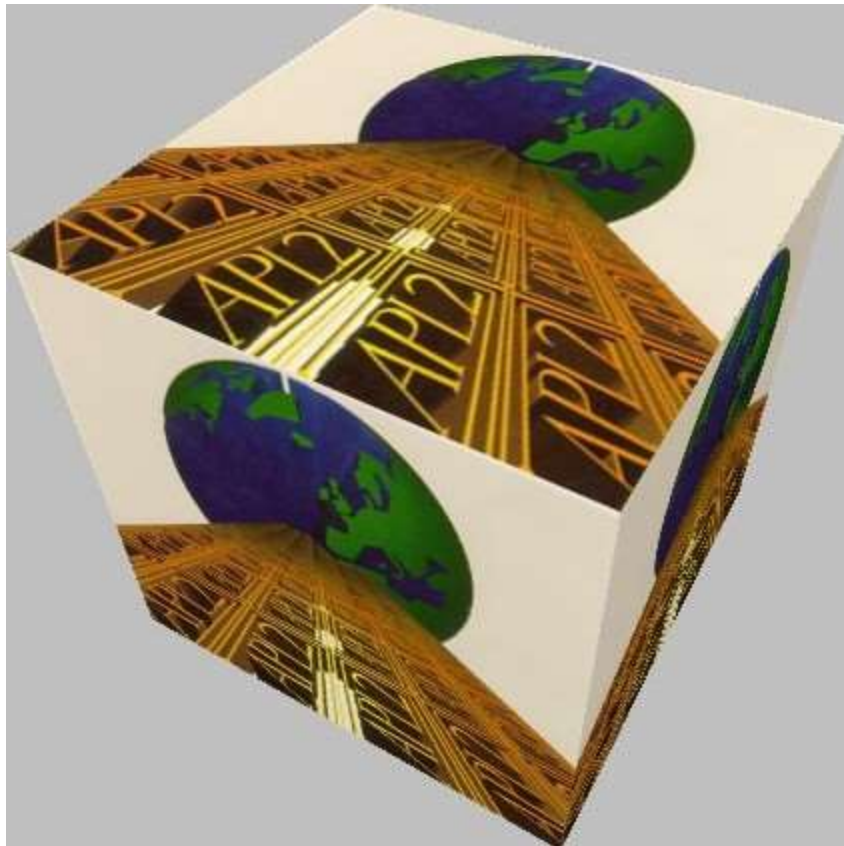


# **APL2 Programming: Developing GUI Applications**

**SC18-7383-17**

APL Products and Services  
IBM Silicon Valley Laboratory  
555 Bailey Avenue  
San Jose, California 95141  
[APL2@vnet.ibm.com](mailto:APL2@vnet.ibm.com)



## ***Copyrights***

© Copyright IBM Corporation 2003, 2017 All Rights Reserved.

US Government Users Restricted Rights - Use, duplication or disclosure restricted by GSA ADP Schedule  
Contract with IBM Corporation

# Contents

Contents .....	1
Notices .....	4
Programming Interface Information .....	5
Trademarks .....	6
We Would Like to Hear from You .....	7
Overview .....	9
A Quick Example.....	10
GUI Fundamentals .....	12
Building Dialogs Dynamically .....	14
Creating a Dialog .....	15
Setting the Dialog's Title .....	16
Using the Status area.....	17
Setting the Dialog's Event Handlers.....	18
Adding a Menu Bar.....	19
Adding Controls.....	21
Size, Position, and Client Area Properties .....	23
Arranging Controls .....	25
Focus, Dialog Navigation, Groups, and Tab Stops.....	28
Mnemonics.....	29
Making Dialogs Resizable .....	31
Providing Online Help .....	34
Fonts.....	35
Building Dialogs using the Dialog Editor.....	36
Starting a New Dialog.....	37
Adding and Deleting Controls .....	38
Selecting and Arranging Controls.....	39
Controlling the Arrange Settings .....	40
Using Menus and the Status Area with Dialog Templates .....	41
Ordering Controls .....	42
Event Handlers.....	43
Changing a Dialog or Control.....	44
Providing help to your users .....	45
Testing the Dialog.....	46
Using Dialog Templates .....	47
Dialog Box Design Guidelines .....	48
Executing Dialogs.....	49
Getting User Input.....	50
Supporting APL Input.....	51
Sharing Variables with Window Properties.....	52
Building More Complex Applications.....	53
Owned, Modal, and Modeless Windows .....	54
Creating Owned Dialogs.....	55
Executing Modal and Modeless Dialogs .....	56
Using Arbitrary Arrays as Event Handlers .....	58
Message Boxes.....	59
Popup Menus .....	61

Font Dialogs.....	62
File Dialogs.....	63
Displaying File Information in Dialogs.....	64
Tab Controls.....	66
Split Dialogs.....	68
Displaying Graphics in a Dialog.....	69
Using AP 145 with other Auxiliary Processors.....	70
Remaining Responsive During Long Operations.....	72
Using AP 145 Services.....	74
Using System Services.....	76
Objects: Timers and DDE.....	88
Timers.....	89
Dynamic Data Exchange.....	90
Class Reference.....	93
Dialog.....	94
ActiveX.....	99
Check box.....	102
Combo box.....	104
Custom.....	108
Date.....	112
Entry field.....	114
Frame.....	116
Graphic Windows (AP 207).....	117
Grid.....	118
Group box.....	125
List box.....	127
Listview.....	130
Menus and Menu Items.....	135
MLE.....	136
Month.....	139
Progress Bar.....	142
Push button.....	144
Radio button.....	146
Rectangle.....	148
Scroll bar.....	149
Slider.....	151
Spin button.....	154
Tab.....	157
Text.....	160
Time.....	162
Treeview.....	164
Desktop.....	168
Common Property Reference.....	170
Object Reference.....	176
DDE DATA.....	177
DDE COMMAND.....	178
DDE SERVER.....	179
DDE TOPIC.....	180

