Using ⎕EditEvents and ⎕EditInfo

Contents

Overview

Events associated with function and variable object editors in the APL64 developer version can be subscribed with APL programmer-defined event handlers.

When a specified edit event is subscribed, the execution of the event handler is treated as an APL64 event callback.

The event handler can be an APL64 user-defined function or an APL64 executable expression.

This feature is available only in the APL64 developer version, because the APL64 object editor is available only in the APL64 developer version.

⎕editevents Configure ⎕edit &)Edit events

Purpose: Subscribe/Unsubscribe an event handler to an object editor event.

Syntax:

⎕editevents "?" or ⎕editevents "Help" return summary documentation character matrix.

subscribe ⎕editevents eventName ehText [ehType]

Arguments:
subscribe is a Boolean scalar used to indicate if the specified event type should be 1/subscribed or 0[default]/unsubscribed.

The same event may be subscribed to the same or different event handler multiple times.

The unsubscribe action has no effect if the specified event is not currently subscribed to an event handler. The unsubscribe action will remove the most recent event subscription, if any, which matches the specified eventName, ehText and ehType.

eventName is non-case sensitive text containing the ⎕edit events type associated with the APL64 object editor. See the documentation below for available event types exposed by ⎕edit and)edit.

ehText is case-sensitive text. Depending on the value of the ehType argument it is:

- The name of the user-defined event handler function that will be run when eventName fires, or
- The text of an APL executable statement that will be run when eventName fires.

ehType is non-case-sensitive text indicating the type of event handler:

The default value is ‘function’ which is the default value when this optional argument is not present. If ehText is the name of an APL programmer-provided event handler function, the ehType argument is not required, but can be 'function' or 'F '.

If ehText is the text of an APL executable statement, the ehType argument should be 'Statement' or 'S'.

Result, if any, depends on the eventName.

Events exposed by]edit and)edit

EditEnded

This event fires after the editor editing the object closes.
eventName: 'EditEnded'
ehText: The EditEnded event handler can be a user-defined APL function or APL executable statement.

- If ehText is the name of a user-defined APL function:
  - It should have no result
  - It should have no left argument
  - It must have a right argument which will be set to the name of object being saved when the event handler function runs.
- If ehText is the text of a user-defined APL executable statement:
  - It should have no result
  - To obtain the name of the object being edited, use the]EditInfo "GetObjName" method from within the scope of the APL executable statement.

EditSaved

This event fires after the edited object have been saved to the current workspace
EndEdit event handler:
eventName: 'EditSaved'
ehText: The EditSaved event handler can be a user-defined APL function or APL executable statement.

- If ehText is the name of a user-defined APL function
  - It should have no result
  - It should have no left argument
  - It must have a right argument which will be set to the name of the object being saved when the event handler is run.
- If ehText is the text of a user-defined APL executable statement
  - It should have no result
  - To obtain the name of the object saved, use the]EditInfo "GetObjName" method from within the scope of the APL executable statement.

EditStarted

This event fires after the name of the object to be edited has been determined. This event does not fire if an object name is selected in the Fetch dialog.
eventName: 'EditStarted'
ehText: The EditStarted event handler can be a user-defined APL function or an APL executable statement.

- If ehText is the name of a user-defined APL function

    - It should have no result
    - It should have no left argument
    - It must have a right argument which will be set to the name of the object editor started when the event handler is run.
- If ehText is the text of a user-defined APL executable statement
    - It should have no result
    - To obtain the name of the object editor started, use the ⎕EditInfo "GetObjName" method from within the scope of the APL executable statement.

ObjectNameSelected

This event fires after an existing object name have been selected in the Open or Fetch dialog. This event will fire before the EditStarted event, if that event is subscribed.

eventName: 'ObjectNameSelected'
ehText: The ObjectNameSelected event handler can be a user-defined APL function or an APL executable statement.

- If ehText is the name of a user-defined APL function
    - It should have no result
    - It should have no left argument
    - If must have a right argument which will be set to the name of object selected when the event handler is run.
- If ehText is the text of a user-defined APL executable statement
    - It should have no result
    - To obtain the name of the object selected, use the ⎕EditInfo "GetObjName" method from within the scope of the APL executable statement.

PreviewObjectNameSelection

This event fires immediately before the object selector dialog is activated for user interaction. The object selector dialog is activated:

- When the Objects | Open or Objects | Fetch menu is used
- When the Ctrl+O or Ctrl+G shortcut is used
- When a visible object selector dialog regains the keyboard focus after losing the keyboard focus

The purpose of a PreviewObjectNameSelection handler is to instantiate or delete APL64 variables or user-defined functions in the current workspace. When the PreviewObjectNameSelection handler is executed, the specified objects are instantiated in the current workspace and the object selector dialog will display those object names and type along with any previously-existing workspace objects.

eventName: 'PreviewObjectNameSelection'
ehText: The PreviewObjectNameSelection event handler can be an APL64 user-defined function or an APL executable statement:

- If ehText is the name of a user-defined APL function
    - It must have no result
    - It must no left argument

    - It must have a right argument and currently the right argument is reserved for future use
    - It must not produce any output to the history
- If ehText is the text of a user-defined APL executable statement
    - It must not produce any output to the history

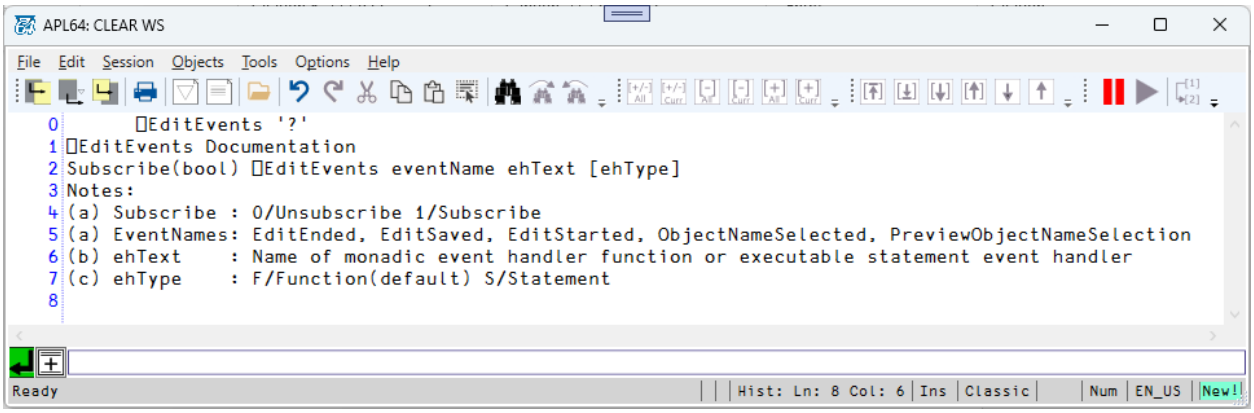
Because the PreviewObjectNameSelection event handler may be executed several times during an APL64 developer instance, the handler should carefully consider if a previously-existing workspace object should be overwritten by the handler's action.

APL64 Object Editors are Independent of the APL+Win ⎕wi Interface

The previous APL+Win ⎕wi-based editor events to not apply to APL64, because the APL+Win developer version was Win32 based.

APL64 Editor Events replacing APL+Win Editor Events	
APL64 Editor Events	**APL+Win Editor Events**
EditEnded	onEditEnd
EditSaved	onEditSaved
EditStarted	onEditLoad
ObjectNameSelected	onEditLoad
⎕Edit system function	onEditStart
PreviewObjectNameSelection	onEditNL

⎕EditEvents Examples



In a production environment, the event handler function or APL statement would perform some useful work which is designed to be completed when the subscribed event fires.

The APL64 programmer may utilize the APL64 ⎕wi interface to create a form-base GUI incorporating a ⎕wi Wait method to present when an editor event fires in an APL64 developer version instance which is subscribed by ⎕editevents.

These examples may use multi-line APL64 executable statements.

Some of these examples cause an exception to be thrown, to illustrate that the APL programmer-provided event handler function or APL statement is executed when the subscribed event fires.

Multiple Event Handlers for the Same Edit Event

The same event may be subscribed to the same or different event handler multiple times. For example, for the EditStarted event, suppose these are the subscribed APL64 programmer-defined functions:

```
1 ⎕EditEvents 'EditStarted' 'EditStartedEh1' 'F'
1 ⎕EditEvents 'EditStarted' 'EditStartedEh2' 'F'
1 ⎕EditEvents 'EditStarted' 'EditStartedEh1' 'F'
```

When an APL64 developer version editor is started:

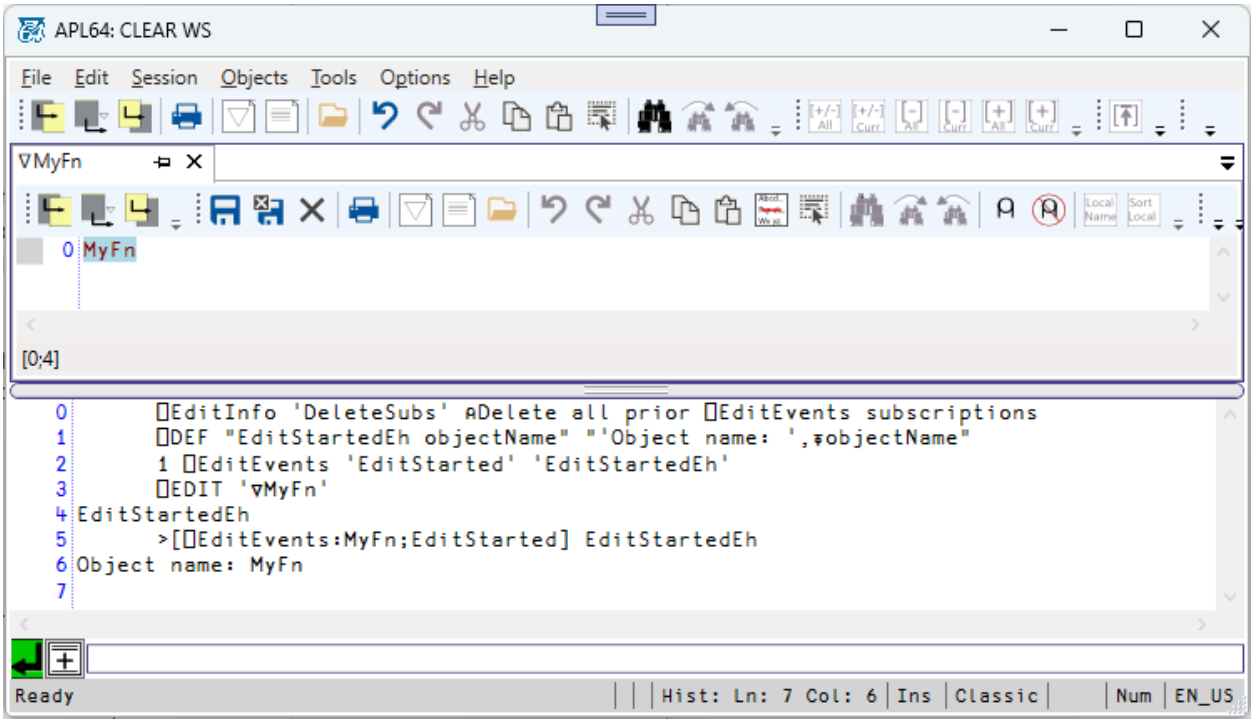
- The EditStartedEh1 function will be executed
- The EditStartedEh2 function will be executed
- The EditStartedEh1 function will be executed again
- The APL64 editor dialog will become active for APL programmer input

Note that subscribing a APL64 programmer-defined function to an Edit Event adds that subscription to the ⎕EditEvents sequence of actions and does not delete any previously-subscribed APL64 programmer-defined functions to that Edit Event. Use the unsubscribe option to ⎕EditEvents or the ⎕EditInfo 'DeleteSubs' action to remove previously-subscribed events.

