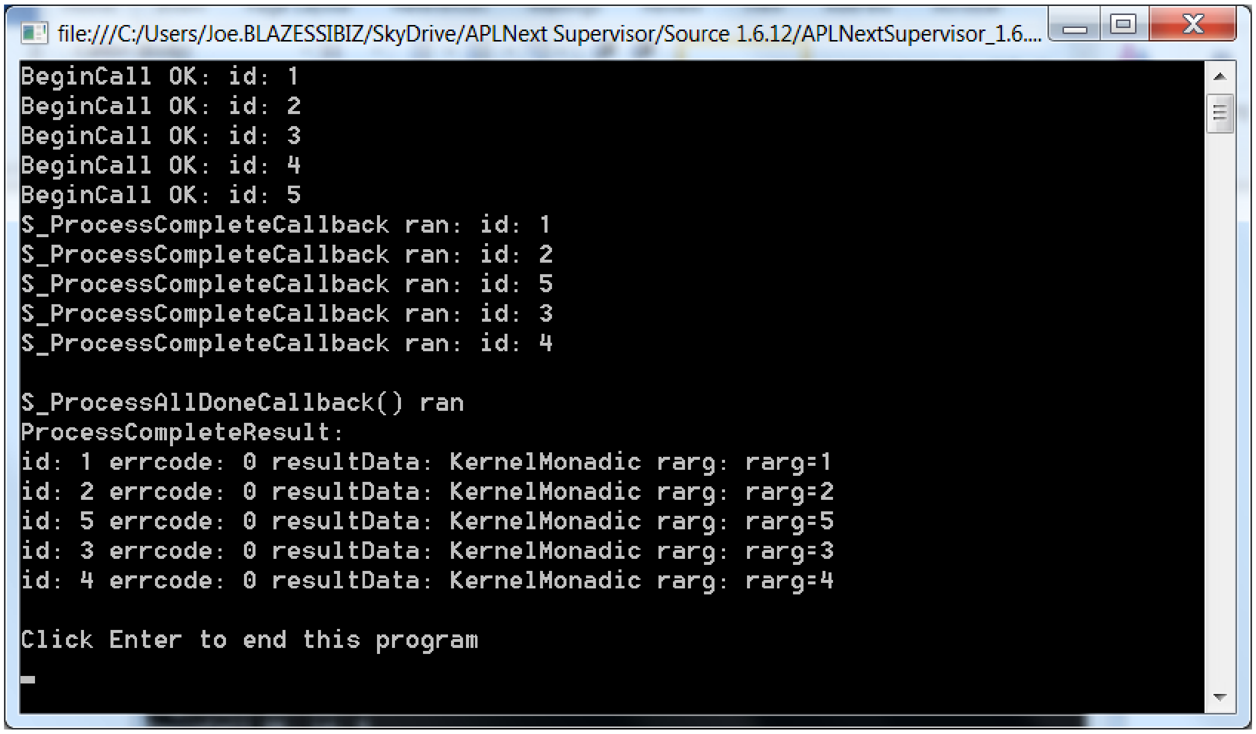
```
      TieSyncResultsFile 0
 1 C:\APLNEXT\APLNEXTSUPERVISOR\SyncResults.cf
      )edit TieSyncResultsFile
      ⎕cfread 1 1
KernelMonadic rarg: rarg=2
      ⎕cfread 1 2
KernelMonadic rarg: rarg=3
      ⎕cfread 1 3
KernelMonadic rarg: rarg=1
      ⎕cfread 1 4
KernelMonadic rarg: rarg=5
      ⎕cfread 1 5
KernelMonadic rarg: rarg=4
```

## C# as the Controlling App in Async Mode

The Visual Studio 2013 ‘APLNext.SCS’ solution contains the ‘APLNext.SCS’ project which illustrates how C# can be used as the ‘ControllingApp’ in ‘async’ mode. In this scenario:

- The ‘ControllingApp’ is the C# Main() method in the ‘APLNext.SCA’ console project in the solution
- The event handler for the APLNext Supervisor ‘ProcessCompleteCallback’ event is the C# S_ProcessCompleteCallback() method
- The event handler for the APLNext Supervisor ‘ProcessAllCompleteCallback’ event is the C# S_ProcessAllDoneCallback() method
- The APL+Win ‘SUPERVISORTEST’ workspace, assumed to be in the ‘c:\APLNext\APLNextSupervisor\ folder on the target workstation, contains the APL+Win ‘kernel’ function, ‘KernelMonadic’, which will be used to satisfy the processing requests submitted to the APLNext Supervisor instance by the C# ‘Controlling App’ using the ‘BeginCall’ method.
- The C# Main() method makes the instances of the APL+Win ActiveX engine visible while the project is running. This illustrates that even though C# is the ‘Controlling App’, it is possible to debug the APL+Win ‘kernel’ function from with a familiar APL+Win developer session.

When the ‘APLNext.SCA’ project is run the output should be similar to:


file:///C:/Users/Joe.BLAZESSIBIZ/SkyDrive/APLNext Supervisor/Source 1.6.12/APLNextSupervisor_1.6....

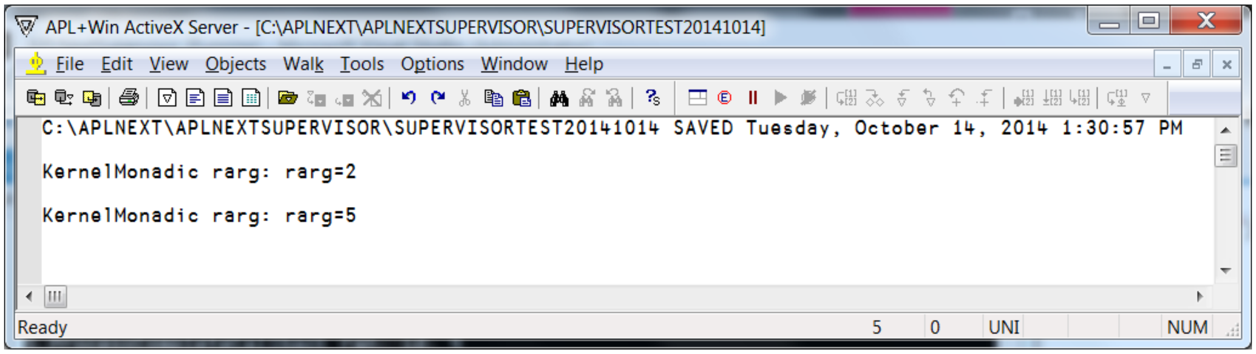
```
BeginCall OK: id: 1
BeginCall OK: id: 2
BeginCall OK: id: 3
BeginCall OK: id: 4
BeginCall OK: id: 5
S_ProcessCompleteCallback ran: id: 1
S_ProcessCompleteCallback ran: id: 2
S_ProcessCompleteCallback ran: id: 5
S_ProcessCompleteCallback ran: id: 3
S_ProcessCompleteCallback ran: id: 4

S_ProcessAllDoneCallback() ran
ProcessCompleteResult:
id: 1 errcode: 0 resultData: KernelMonadic rarg: rarg=1
id: 2 errcode: 0 resultData: KernelMonadic rarg: rarg=2
id: 5 errcode: 0 resultData: KernelMonadic rarg: rarg=5
id: 3 errcode: 0 resultData: KernelMonadic rarg: rarg=3
id: 4 errcode: 0 resultData: KernelMonadic rarg: rarg=4

Click Enter to end this program
_
```

When all the queued tasks are processed the ‘S_ProcessAllDoneCallback()’ method runs and displays the accumulated results of the processing of the tasks.

During the processing the queued tasks, instances of APL+Win ActiveX Engine will be visible. For a production deployment, these can be hidden and the runtime version of APL+Win would be used. For example:

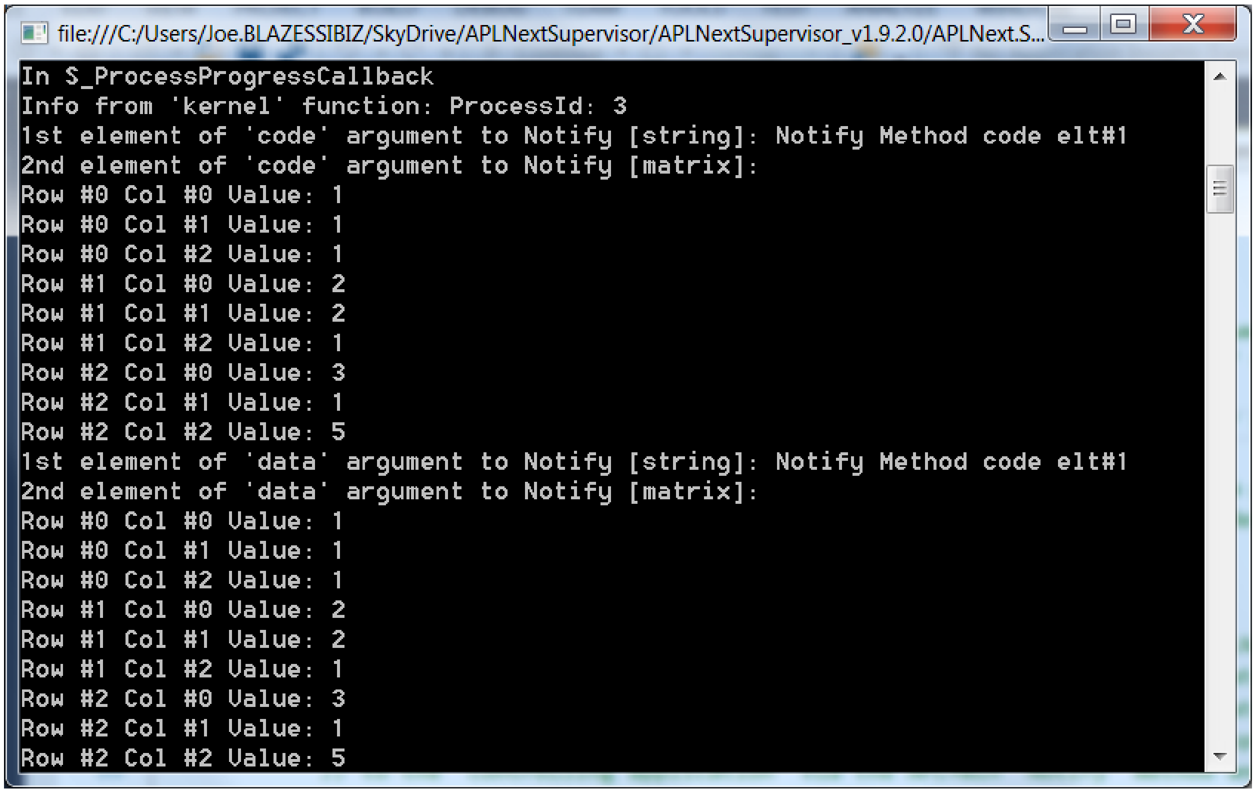

APL+Win ActiveX Server - [C:\APLNEXT\APLNEXTSUPERVISOR\SUPERVISORTEST20141014]
File Edit View Objects Walk Tools Options Window Help

```
C:\APLNEXT\APLNEXTSUPERVISOR\SUPERVISORTEST20141014 SAVED Tuesday, October 14, 2014 1:30:57 PM

KernelMonadic rarg: rarg=2

KernelMonadic rarg: rarg=5
```

When the ProcessProgressCallback event fires information from the 'kernel' function is displayed:


file:///C:/Users/Joe.BLAZESSIBIZ/SkyDrive/APLNextSupervisor/APLNextSupervisor_v1.9.2.0/APLNext.S...