DDE ITEM.....	181
TIMER .....	182
GUITOOLS Function Reference .....	183
GPDLGPROCESS - Dialog Processing Tools .....	184
GPUTILITY - Utilities .....	204
GPPRINT - Printing Tools and Constants .....	208
GUIVARS Constants Reference.....	209
Accelerator Flags .....	210
Accelerator Virtual Key Codes.....	211
Menu Style Flags .....	212
Menu Attribute Constants .....	213
Message Box Style Flags .....	214
Message Box Result Codes.....	215
DEMO145 Function Reference .....	216
Appendix A: AP 145 Services .....	218
Apl Services.....	220
Appendix B: Where to find Tools, Samples, and Other Information .....	233
Appendix C: Using APL2, DDE and Microsoft Excel .....	235
Appendix D: Using APL2, DDE and Microsoft Access.....	236
Appendix E: Microsoft Windows Common Control Library .....	242
Appendix F: GDI+ Graphics Device Interface Plus .....	243

## **Notices**

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe on any of IBM's intellectual property rights may be used instead of the IBM product, program, or service. Evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, are the user's responsibility.

IBM may have patents or pending patent applications covering subject material in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing  
IBM Corporation  
North Castle Drive  
Armonk, NY 10504-1785  
U.S.A.

## **Programming Interface Information**

This user's guide is intended to help programmers write applications in APL2. It documents General-Use Programming Interface and Associated Guidance Information provided by APL2. General-use programming interfaces allow the customer to write programs that obtain the services of APL2.

## **Trademarks**

### **IBM Trademarks**

The following terms are trademarks of the IBM Corporation in the United States or other countries or both:

APL2  
IBM  
OS/2  
Presentation Manager  
VisualAge

### **Other Trademarks**

The following terms are trademarks of the Microsoft Corporation:

ActiveX  
Excel  
Internet Explorer  
Microsoft  
Visual Basic  
Visual Studio  
Windows  
Windows NT



## ***We Would Like to Hear from You***

### **Developing GUI Applications with APL2**

Please let us know how you feel about this online documentation by placing a check mark in one of the columns following each question below:

To return this form, print it, write your comments, and then mail it to:

International Business Machines Corporation  
APL Products and Services  
PGUA/E1  
555 Bailey Avenue  
San Jose, California  
95141  
USA

For postage-paid mailing, please give the form to your IBM representative.

You can also send us your comments on Internet. To send us this form, copy it to a file, write your comments using a file editor, and then send it to:

[apl2@vnet.ibm.com](mailto:apl2@vnet.ibm.com)

Overall, how satisfied are you with the online documentation?

	Very Satisfied		Very Dissatisfied	
	1	2	3	4
Overall Satisfaction	___	___	___	___

Are you satisfied that the online documentation is:

Accurate	___	___	___	___
Complete	___	___	___	___
Easy to find	___	___	___	___
Easy to understand	___	___	___	___
Well-organized	___	___	___	___
Applicable to your tasks	___	___	___	___

Please tell us how we can improve the online documentation:

---

---

---

Thank you! May we contact you to discuss your responses?

Yes       No

Name:

\_\_\_\_\_  
Title:

\_\_\_\_\_  
Company or Organization:

\_\_\_\_\_  
Address:

\_\_\_\_\_  
\_\_\_\_\_

\_\_\_\_\_  
Phone:  
( ) \_\_\_\_\_

Fax:  
( ) \_\_\_\_\_

E-mail:  
\_\_\_\_\_

Please do not use this form to request IBM publications. Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office servicing your locality.

## **Overview**

Workstation APL2 includes tools for building Graphical User Interface (GUI) applications on Microsoft® Windows®. These tools make it easy to build applications with the interface features users have come to expect.

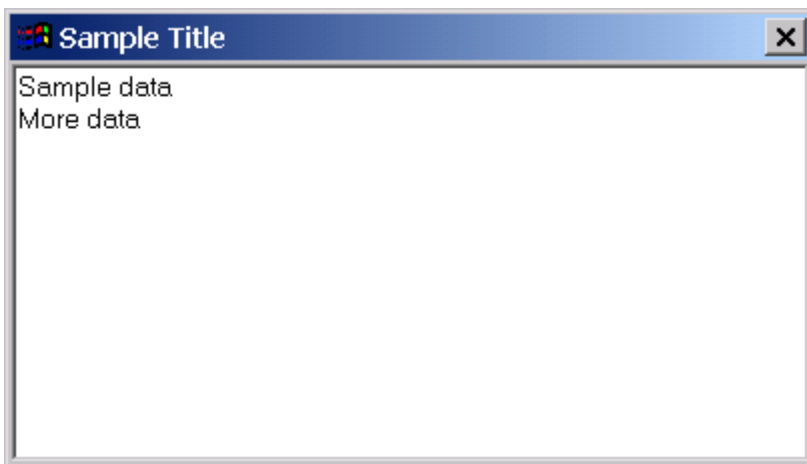
This document describes the tools and procedures used to build GUI applications.