EditStarted Example: APL Function Event Handler

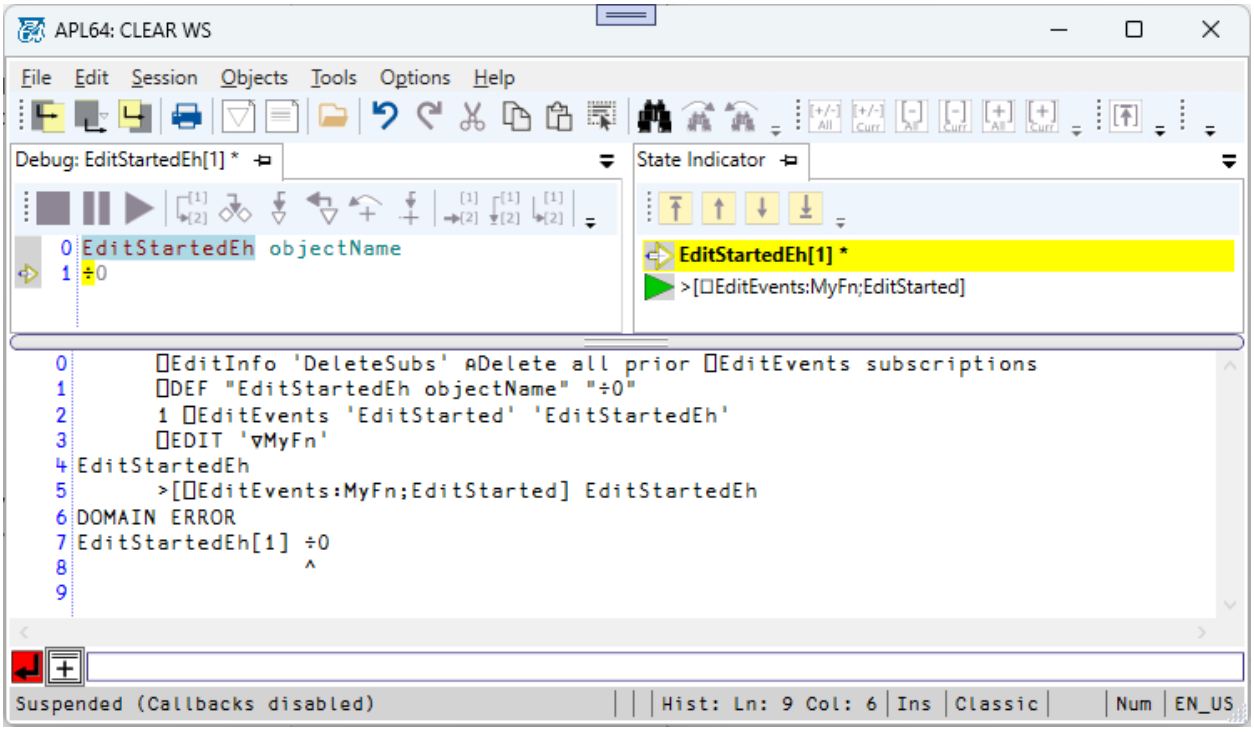
No exception in the APL64 user-defined function. The event handler function will complete its processing and the specified object will be opened in the editor:

```
⎕EditInfo 'DeleteSubs' ⍝Delete all prior ⎕EditEvents subscriptions
←⎕DEF "EditStartedEh objectName" "'Object name: ', ⍕ objectName"
1 ⎕EditEvents 'EditStarted' 'EditStartedEh'
⎕EDIT '∇MyFn'
```



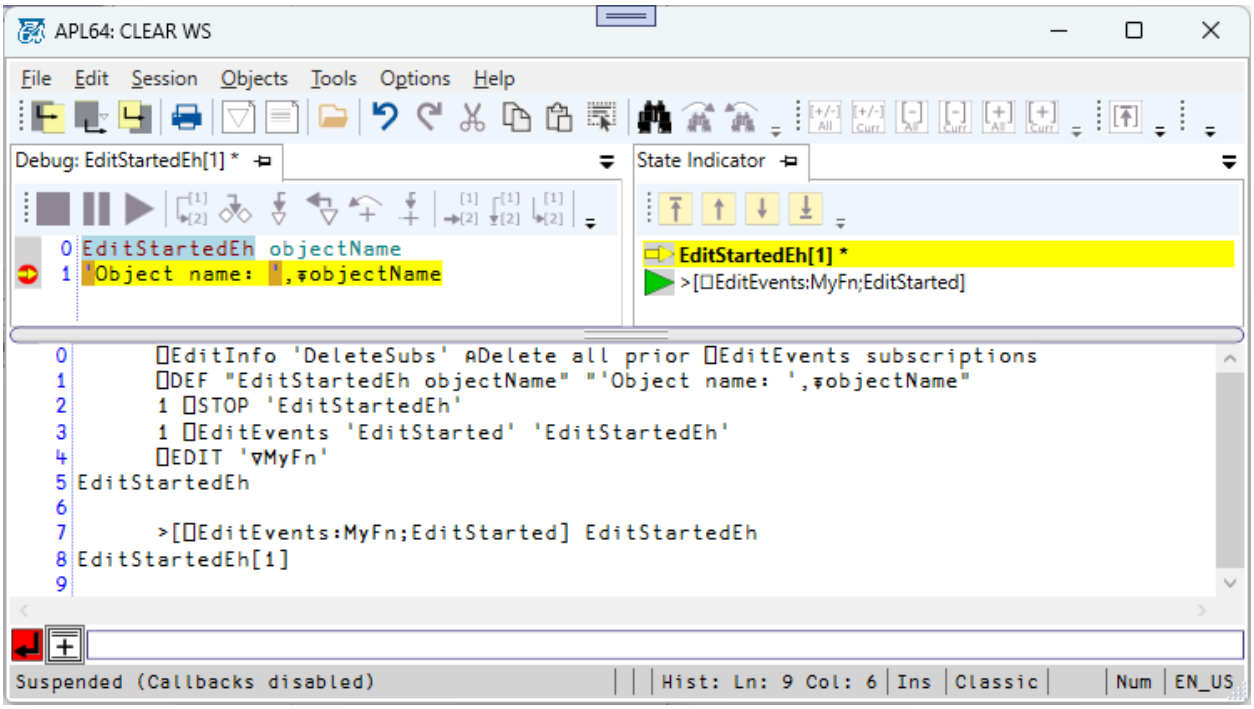
Exception in the APL64 user-defined function. The EditStartedEh function will be suspended in the debugger due to the exception, the event handler function will be the name of the function being edited and the object editor will not be presented:

```
⎕EditInfo 'DeleteSubs' ⍝Delete all prior ⎕EditEvents subscriptions
←⎕DEF "EditStartedEh objectName" "÷0"
1 ⎕EditEvents 'EditStarted' 'EditStartedEh'
⎕EDIT '∇MyFn'
```



Stop in the APL64 user-defined function. The event handler function will be suspended in the debugger due to the stop, the event handler argument will be the specified object name and the object editor will not be displayed:

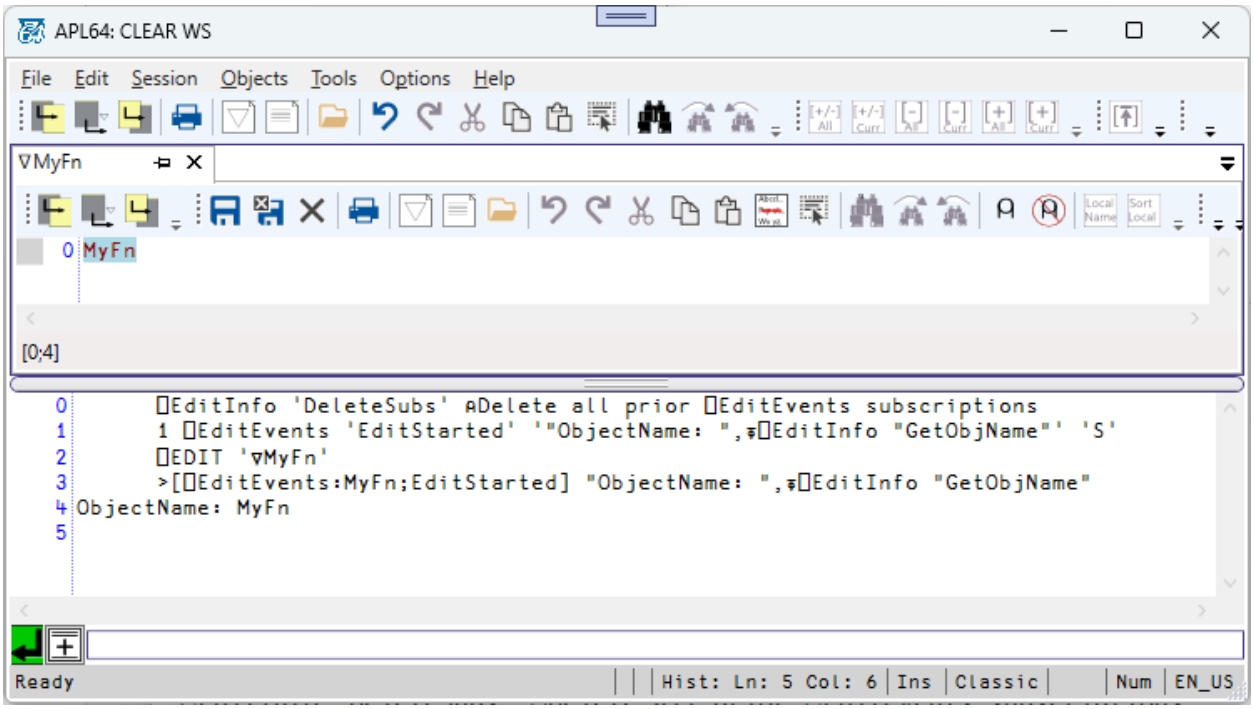
```
⎕EditInfo 'DeleteSubs' ⍝Delete all prior ⎕EditEvents subscriptions
←⎕DEF "EditStartedEh objectName" "'Object name: ', ⍕ objectName"
1 ⎕STOP 'EditStartedEh'
1 ⎕EditEvents 'EditStarted' 'EditStartedEh'
⎕EDIT '∇MyFn'
```