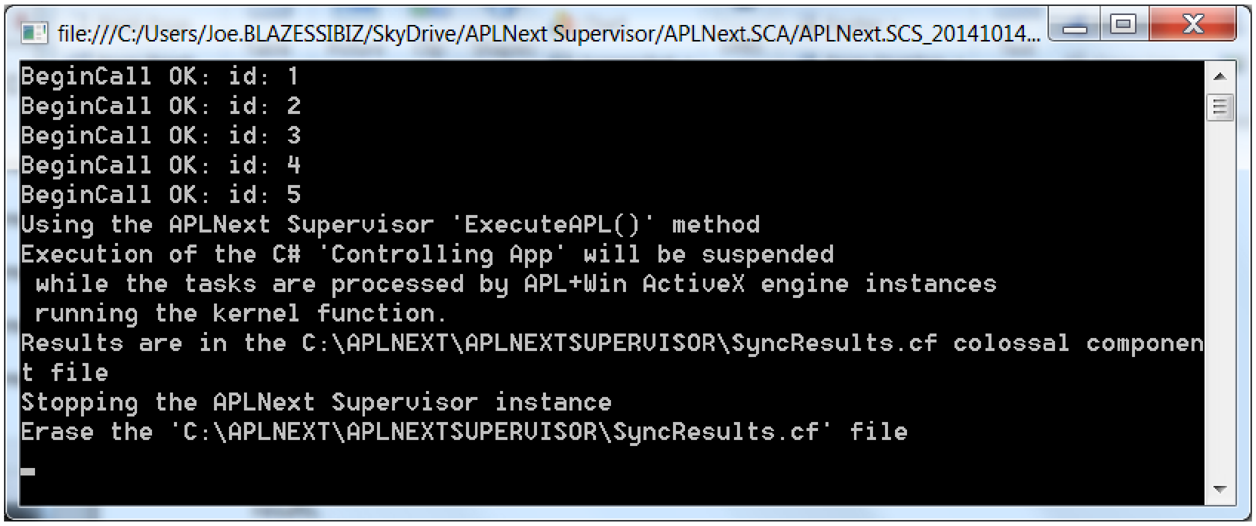
```
In S_ProcessProgressCallback
Info from 'kernel' function: ProcessId: 3
1st element of 'code' argument to Notify [string]: Notify Method code elt#1
2nd element of 'code' argument to Notify [matrix]:
Row #0 Col #0 Value: 1
Row #0 Col #1 Value: 1
Row #0 Col #2 Value: 1
Row #1 Col #0 Value: 2
Row #1 Col #1 Value: 2
Row #1 Col #2 Value: 1
Row #2 Col #0 Value: 3
Row #2 Col #1 Value: 1
Row #2 Col #2 Value: 5
1st element of 'data' argument to Notify [string]: Notify Method code elt#1
2nd element of 'data' argument to Notify [matrix]:
Row #0 Col #0 Value: 1
Row #0 Col #1 Value: 1
Row #0 Col #2 Value: 1
Row #1 Col #0 Value: 2
Row #1 Col #1 Value: 2
Row #1 Col #2 Value: 1
Row #2 Col #0 Value: 3
Row #2 Col #1 Value: 1
Row #2 Col #2 Value: 5
```

## C# as the Controlling App in Sync Mode

The 'APLNext.SCS' solution contains the 'APLNext.SCSsync' project which illustrates a C# 'Controlling App' method which uses the APLNext Supervisor in sync mode.

The APLNext Supervisor events are not used in 'sync' mode, i.e. the 'Controlling App' using the APLNext Supervisor 'StartSync' and 'ExecuteAPL' methods.   Instead the execution of the 'Controlling App' is suspended by the APLNextSupervisor after the 'ExecuteAPL' method is used until all the queued tasks have been processed.  In this case the processing results are saved by the kernel function to a shared result repository.  In this example an APL+Win colossal component file is used, however a commercial database such as Microsoft SQL Server can also be used as the shared repository for kernel function results.

When the 'APLNext.SCSsync' project is run the output should be similar to:


file:///C:/Users/Joe.BLAZESSIBIZ/SkyDrive/APLNext Supervisor/APLNext.SCA/APLNext.SCS_20141014...

```
BeginCall OK: id: 1
BeginCall OK: id: 2
BeginCall OK: id: 3
BeginCall OK: id: 4
BeginCall OK: id: 5
Using the APLNext Supervisor 'ExecuteAPL()' method
Execution of the C# 'Controlling App' will be suspended
 while the tasks are processed by APL+Win ActiveX engine instances
 running the kernel function.
Results are in the C:\APLNEXT\APLNEXTSUPERVISOR\SyncResults.cf colossal componen
t file
Stopping the APLNext Supervisor instance
Erase the 'C:\APLNEXT\APLNEXTSUPERVISOR\SyncResults.cf' file
```