## A Quick Example

Here is an example that demonstrates how easy it can be to build a GUI application:

```
)LOAD 2 GUITOOLS
SAVED 2002-10-01 16.05.15 (GMT-4)
Licensed Materials - Property of IBM
(c) Copyright IBM Corp. 1994, 2002.
  DIALOG←CREATEDLG 'VISIBLE'
  SET_PROPERTY DIALOG 'Sample Title'
  MLE←CREATECTL DIALOG 'MLE' '' 32776
  SET_PROPERTY MLE ('Sample data' 'More data')
  'EVENTS' SET_PROPERTY DIALOG (1 2ρ'CLOSE' '→0')
0 EXECUTEDLG DIALOG
DATA←GET_PROPERTY MLE
DESTROYDLG DIALOG
```

And here is the window the example creates:



Briefly, the example creates a window with a typing area, sets the data displayed by the window, specifies what should happen when the user closes the window, and retrieves the user's data after the window has been closed.

Here are some notes about the sample. It is not important that you understand them all now, but they should give you the flavor of GUI programming in APL2.

- CREATEDLG is used to create an empty visible dialog. (The window with a titlebar.)
- SET\_PROPERTY is used to set properties, or attributes, of windows. In this case, it sets the dialog's title.
- CREATECTL creates components such as list boxes, push buttons, and here, a Multi-Line Edit field.
- The EVENTS property specifies what should happen in response to user actions. In this case, we specify the program should stop waiting for events, or branch to zero, when the user closes the dialog.
- EXECUTEDLG is used to actually wait for and respond to user actions.
- GET\_PROPERTY retrieves window properties. In this case, it retrieves the contents of the typing area.

- DESTROYDLG is used to destroy the dialog. It ends the program.

## **GUI Fundamentals**

This section is a short course on Graphical User Interface concepts as provided by APL2. If you are new to GUI applications, this section should provide you with the information you need to understand the world of GUI programming.

Graphical User Interfaces enable applications to use windows to interact with users. Applications that use standard graphical user interfaces have a consistent look and feel, which makes them easier for users to learn.

There are two main features of APL2 GUI applications: windows and event handlers. Windows are the components of the user interface that the user interacts with. Event handlers are APL expressions to be executed in response to user actions.

### **Types of Windows**

Applications usually have a top-level window that contains a titlebar and a border. These windows are called dialog windows. Dialog windows provide a consistent user interface for application management. For example, they typically include a border, a titlebar, and a system menu containing Move, Size, and Close.

Components such as push buttons, list boxes, and entry fields are called control windows or simply controls. Each different type of control window is called a class. Each class of control window is used to display information in a specialized way or to gather specific types of information. For example, entry fields display and gather text data.

### **Styles and Properties**

Both dialog and control windows have styles and properties. Styles are set when the window is created and generally can't be changed for the life of the window. For example, the Max style is used to create a dialog with a Maximize button. Properties are set, and can be changed, after the window is created. For example, the SIZE property is used to set or query the size of a window.

### **Parents and Children**

Controls within a dialog are said to be children of the dialog. Likewise, the dialog is the parent of its controls. Parent windows control the appearance of child windows. For example, if a dialog is invisible, all its children are hidden.

### **Events**

Windows generate events in response to user actions. For example, when a user presses a push button, the button generates a Command event. Some classes of windows generate many different kinds of events, and some generate no events at all. For example, dialogs generate events for many user actions including Close, Minimize, and Maximize. Text controls generate no events at all since they do not support any user interactions.

AP 145 performs default processing for some, but not all, events. For example, when the user presses a dialog's Maximize button, AP 145 automatically maximizes the dialog. When the user presses a dialog's Close button, AP 145 does not process the event. The application must process the Close event.

To process a window's events, an application must specify event handlers for the events. An event handler is processed when an event occurs. Event handlers are assigned to windows' EVENTS property. Event handlers are only required for the events the application needs to detect.

## Creating Windows

APL2 supports two techniques for designing and creating application windows: dynamic dialog creation and dialog editing.

The CREATEDLG and CREATECTL functions are used to create dialogs and controls dynamically. Parameters of these functions are used to specify window styles. First an empty dialog is created with CREATEDLG and then controls are created with CREATECTL. The SET\_PROPERTY and GET\_PROPERTY functions are then used to set window properties such as size and position.

The Dialog Editor is a graphical tool for designing dialogs. Using the dialog editor you can drag and drop controls on a dialog, resize and arrange windows, and specify window styles, event handlers, and other properties. The Dialog Editor produces dialog templates. A dialog template defines a dialog and all the controls within it. The CREATEDLG function is then used to create a complete application window from the dialog template. Dialog templates can also be created by referencing the TEMPLATE property of a dynamically created dialog.

Dynamic creation supports definition of application windows at runtime. The Dialog Editor supports use of predefined application windows. The Dialog Editor and dialog templates can provide higher performance window creation at the expense of some runtime flexibility. Which technique you use may depend on the degree of control you need at runtime and your preferred style of programming.