EditStarted Example: APL Statement Event Handler

```
⎕EditInfo 'DeleteSubs' ⍝Delete all prior ⎕EditEvents subscriptions
1 ⎕EditEvents 'EditStarted' '"ObjectName: ",⍕⎕EditInfo "GetObjName"' 'S'
⎕EDIT '∇MyFn'
```

The APL64 executable expression, '"ObjectName: ",⍕⎕EditInfo "GetObjName"', is executed to display the name of the specified object and the MyFn object is presented in an editor:



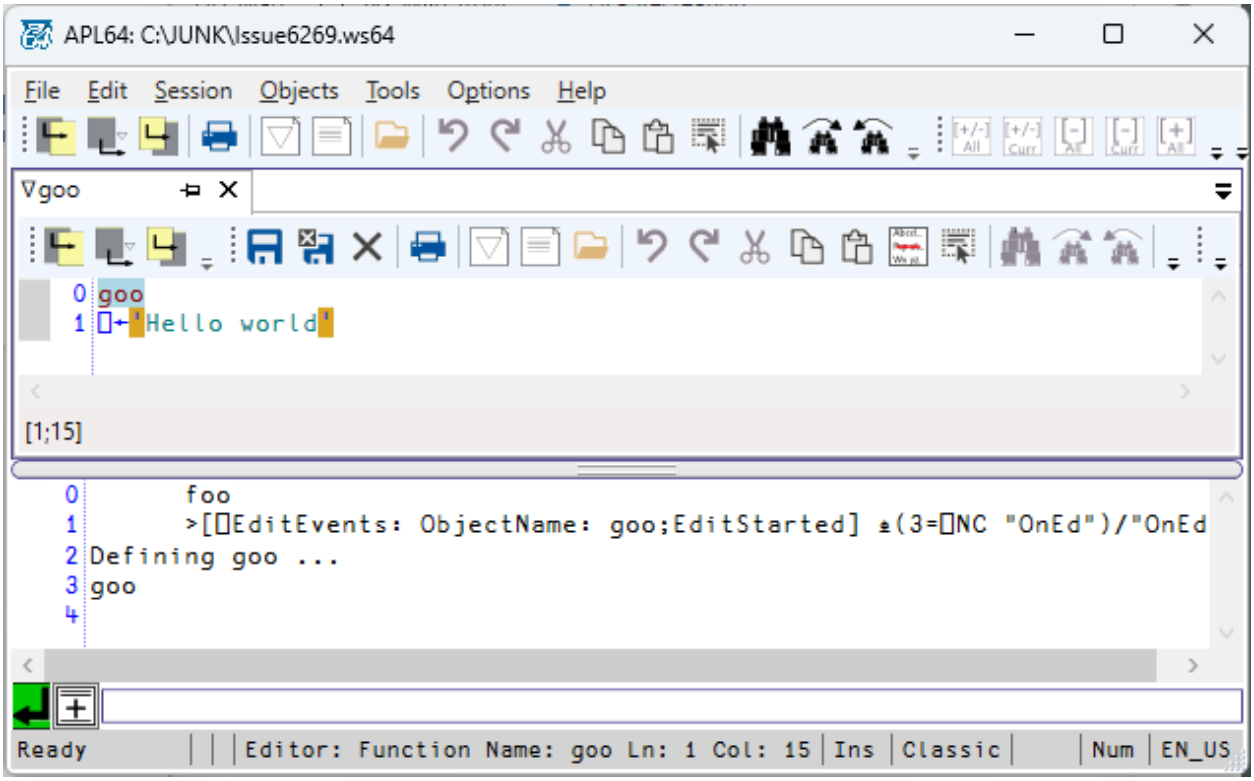
EditStarted Example: APL Statement Event Handler Defining Target Function

In this example the APL64 executable statement to be run when the EditStarted event fires defines the target function to be edited.

```
foo
 ⎕EditInfo 'DeleteEditEventsSubscriptions'
1 ⎕EditEvents 'EditStarted' '⍎(3=⎕NC "OnEd")/"OnEd"' 'Statement'
 ⎕ERASE 'goo' ⍝ make sure it does not exist
 ⎕EDIT 'goo'
```

```
OnEd;N
 N←⎕EditInfo 'GetObjName'
 :if 0=⎕NC N
   ⍝ Pretend to load function from code base
   ⎕←'Defining ',N,' ...'
   ⎕DEF N "⎕←'Hello world'"
 :end
```

When the foo function is run, the goo function is defined and its source code is presented in a function editor:

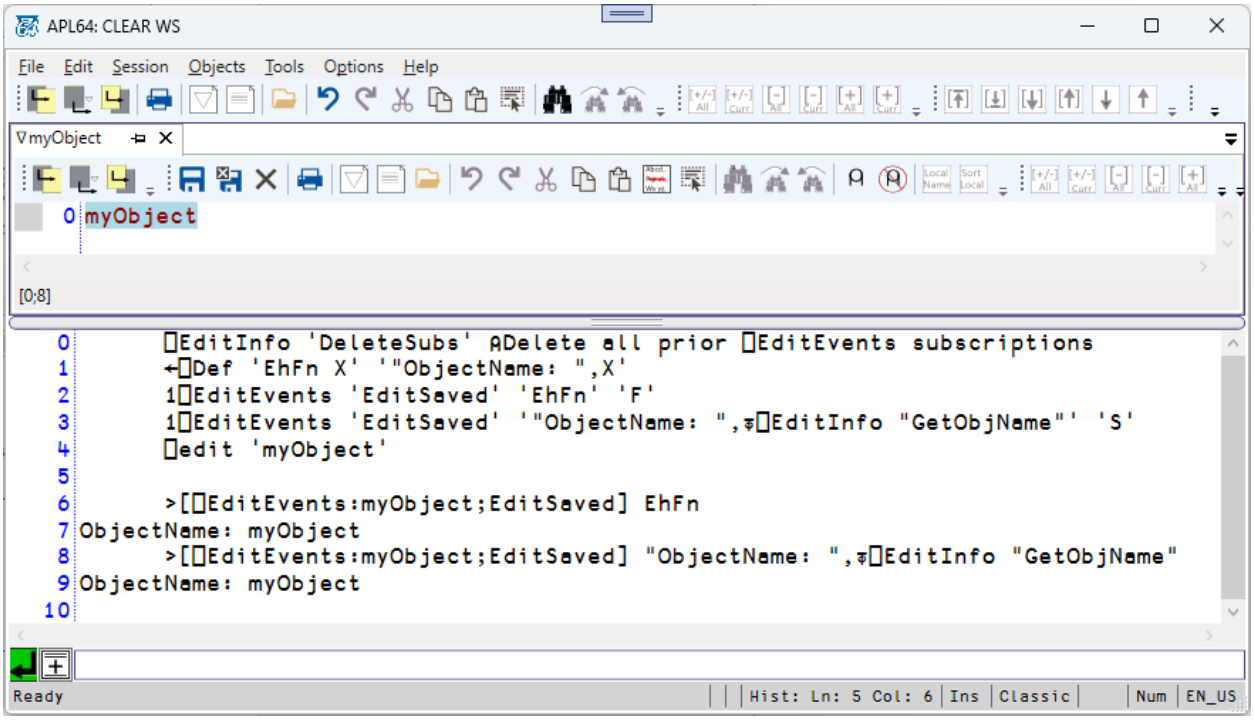


EditSaved Example: APL Function Event Handler

Two EditSaved event handlers are subscribed, an APL64 user-defined function (EhFn) and an APL64 executable expression ("ObjectName: ",⍕⎕EditInfo "GetObjName"). Both of the event handlers are executed after the user manually saves the specified object in the editor:

```
⎕EditInfo 'DeleteSubs' ⍝Delete all prior ⎕EditEvents subscriptions
←⎕Def 'EhFn X' '"ObjectName: ",X'
1⎕EditEvents 'EditSaved' 'EhFn' 'F'
1⎕EditEvents 'EditSaved' '"ObjectName: ",⍕⎕EditInfo "GetObjName"' 'S'
⎕edit 'myObject'

⍝Manually save myObject in the editor!
```

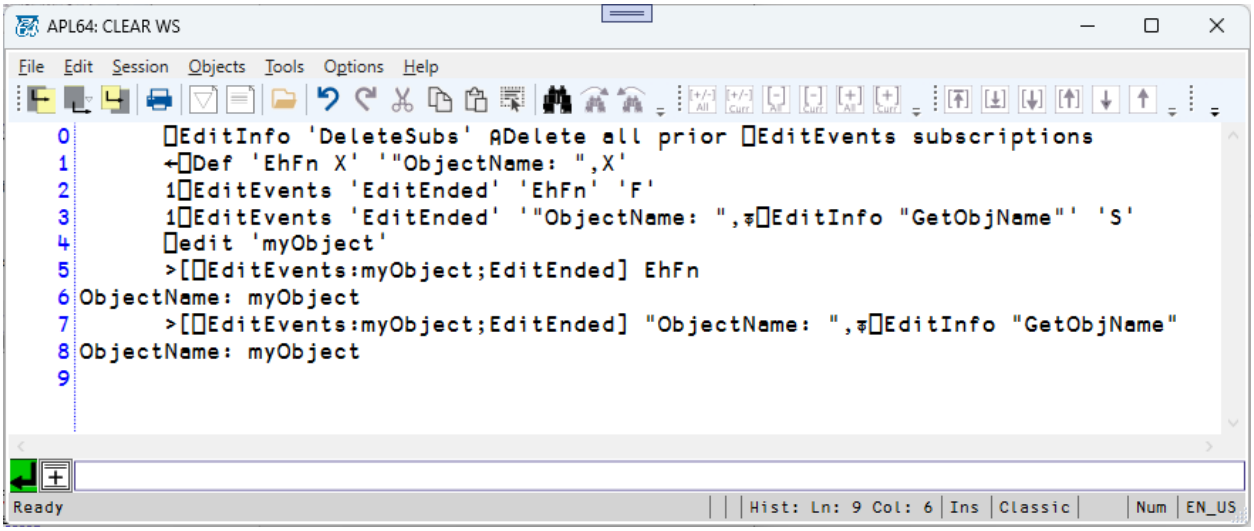


EditEnded Example: APL Function Event Handler

Two EditEnded event handlers are subscribed, an APL64 user-defined function (EhFn) and an APL64 executable expression ("ObjectName: ", ⍕ ⍝EditInfo "GetObjName"). Both of the event handlers are executed after the user manually ends the editing of the specified object in the editor:

```
⍝EditInfo 'DeleteSubs' ⍝Delete all prior ⍝EditEvents subscriptions
←⍝Def 'EhFn X' '"ObjectName: ",X'
1⍝EditEvents 'EditEnded' 'EhFn' 'F'
1⍝EditEvents 'EditEnded' '"ObjectName: ", ⍕ ⍝EditInfo "GetObjName"' 'S'
⍝edit 'myObject'

⍝Manually end the editing of myObject in the editor!
```