## Running GUI Applications

Once an application has created its dialogs and controls, including the appropriate event handlers, it typically calls an APL2 cover function to wait for events. APL2 handles execution of the event handlers. It is important to realize this means that GUI applications usually do not have a single sequential list of instructions. Rather, they have a series of event handler expressions that are executed in response to user actions.

For example, an application could specify the event handler expressions ' PUSH ' for a push button's Command event and ' →0 ' for a dialog's Close event. When the button was pushed, the function PUSH would be executed. When the user closed the window, the Close event handler would cause APL2 to stop waiting for events.

## What's to Come

The next chapter of this book describes creating dialogs and controls dynamically. It provides a step-by-step explanation of the use of the GUI application development tools. The following chapter describes using the Dialog Editor to design windows. Some of the material in the first chapter is also applicable to dialogs created with the dialog editor. You should read both chapters so that you understand all the facilities that are available.

## ***Building Dialogs Dynamically***

The first step in building a GUI application is to load the tools used to create and process windows:

```
)LOAD 2 GUITOOLS
SAVED 2002-10-01 16.05.15 (GMT-4)
Licensed Materials - Property of IBM
(c) Copyright IBM Corp. 1994, 2002.
```

The GUITOOLS workspace contains many tools for use in GUI applications. This section will provide step-by-step instructions for using the basic tools. For more information on the GUITOOLS workspace, see [Where to find Tools, Samples, and Other Information](#).



## Creating a Dialog

To begin, use the `CREATEDLG` function to build a dialog:

```
DIALOG←CREATEDLG 'MIN'
```

The right argument of `CREATEDLG` is a list of dialog styles. By default, `CREATEDLG` creates a dialog with a system menu, a titlebar, and a 3 dimensional border. The statement above adds a minimize button to the dialog. For a complete list of the supported dialog styles, see [Class Reference](#).

By default the dialog is initially invisible. This lets the application initialize the window by adding and positioning controls before showing the dialog to the user. If you want to be able to see the dialog when it is initially created, add the `VISIBLE` style or use the `SHOW` function. After it is visible, the dialog looks like this:



`CREATEDLG` returns an integer scalar called a window handle that uniquely identifies the dialog. The application can use the window handle to refer to the window in calls to other functions. For example, you can destroy the dialog like this:

```
DESTROYDLG DIALOG
```

The `GUITOOLS` functions use Auxiliary Processor 145 to create and manage windows. The `CREATEDLG` function shares a variable named `SV145` with AP 145. This variable should be localized in each application's top-level function. When the variable is retracted, all windows created using it are destroyed. It is good style however, to explicitly destroy windows during application termination.

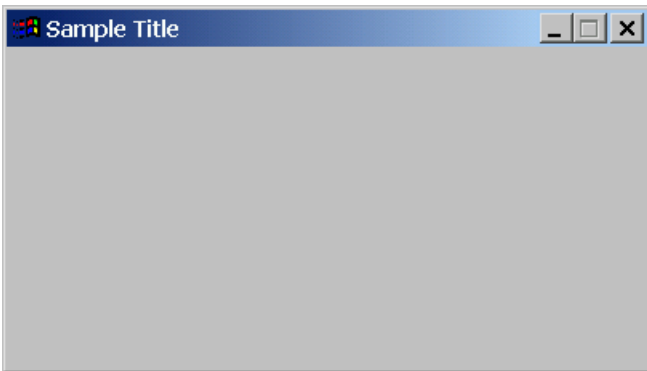
## Setting the Dialog's Title

Use the SET\_PROPERTY function to set the dialog's titlebar:

```
'DATA' SET_PROPERTY DIALOG 'Sample Title'
```

In this example, the right argument of SET\_PROPERTY has 2 elements. The first element is the handle returned by CREATEDLG and the second element is the text to display in the titlebar.

The left argument of SET\_PROPERTY is the name of a property. The DATA property is used to set the data displayed by a window. Since dialogs display data in their titlebars, the DATA property is used to set dialog titlebars. The dialog now looks like this:



Different properties for different classes of windows require different types of values. In this case, the dialog window's DATA property must be a character vector. For more information about the properties supported by the different classes of windows see [Class Reference](#).

## Using the Status area

Each dialog has a status area that can be used for displaying messages. By default, the status area is invisible. The `STATUS_VISIBLE` property is used to make it visible:

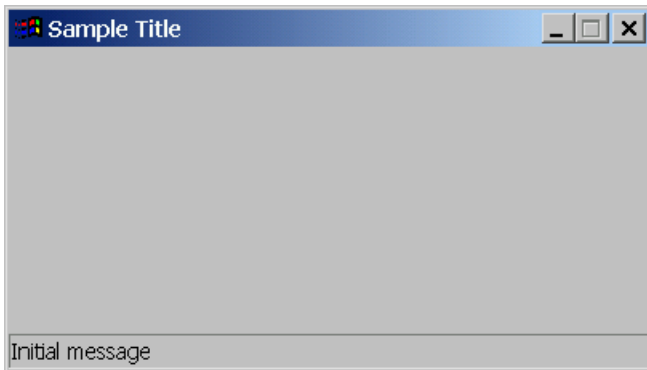
```
'STATUS_VISIBLE' SET_PROPERTY DIALOG 1
```

Notice the `STATUS_VISIBLE` property uses a different type of value. It requires a Boolean value; 1 means visible and 0 means invisible.

Use the `STATUS_DATA` property to set the message displayed in the dialog's status area:

```
'STATUS_DATA' SET_PROPERTY DIALOG 'Initial message'
```

Now the dialog looks like this:



Like the dialog's `DATA` property, the `STATUS_DATA` property requires a character vector value.

## Setting the Dialog's Event Handlers

The EVENTS property is used to specify the event handlers that should be executed in response to user actions. The EVENTS property is a 2-column matrix. For windows, each row specifies an event name and an event handler.

```
'EVENTS' SET_PROPERTY DIALOG (1 2ρ'Close' '→0')
```

This example specifies that when the Close event occurs, the event handler →0 should be executed. (The Close event occurs when the user closes the dialog.)

The event handler is a character vector and can be any APL2 expression. In this case, the branch to zero causes APL2 to stop waiting for events and return to the application.

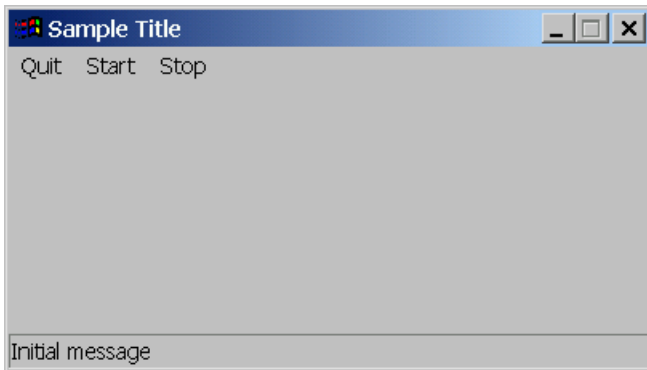
**Note:** Event handlers are usually character vectors. However, they can also be arbitrary arrays. For more information, see [Using Arbitrary Arrays as Event Handlers](#).

For a complete list of the events supported by each class of window, see [Class Reference](#).

## Adding a Menu Bar

A menu bar provides users with access to an application's commands. Each main application window should have a menu bar.

Use the `CREATEMENU` tool to add a menu bar to a dialog. The simplest menu bar is just a row of choices:

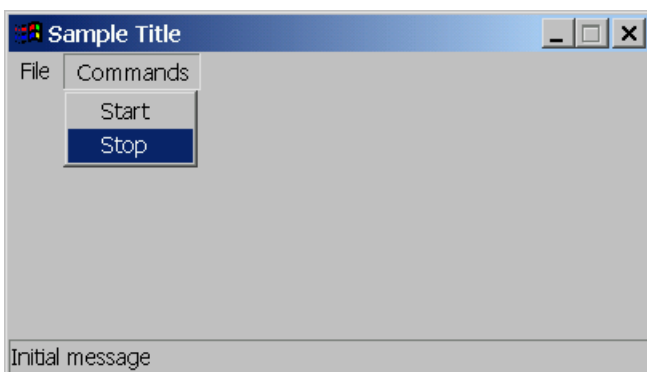


```
ARRAY←(11 'Quit') (12 'Start') (13 'Stop')  
MENU←CREATEMENU DIALOG ARRAY
```

The right argument of `CREATEMENU` is a dialog handle and an array of menu items. For the simple menu bar above, each menu item contains an integer and a character vector. The integers are used to identify the menu items when setting properties.

`CREATEMENU` returns a handle that identifies the menu. It is used to set properties for the menu and its menu items.

Menus can also contain pull-down menus. Selecting **Commands** in the window below displays a pull-down menu:



```
FILE←(0 'File') (11 'Quit')  
CMDS←(0 'Commands') (21 'Start') (22 'Stop')  
MENU←CREATEMENU DIALOG (FILE CMDS)
```

Notice in this case the array passed to `CREATEMENU` is two vectors of menu items. Each level of nesting adds a pull-down menu. The first item in each vector of menu items supplies the title of the pull-down menu. Subsequent items provide the pull-down menu's items.

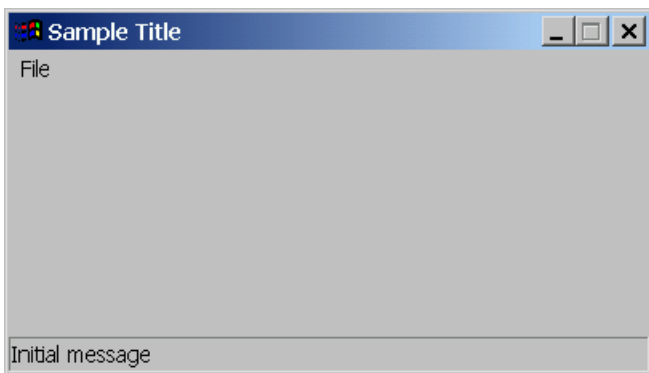
Menu items that do not display pull-down menus correspond to application actions and must be assigned unique positive integer identifiers less than or equal to 65535. Menu items that display pull-down menus do not correspond to application actions and their identifiers can be zero. In the example above, because the **File** and **Commands** choices display pull-down menus, their identifiers are both zero.

Menu items with identifiers 1 and 2 have special behavior. When the user hits the Enter key the event for menu item with identifier 1 is signaled if no control with identifier 1 exists. When the user hits the Escape key the event for menu item with identifier 2 is signaled if no control with identifier 2 exists.