ObjectNameSelected Example: APL Function Event Handler

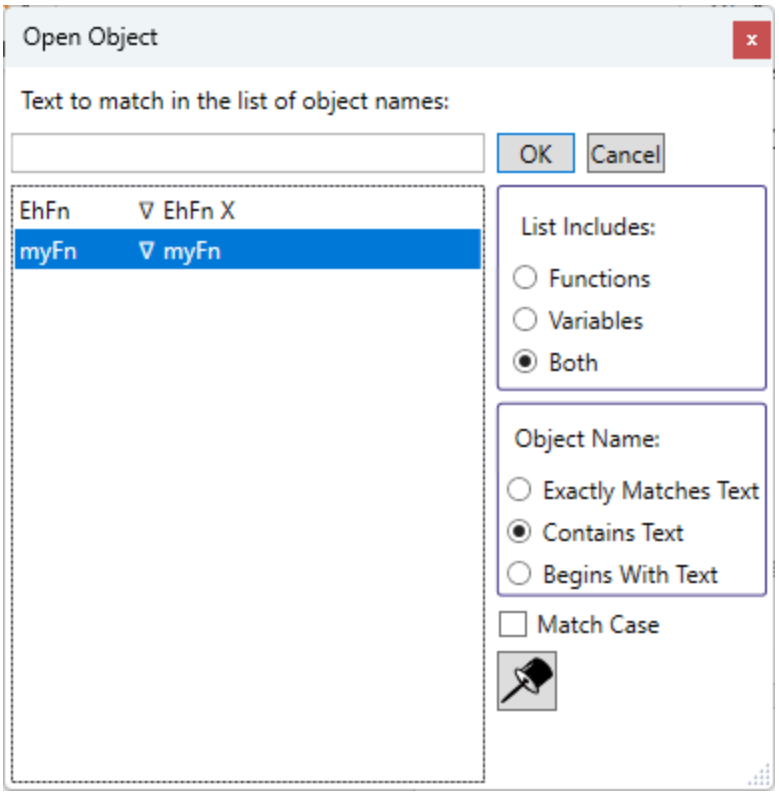
Two ObjectNameSelected event handlers are subscribed, an APL64 user-defined function (EhFn) and an APL64 executable expression ("ObjectName: ", ⍕ ⎕EditInfo "GetObjName").

The object selector dialog is accessed in the APL64 developer version using Ctrl+O, Ctrl+G, Objects | Open or Objects | Fetch. Both of the event handlers are executed after the user manually selects an object from the object selector dialog:

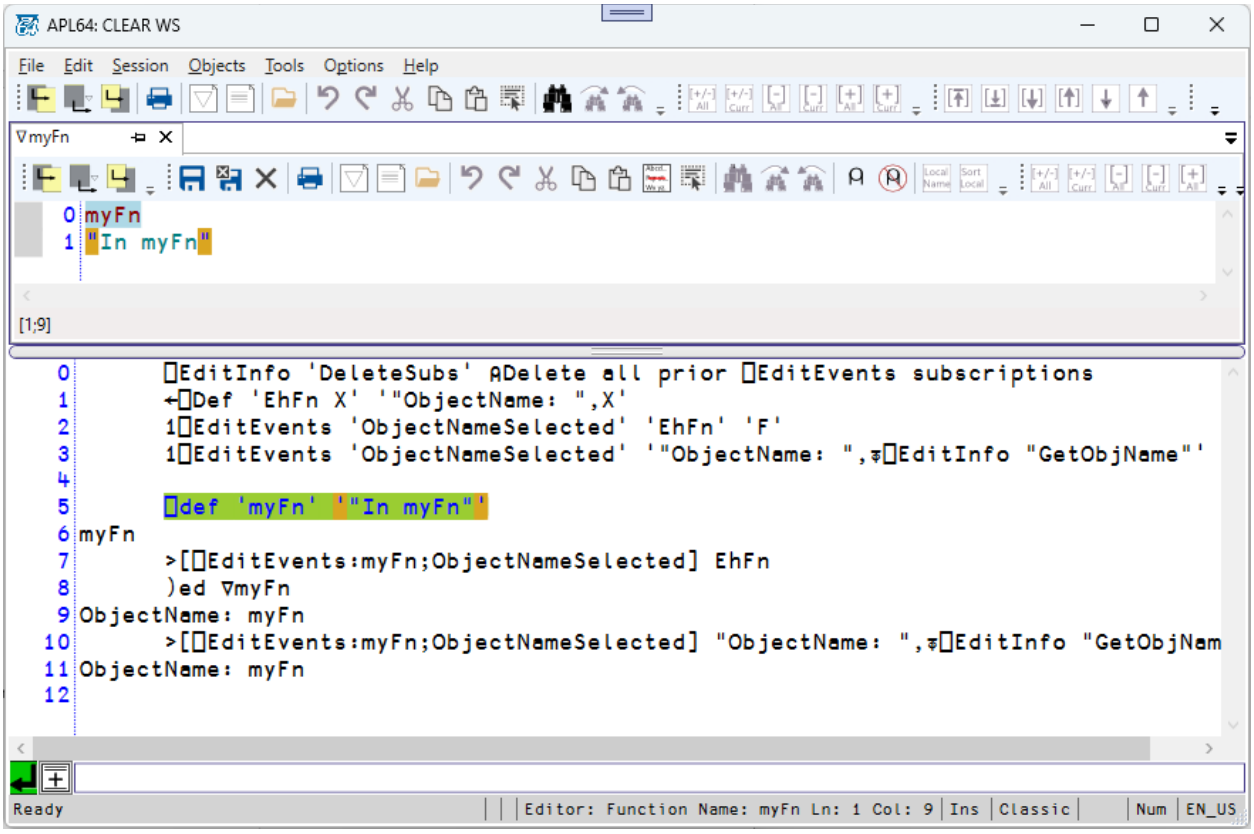
```
⎕EditInfo 'DeleteSubs' ⍝Delete all prior ⎕EditEvents subscriptions
←⎕Def 'EhFn X' '"ObjectName: ",X'
1⎕EditEvents 'ObjectNameSelected' 'EhFn' 'F'
1⎕EditEvents 'ObjectNameSelected' '"ObjectName: ",⍕⎕EditInfo "GetObjName"' 'S'

⎕def 'myFn' '"In myFn"'
⍝^ Manually define a function, so that it can be selected in the object selector dialog

⍝Manually select the object to edit from the object selector dialog and click OK
```



The event handlers will be executed and the selected object will be presented in an editor:

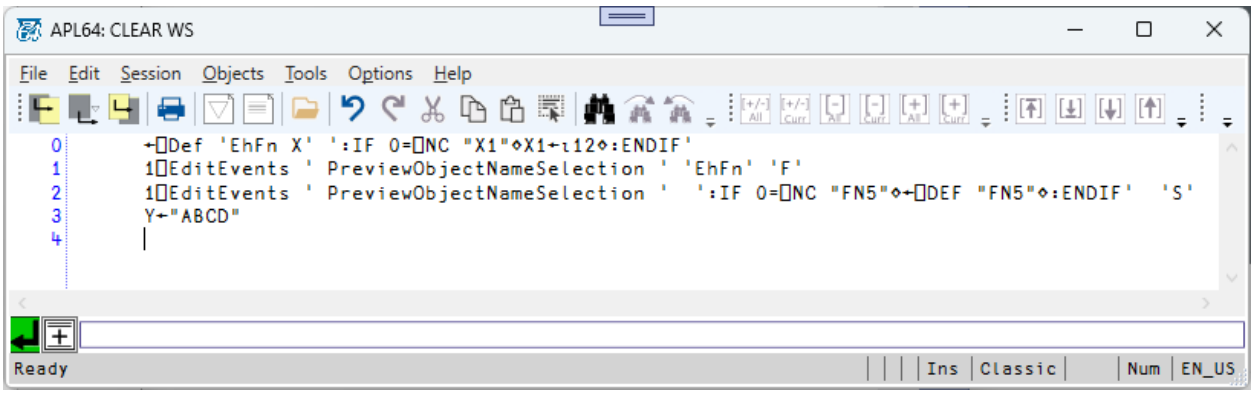


PreviewObjectNameSelection Example: Function Event Handler

Two PreviewObjectNameSelection event handlers are subscribed, an APL64 user-defined function (EhFn) and an APL64 executable expression (':IF 0=⎕NC "FN5"⋄←⎕DEF "FN5"⋄:ENDIF').