Continuing with our dialog, here is how to add a menu bar with one pull-down menu. Notice the use of `enclose` to force creation of a pull-down menu:

```
MENU←CREATEMENU DIALOG (,c(0 'File')(11 'Quit'))
```

The dialog now has a File pull-down menu and looks like this:



For further information about creating menus, see [CREATEMENU](#) and the `DEMO_MENU` function in the `DEMO145` workspace.

### Setting the Menu's Event Handlers

The `EVENTS` property is also used to specify event handlers that should be executed in response to user selections of menu choices. The `EVENTS` property for menus is a 2-column matrix. Each row specifies a menu item identifier and an event handler.

```
'EVENTS' SET_PROPERTY MENU (1 2ρ11 '→0')
```

This example associates the event handler `→0` with the menu item that has identifier 11, in the example, the `Quit` item.

Menu items that display pull-down menus do not generate events. Do not specify event handlers for them.

## Adding Controls

Use the CREATECTL function to add control windows to dialogs:

```
TEXT←CREATECTL DIALOG 'TEXT' '' 101
MLE←CREATECTL DIALOG 'MLE' 'HSCROLL VSCROLL'
OK←CREATECTL DIALOG 'PUSH BUTTON' 'DEFAULT' 1
CANCEL←CREATECTL DIALOG 'PUSH BUTTON' '' 2
```

The CREATECTL function's right argument has 3, 4, or 5, elements:

- [1] A handle returned by CREATEDLG specifying the parent window.
- [2] A character vector containing the class of control to create.
- [3] A character vector containing a list of styles.
- [4] An optional integer scalar or character vector used to identify the control

In addition to a window handle, each control window can have an integer identifier and a character name. They can be used with the parent window's handle to refer to the control in calls to GUITOOLS functions. If an integer is supplied to CREATECTL, it is used as the control's identifier. If a character vector is supplied, it is used as the control's name. The name is case sensitive. The identifier defaults to 0 and the name defaults to null. The ID and NAME properties can be used to reference the control's identifier and name.

You can use identifiers of 1 and 2 or names of 'OK' and 'CANCEL' with push buttons to automatically assign the text Ok and Cancel to the buttons.

If the dialog will have only one visible control, you can use identifier 32776 or the name 'CLIENT' to make the control automatically fill the dialog's client area. This control is automatically resized as the dialog is resized.

When using any value other than 32776 or 'CLIENT', CREATECTL creates the control with a default size that is appropriate to display data using the dialog's current font. For example, push buttons are the correct height to display a line of text. This fact can be useful when resizing controls.

### [5] Control Data

Control data specifies the amount or range of data the control will contain. It is only used with grid and slider controls.

For detailed information about the supported control classes and styles, see [Class Reference](#).

CREATECTL returns a window handle that uniquely identifies the control. The handle can be used to set and reference the control's properties. If the control was created with a nonzero identifier, the parent window's handle and the identifier can also be used. For example, these two statements have the same effect:

```
SET_PROPERTY TEXT 'Sample Text'
SET_PROPERTY DIALOG 'Sample Text' 101
```

The example demonstrates that the text control's properties can be set using either the control's handle or the parent's handle and the control's identifier. Notice also that if no left argument is supplied to SET\_PROPERTY, the default property of DATA is set.

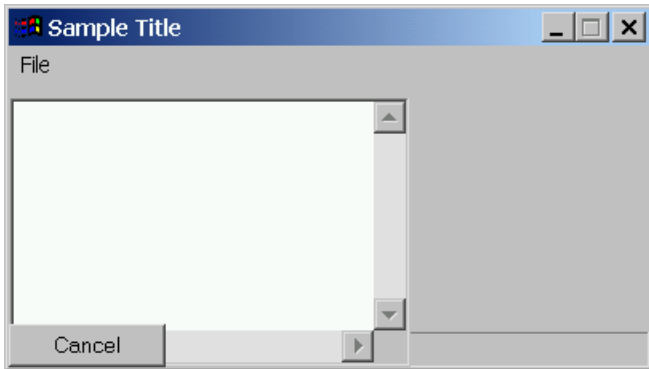
The WINDOWFROMID function can be used to retrieve the handle of a control from the parent window handle and the control's identifier or name:

```
      □←DIALOG←CREATEDLG 'TITLEBAR'  
16188708  
      □←TEXT←CREATECTL DIALOG 'TEXT' '' 'CTLNAME'  
13763616  
      WINDOWFROMID DIALOG 'CTLNAME'  
13763616
```



## Size, Position, and Client Area Properties

The `CREATEDLG` and `CREATECTL` functions create windows with default sizes and positions. `CREATECTL` positions all controls at the lower left corner of the parent so our example dialog now looks like this:



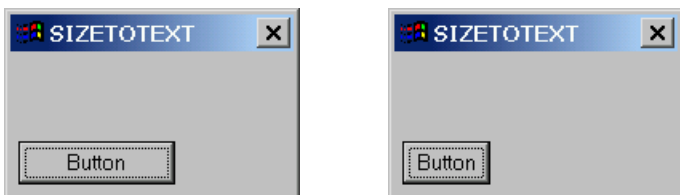
Notice that the controls overlap each other and need to be repositioned.

Windows can be resized and repositioned using the `SIZE` and `POSITION` properties.