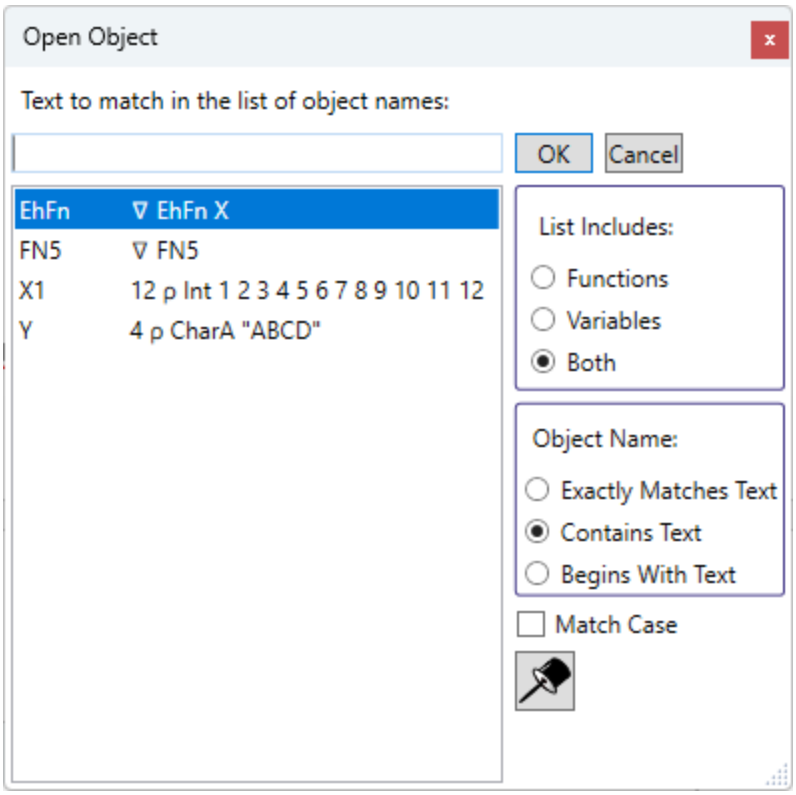
The object selector dialog is accessed in the APL64 developer version using Ctrl+O, Ctrl+G, Objects | Open or Objects | Fetch. Both of the event handlers are executed immediately before the object selector dialog is activate for user interaction.

```
Start a new APL64 Developer version instance
←⎕Def 'EhFn X' ':IF 0=⎕NC "X1"⋄X1←⍳12⋄:ENDIF'
1⎕EditEvents ' PreviewObjectNameSelection ' 'EhFn' 'F'
1⎕EditEvents ' PreviewObjectNameSelection ' ':IF 0=⎕NC "FN5"⋄←⎕DEF "FN5"⋄:ENDIF' 'S'
Manually define a variable Y1←"ABCD" so its name will be included in the object selector dialog
Use Objects | Open to present the object selector dialog
```



The object selector dialog will illustrate the name of the objects in the workspace including:

- The previously-existing Y1 variable
- X1 variable and FN5 user-defined function instantiated by the PreviewObjectNameSelection event handler
- The user-defined event handler function EhFn



⎕editinfo Obtain information about object editors

Purpose: Obtain and use information about the APL64 function and variable editors.

Syntax: result ←[larg] ⎕editinfo action [rarg]

Arguments:
larg are the optional arguments to the action.

action is a text vector that determines how the system function will operate on the left and right argument.

rarg are the optional arguments to the action.

Result:
result, if any, depends on the action used.

Remarks:

- The required ⎕editinfo action argument is a non-case-insensitive, text value.
- The ⎕editinfo, ⎕edit and ⎕editevents system functions may be used to develop user tools in an APL64 Developer version.

Action: GetEditEventsSubscriptions

Use this action to obtain the subscriptions created using the ⎕EditEvents system function.
The abbreviated 'GetSubs' action name can also be used for this action.

Syntax: subscriptionInfo ← ⎕editinfo ' GetEditEventsSubscriptions' [EventType]

The EventType argument is optional. If present, the subscriptionInfo result is filtered to contain event subscriptions of the type specified by the EventType argument. The available values for the EventType argument are:

- EditEnded
- EditSaved
- EditStarted
- ObjectNameSelected
- PreviewObjectNameSelection

The subscriptionInfo result is a matrix of character vectors containing one row for each ⎕EditEvents subscription. The columns of the subscriptionInfo result are:

- EhText: Depending on the EhType, this will be an event handler function name or the text of an APL executable statement.
- EditEventType: One of the ⎕EditEvents action names
- EhType: FunctionName or AplStatementText

Action: DeleteEditEventsSubscriptions

Use this action to delete all subscriptions or all subscriptions of a specified EventType.
The abbreviated 'DeleteSubs' action name can also be used for this action.

Syntax: ⎕editinfo ' DeleteEditEventsSubscriptions' [EventType]

The EventType argument is optional. If present, the subscriptionInfo result is filtered to contain event subscriptions of the type specified by the EventType argument. The available values for the EventType argument are:

- EditEnded
- EditSaved
- EditStarted
- ObjectNameSelected
- PreviewObjectNameSelection

Action: GetEditEventsObjName

Use this action to obtain the objectName associated with specified ⎕EditEvents actions.
The abbreviated 'GetObjName' action name can also be used for this action.

Syntax: objectName ← ⎕editinfo ' GetObjName'

- The objectName result is the useful only while the APL programmer-specified event handler, associated with an ⎕EditEvents EditEnded, EditSaved, EditStarted or ObjectNameSelected event, is running. Otherwise, the objectName result is an empty character vector.

Action: GetEditEventsObjTypes

Use this action to obtain the objTypes associated with specified ⎕EditEvents actions.
The abbreviated 'GetObjTypes' action name can also be used for this action.

Syntax: Int32[] objectTypes← ⎕editinfo ' GetObjTypes'

The objectTypes result is the useful only while the APL programmer-specified event handler, associated with an ⎕EditEvents PreviewObjectNameSelected event, is running. Otherwise, the objectName result is an empty integer vector.

If the result contains 2, the object selector dialog will display functions.

If the result contains 3, the object selector dialog will display variables.

Action: GetFnEdIds

Use this action to obtain function editor identification information which may be necessary for the other ⎕editinfo actions. There can be multiple non-modal function editors active in an APL64 Developer instance. This action considers only non-modal function editors.

Syntax: (fnEdIds indexOfKbdFocusedFnEd) ← ⎕editinfo 'GetFnEdIds'

- The first element of result, fnEdIds, is a vector of function editor identifiers for the current non-modal function editors in the APL64 Developer version GUI. A function editor identifier is a two-element vector of character vectors with format: (fnName GUID).
    - The fnName element of a function editor identifier is the current name of the function in the function editor. The fnName element value may change as the user edits the function text.
    - The GUID element of a function editor identifier is a character vector which does not change during the current APL64 Developer version instance as the function editor text is modified. Each active, non-modal function editor will have a distinct GUID. The GUID is not preserved when a function editor is closed.

- The second element of result, indexOfKbdFocusedFnEd, is the index, considering the current value of ⎕io, in the first element of the result indicating which function editor has the keyboard focus, or ¯1, if no non-modal function editor has the keyboard focus. No function editor may have the keyboard focus because the Command Line, Session History, Debugger pane, State Indicator pane. Modal editor or a variable editor may have the keyboard focus.

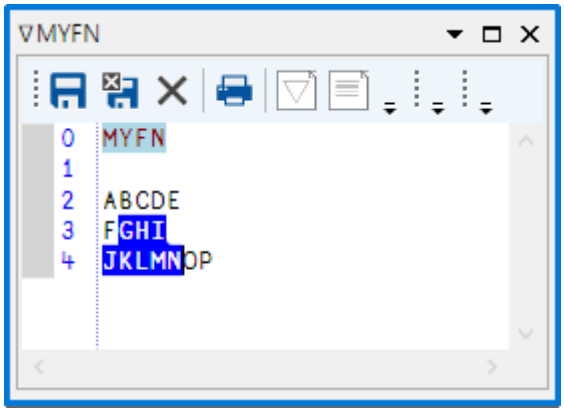
Action: GetFnText

Use this action to obtain the function text and current text selection from a specified function editor. This action considers only non-modal function editors.

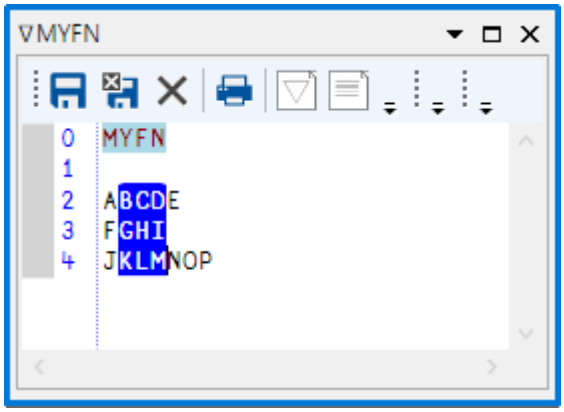
Syntax: (fnText selSegsMatrix caretOffset) ← ⎕editinfo 'GetFnText' FnEdId

- The required FnEdId right argument is obtained from the result of the ⎕editinfo ‘GetFnEdIds’ action. The FnEdId need not be associated with the function editor, if any, which has the keyboard focus.
- The first element of result, fnText, is a character vector containing the text of the specified function editor.
- The second element of result, selSegsMatrix, is an integer matrix with one row for each segment selection in the specified function editor, if any.
    - Each segment selection in the result contains two columns:
        - Index of the start of the segment selection in the function text

        - Length of the segment selection
    - If there is no current selection in the function editor text, the second element of the result will have no rows.
    - If the current selection in the function editor text is a continuous, linear selection, the second element of the result will have one row. If the linear selection spans multiple rows, the line-end characters are considered in the current selection information.



    - If the current selection in the function editor is a discontinuous rectangular selection, the selection element of the result will have one row for each selection segment of the rectangular selection.



- The third element of the result, caretOffset, is the index in the function text of the current caret position.

Action: SetFnText

Use this action to set the function text and, optionally, the current text selection of a specified function editor. This action considers only non-modal function editors.

Syntax: (FnText SelectionVector caretAtEndOfSelection) □editinfo 'SetFnText' 'FnEdId'

- The required FnEdId argument is obtained from the first element of the result of the □editinfo 'GetFnEdIds' action. The FnEdId need not be associated with the function editor, if any, which has the keyboard focus.
- The required FnText argument is a character scalar, character vector or string scalar.
- The required SelectionVector argument is an integer vector:

- If there should be no selection in the updated function editor text, the SelectionVector should be (desiredCaretOffset 0), where desiredCaretOffset is the index in the updated function editor text of the desired caret position.
- If the selection in the updated function editor text is to be a continuous linear selection, the SelectionVector should be a two-element integer vector
  - The first element of the SelectionVector is the start of the selection in the updated function editor text
  - The second element of the SelectionVector is the length of the selection
- If the selection in the updated function editor text is to be a discontinuous rectangular selection, the SelectionVector should be a four-element integer vector.
  - The first element should be the line number, considering the current value of ⎕io, in the updated function editor text of the start of the selection
  - The second element should be the starting column number, considering the current value of ⎕io, of the selection
  - The third element should be the line number, considering the current value of ⎕io, in the updated function editor text of the end of the selection
  - The fourth element should be the ending column number, considering the current value of ⎕io, of the selection

- The required caretAtEndOfSelection argument is a Boolean scalar indicating if the caret position in the updated function editor text should before the beginning of the selection (0) or after the end of the selection (1). This argument has no effect if the updated function editor text will have no selection.

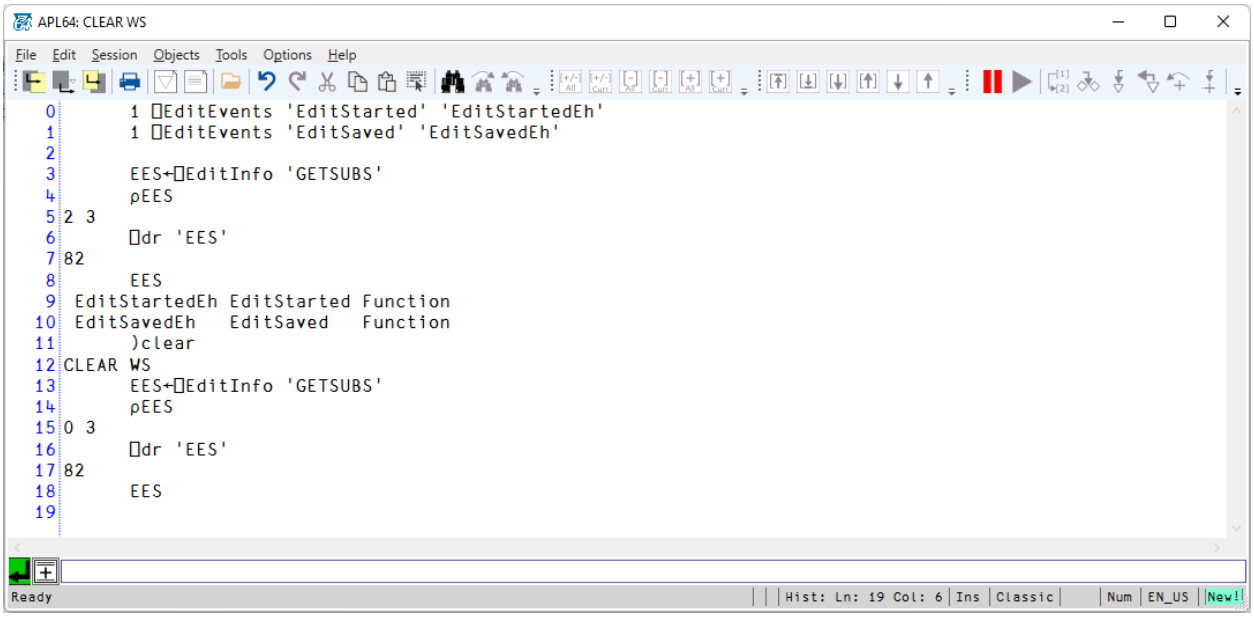
Action: Help

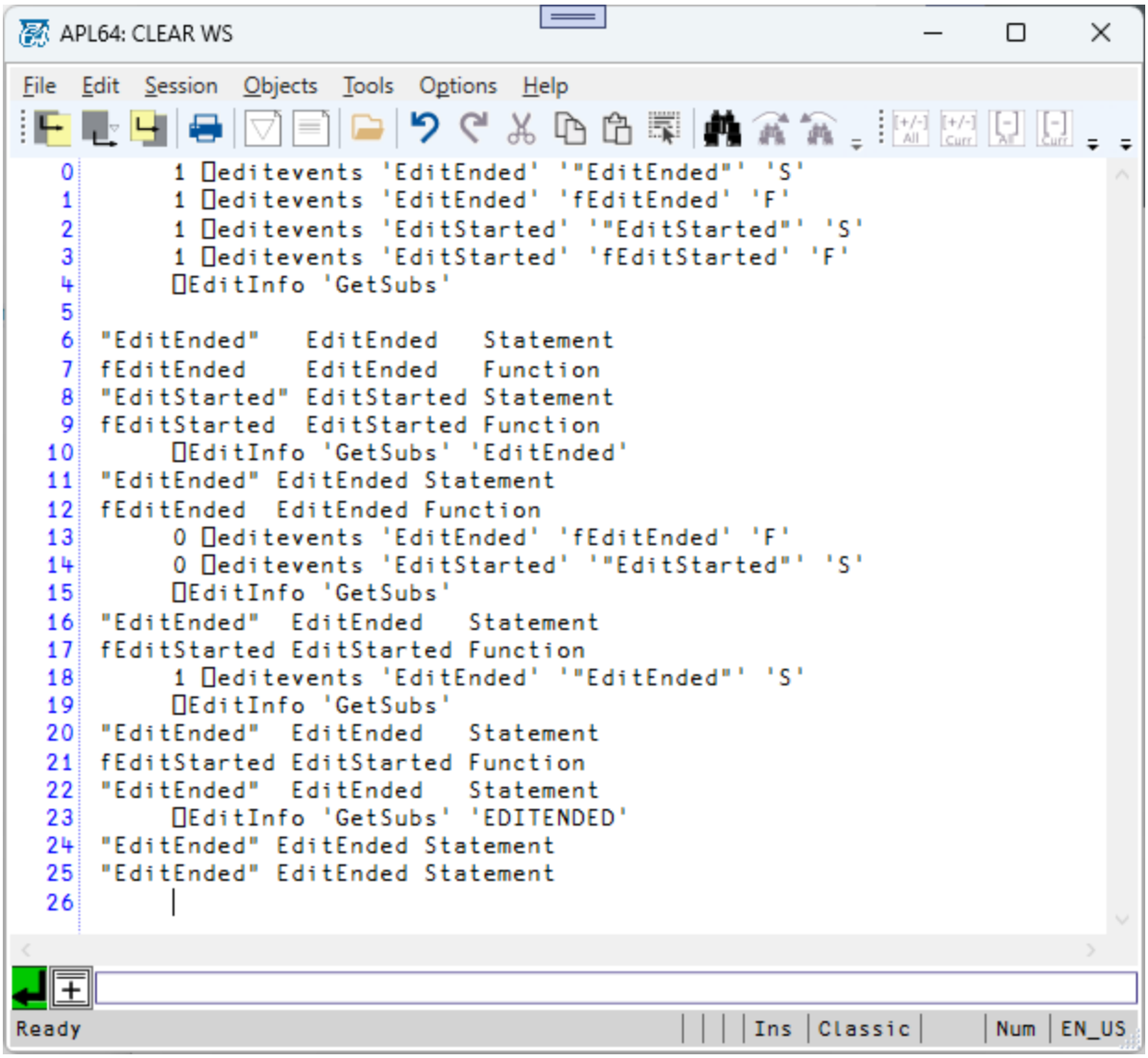
The abbreviated '?' action name can also be used for this action.
The result is a text matrix containing summarized documentation of the ⎕EditInfo actions.

⎕EditInfo Examples

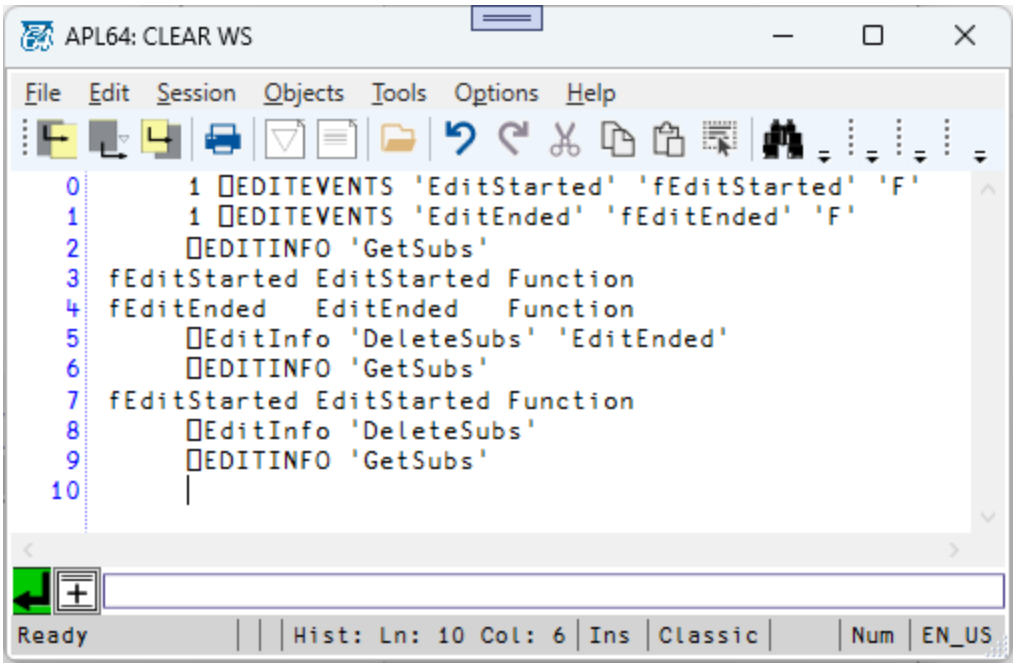
Note: ⎕io is 1 in these examples.

⎕EDITINFO 'GetEditEventsSubscriptions'



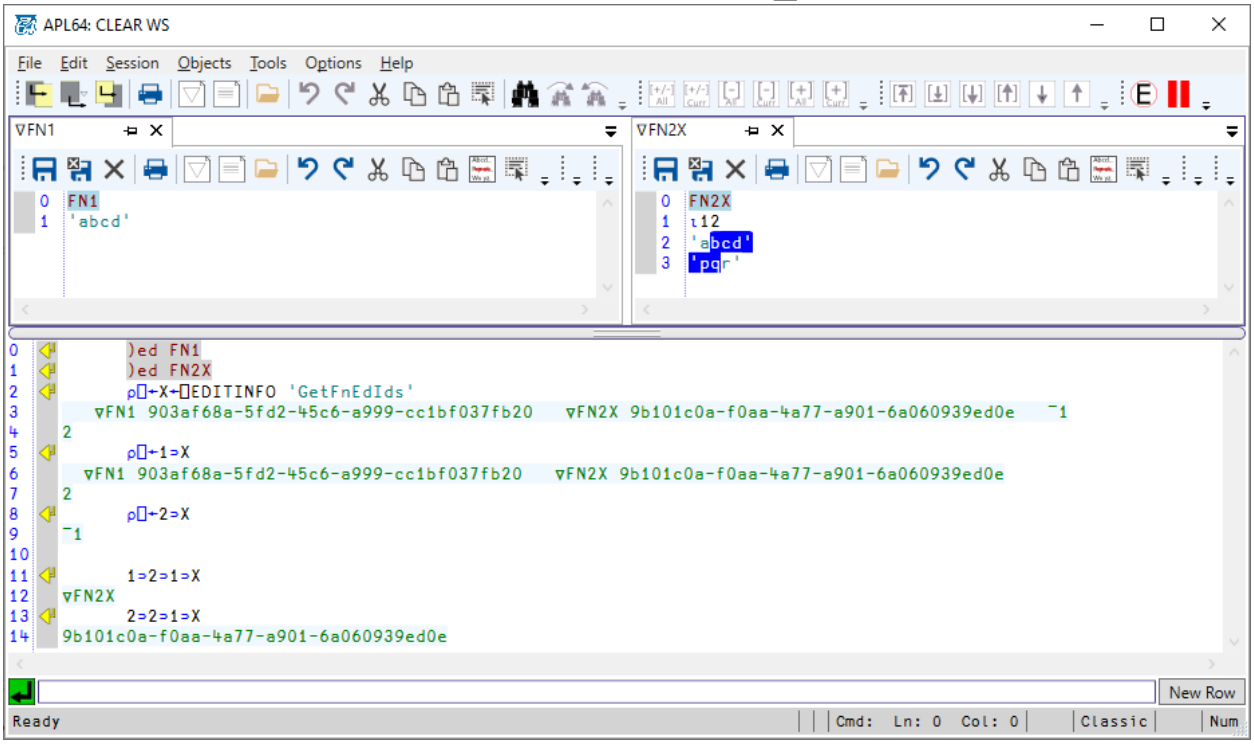


⎕editinfo ‘DeleteEditEventsSubscriptions’