The `SIZE` property is the width and height of the window in pixels.

```
'SIZE' SET_PROPERTY DIALOG (50 100)
```

The `SIZETOTEXT` function can be used to resize control windows so they are the correct size to display their data. For example, the pictures below show a button with the default size and after it has been resized with `SIZETOTEXT`.



```
BUTTON←CREATECTL DIALOG 'PUSH BUTTON' ''  
SET_PROPERTY BUTTON 'Button'  
SIZETOTEXT BUTTON
```

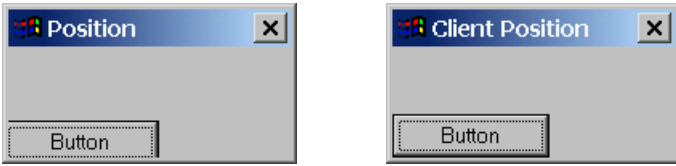
The `POSITION` property is two integers that indicate the distance in pixels of the window's lower left hand corner from the lower left hand corner of the parent window.

```
'POSITION' SET_PROPERTY DIALOG (400 200)
```

If the parent is the desktop (the default for dialogs), the distances are from the lower left hand corner of the screen.

It is often useful to position a window relative to its parent's client area. The client area is the area within the parent's borders, menu, status area, and scroll bars. The `CLIENT POSITION` property is the position of the

window relative to the lower left hand corner of the parent's client area. For example, compare these two images:



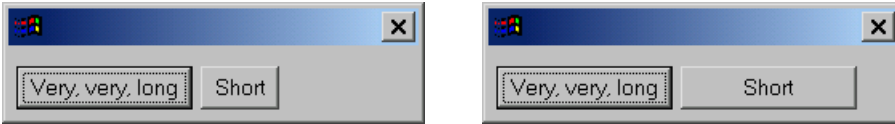
The left button was positioned using the POSITION property; the right with the CLIENT POSITION property:

```
'POSITION' SET_PROPERTY BUTTON (0 0)  
'CLIENT POSITION' SET_PROPERTY BUTTON (0 0)
```

## Arranging Controls

Although the CLIENT POSITION and SIZE properties can be used to arrange controls, the RESIZE, ALIGN, and SPACE functions make it easy.

RESIZE resizes controls so they have the same width or height. The position of the controls is not changed. For example, consider these two pictures:



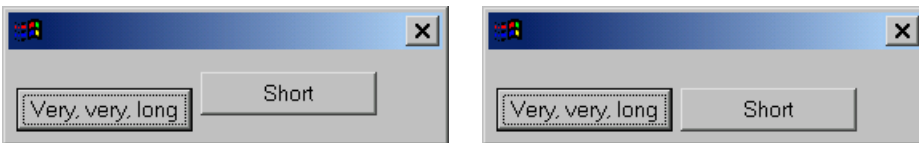
The two buttons were made the same width with this statement:

```
0 RESIZE BUTTON1 BUTTON2
```

The left argument of RESIZE is a Boolean scalar. A value of 0 indicates the controls should be made the same width; 1 indicates they should be made the same height.

The right argument is a list of control handles. All the controls are made the same width or height as the first control.

The ALIGN function aligns a list of controls. The size of the controls is not changed. Here are another two pictures:



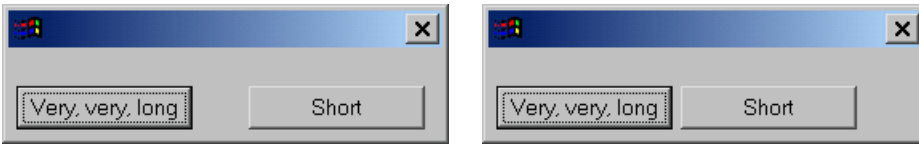
The two buttons were aligned along the bottom edge using this statement:

```
0 0 0 1 ALIGN BUTTON1 BUTTON2
```

The left argument of ALIGN indicates which edges of the controls should be aligned. It is a Boolean vector containing 4 elements corresponding to the left, top, right, and bottom edges. 1 should be specified for the edges that should be aligned. (1 should not be specified for two opposite edges.)

The right argument is a list of control handles. The controls are aligned along the specified edges of the first control in the list.

SPACE repositions a list of controls so that they are evenly spaced. The size of the controls is not changed. Here are still two more pictures:



The two buttons were repositioned so they are 5 pixels apart with this statement:

```
0 5 SPACE BUTTON1 BUTTON2
```

The left argument of SPACE is an integer vector containing 2 elements. The first element is 0 or 1. 0 indicates the controls should be arranged horizontally; 1 indicates they should be arranged vertically. The second element indicates how far apart the controls should be spaced. The distance is expressed in pixels.

The right argument is a list of control handles. The position of the first control is used as the starting point for positioning the other controls.

After rearranging controls, it is usually necessary to resize the dialog so it fits the rearranged controls. The CLIENT SIZE property can be used to resize a dialog so its client area is a specific size.

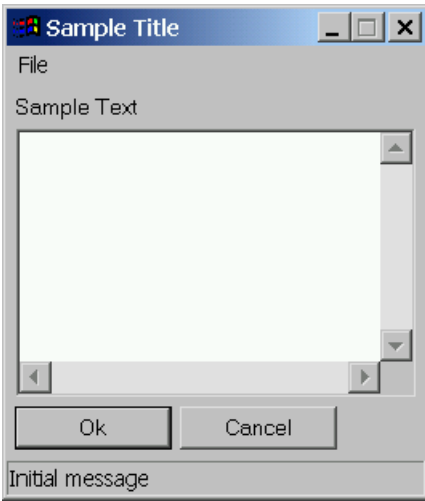
```
'CLIENT SIZE' SET_PROPERTY DIALOG (200 200)
```

The CLIENT SIZE property is the width and height of the window's client area in pixels. Usually, you calculate the position of the right most and top most control edges and add a few pixels for spacing.