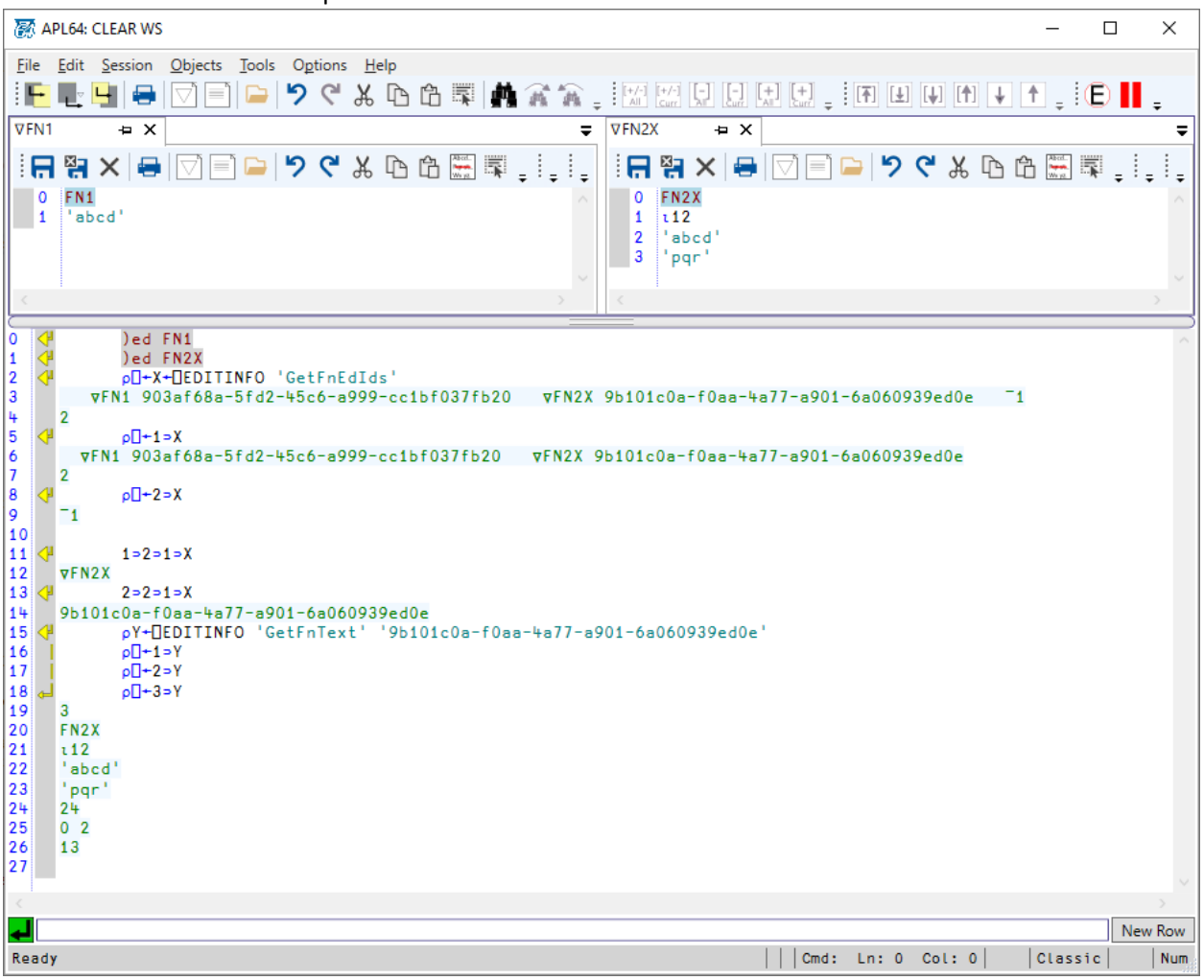
⎕editinfo 'GetFnEdIds'

Two non-modal function editors are active in the APL64 Developer version instance, so the first element of the result of ⎕editinfo 'GetFnEdIds' has two text vector elements. Neither of these function editors have the keyboard focus, so the second element of the result of ⎕editinfo 'GetFnEdIds' is ¯1.



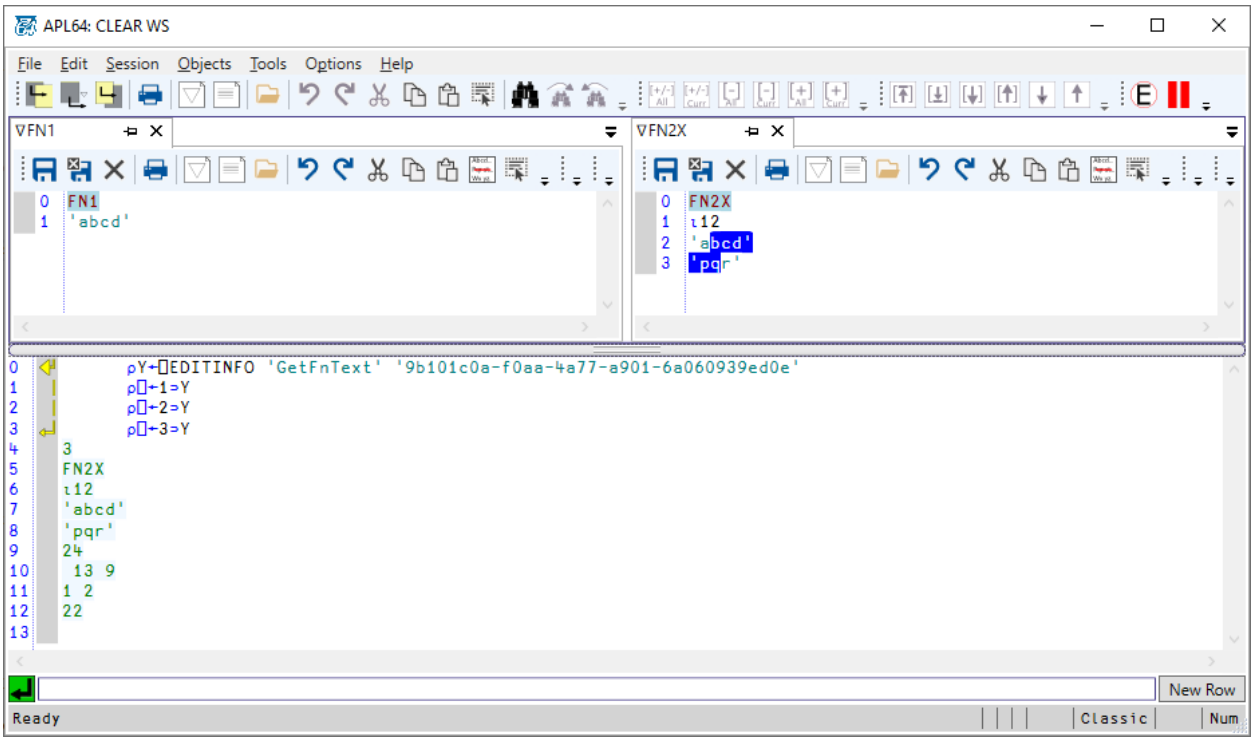
⎕editinfo 'GetFnText'

Obtain the editor information for the function editor containing the FN2X function text. The first element of the result is the current function editor text. This function text has no selection, so the second element of the result of ⎕editinfo 'GetFnText' has no rows. Although it is not visible in the screen capture, the caret offset is after the 'a' character, so the third element of the result is 13, which considers the two line-end characters which occur prior to the caret offset.



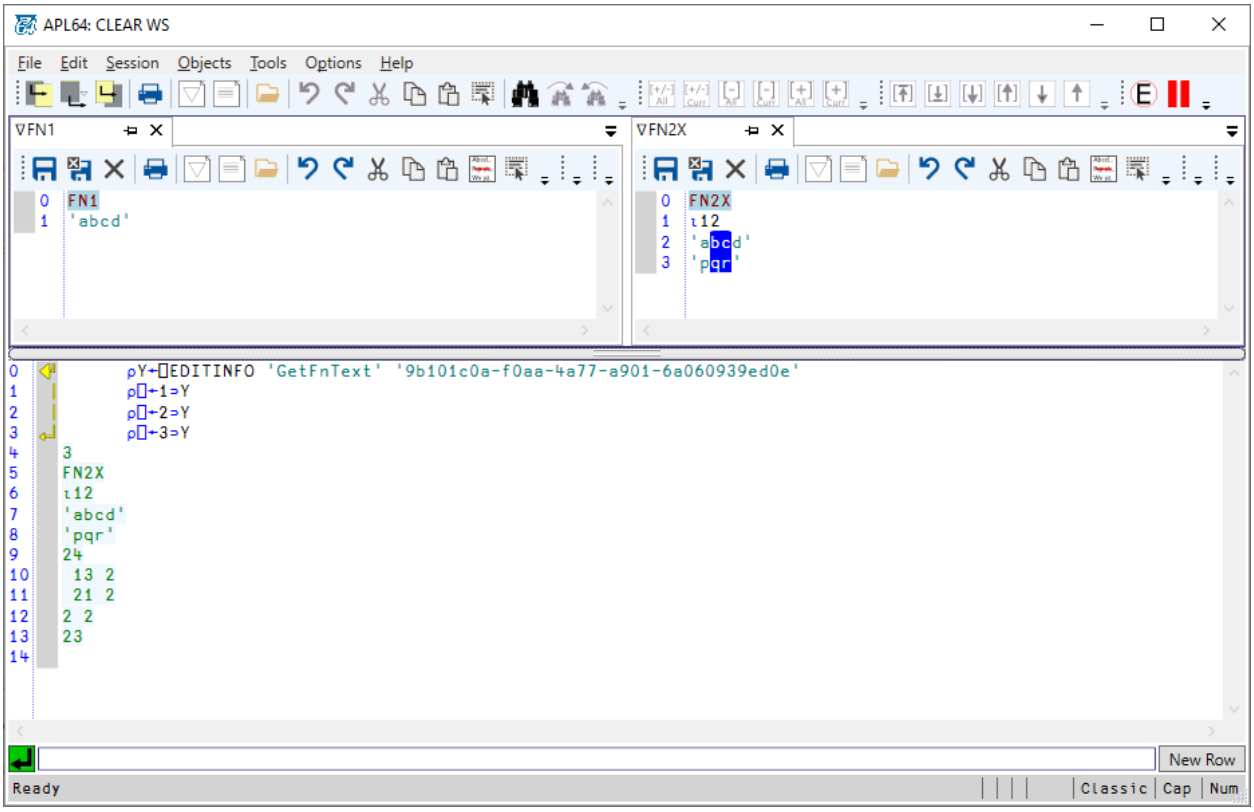
⎕editinfo 'GetFnText'

Obtain the editor information for the function editor containing the FN2X function text. This function text now has a continuous, linear selection, so the second element of the result of ⎕editinfo 'GetFnText' has one row containing the index in the function text of the start of the selection and the length of the selection. Although it is not visible in the screen capture, the caret offset is at the end of the selection.



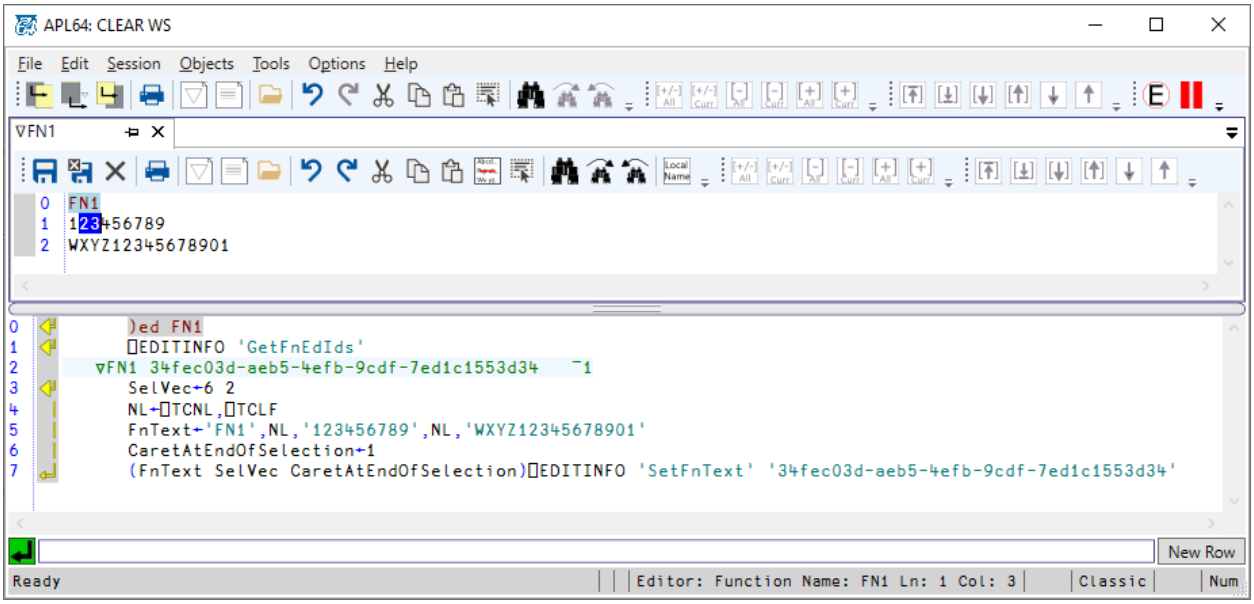
⎕editinfo 'GetFnText'

Obtain the editor information for the function editor containing the FN2X function text. This function text has a discontinuous, two-row, rectangular selection, so the second element of the result of ⎕editinfo 'GetFnText' has two rows, one for each selection segment in the rectangular selection. Although it is not visible in the screen capture, the caret offset is at the end of the selection.

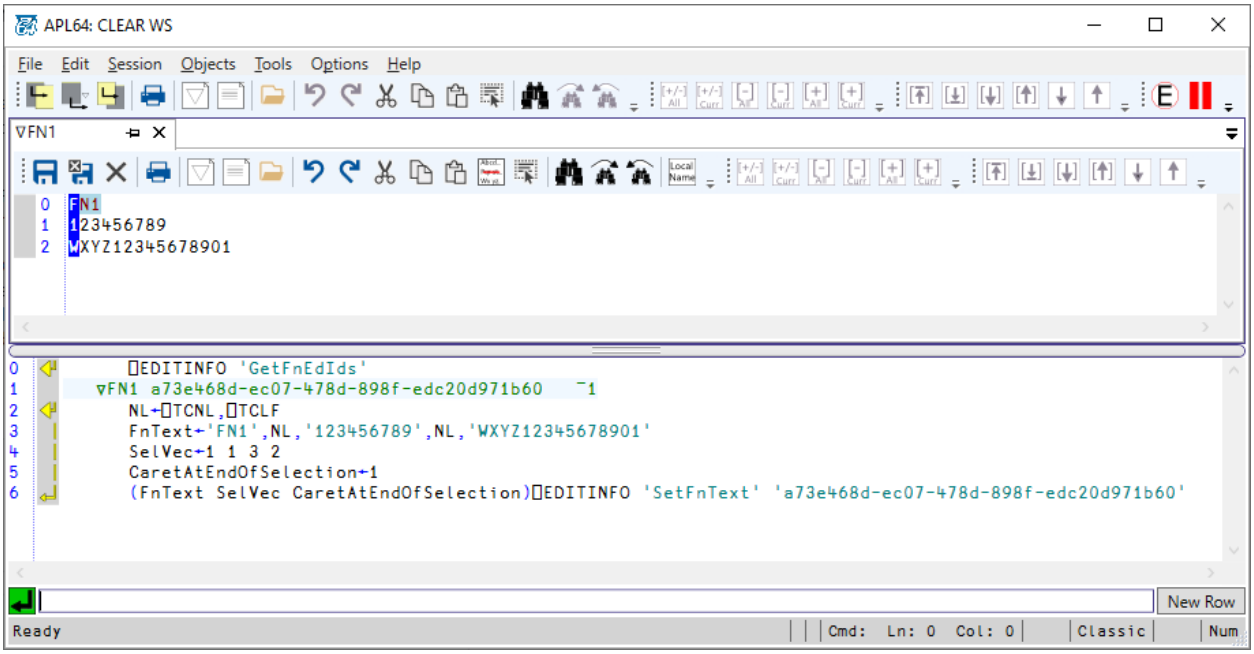


⎕editinfo 'SetFnText'

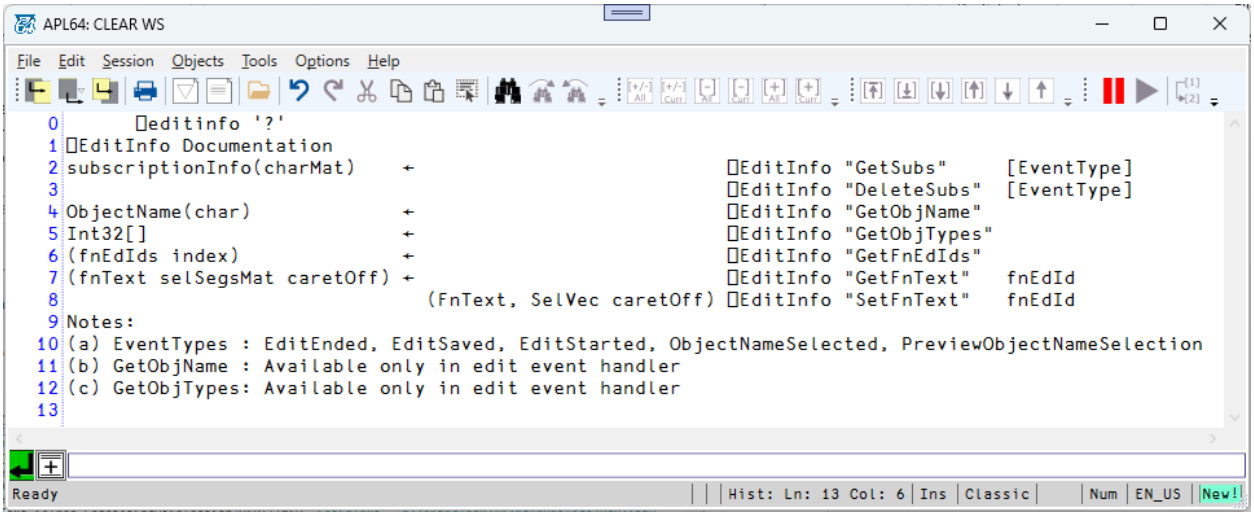
Function editor state after executing ⎕editinfo 'SetFnText' to create a continuous, linear selection. The function text is entirely replaced. The linear selection in the updated function text begins at the sixth character and has a length of two characters. The linear selection start considers the one line-end character in the function editor text prior to the selection. The caret is placed after the end of the selection, although it is not visible in the screen capture.



Function editor state after executing ⎕editinfo 'SetFnText' to create a non-continuous, rectilinear selection. The function text is entirely replaced. The rectilinear selection in the update function text starts on the first line and first column and extends to the third line and second column. The caret is placed after the end of the selection, although it is not visible in the screen capture.



⎕editinfo '?'



Debugging ⎕EditEvents Event Handlers

Debugging an ⎕EditEvents event handler often requires editing the event handler, but this can cause the pre-existing ⎕EditEvents subscribed events to fire. In this scenario, it is necessary to unsubscribe the applicable ⎕EditEvents subscriptions before attempting to edit the defective programmer-defined event handler function. For this purpose, use ⎕editinfo ' DeleteEditEventsSubscriptions' [EventType].

```
⎕def 'EditStartedEh X' '3÷0'
1 ⎕EditEvents 'EditStarted' 'EditStartedEh' 'F'
```

In this example an exception is thrown by the EditStarted event handler function EditStartedEh when the user attempts to initialize an editor. After that attempt, each attempt to edit the EditStartedEh function adds another level to the state indicator, but does not permit the event handler function to be edited. In this case, unsubscribe the EditStarted event handler using use ⎕editinfo ' DeleteEditEventsSubscriptions' 'EditStarted', so that the EditStartedEh event handler function can be corrected. After the correction has been made, use 1 ⎕EditEvents 'EditStarted' 'EditStartedEh' 'F' to subscribe the EditStarted event for re-testing.