Going back to our earlier example, here is how to arrange the Text, MLE, Ok, and Cancel controls and resize the dialog to fit them:

```
SIZETOTEXT TEXT
'CLIENT POSITION' SET_PROPERTY OK (5 5)
0 0 0 1 ALIGN OK CANCEL
1 0 0 0 ALIGN OK MLE TEXT
0 5 SPACE OK CANCEL
1 5 SPACE OK MLE TEXT
WIDTH←+/1>'CLIENT POSITION' 'SIZE' GET_PROPERTY''MLE
HEIGHT←+/2>'CLIENT POSITION' 'SIZE' GET_PROPERTY''TEXT
'CLIENT SIZE' SET_PROPERTY DIALOG (WIDTH HEIGHT+5)
```

And here's how the dialog appears after being rearranged:



## The Desktop

Finally, it is nice to position the dialog on the screen. To position the dialog, you need the screen size. Use 1 for the handle of the desktop and query the `SIZE` property:

```
SCREEN←'SIZE' GET_PROPERTY 1
```

You can then use the screen size to calculate where to position a dialog. For example, these statements will center a dialog:

```
SCREEN←'SIZE' GET_PROPERTY 1  
SIZE←'SIZE' GET_PROPERTY DIALOG  
POSITION←⌊ (SCREEN-SIZE) ÷ 2  
'POSITION' SET_PROPERTY DIALOG POSITION
```

Notice that floor is used because the `POSITION` property requires integer values.

## Focus, Dialog Navigation, Groups, and Tab Stops

One control at a time receives keyboard input. The control receiving keyboard input is said to have the focus. Only controls that accept input can have the focus.

The user can move focus between a dialog's controls using either the mouse or the keyboard. A well-designed dialog should make it easy for the user to navigate the dialog's controls using the keyboard.

By default, the Tab key moves focus between a dialog's controls in the order they were created. Therefore, create controls in an order so that pressing the Tab key moves the focus between the controls from top to bottom and left to right. Use the NOTABSTOP style to prevent the Tab key from moving focus to a control.

You can group controls so that the Arrow keys move focus between the controls in the group. By default, all controls begin a new group. To create a group with multiple controls, create the first control in the group normally. Then, create all the other controls in the group with the NOGROUP style. Also, use the NOTABSTOP style. With these settings, the user can tab between the groups and use the Arrow keys to move focus between the controls within the groups.

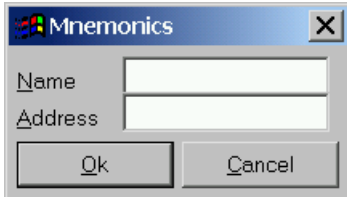
For example, adding the NOGROUP and NOTABSTOP styles to the Cancel button in the example would make the Arrow keys move the focus between the two buttons and the Tab key move between the MLE and the Ok button.

```
CANCEL←CREATECTL DIALOG 'PUSH BUTTON' 'NOGROUP NOTABSTOP' 2
```

## Mnemonics

In addition to the Tab and Arrow keys, Mnemonics also aid dialog navigation. A mnemonic is a character in a button or static control's data. When the user presses the character, or Alt and the character, the dialog moves the focus to the control associated with the mnemonic.

Create a mnemonic by inserting an ampersand (&) immediately before the selected character in the text of the control. For example, here is a dialog that uses mnemonics:



The mnemonics were set like this:

```
SET_PROPERTY LNAME '&Name '  
SET_PROPERTY LADDR '&Address '  
SET_PROPERTY OK '&Ok '  
SET_PROPERTY CANCEL '&Cancel '
```

When the user presses Alt+N, focus moves to the Name entry field. When the user presses Alt+A, focus moves to the Address entry field. When the user presses Alt+O, the Ok button is pushed. When the user presses Alt+C, the Cancel button is pushed.

When the user presses a key, if the control that has focus can process the key, the dialog simply sends the keystroke to the control. In the example above, if an entry field has focus and the user simply presses a character, the dialog sends the character to the entry field.

If the control that has the focus cannot process the keystroke, the dialog processes it as a mnemonic. In the example above, if an entry field has focus and the user presses Alt and a character, the entry field cannot process the keystroke combination and the dialog processes it as a mnemonic. If a button has focus, both a character and Alt plus a character are processed as mnemonics since buttons do not accept character input.

Dialogs process mnemonics as follows:

1. The dialog locates the control containing the mnemonic.
2. If the control is a push button, the button issues a Command event.
3. If the control with the mnemonic does not accept input, the dialog moves the focus to the next visible enabled control that does not have the NOTABSTOP style.

Mnemonics also aid menu navigation. Create a menu mnemonic by inserting an ampersand (&) immediately before the selected character in the text of a menu item. When the user presses Alt, the focus moves to the menu. The user can then press a mnemonic character to select a menu choice. Mnemonic characters on menus may not appear until the user presses Alt. Users can control when mnemonic characters appear by setting the

**Hide keyboard navigation indicators until I use the Alt key** checkbox on the **Effects** page of the **Windows Display Properties** notebook.



## Making Dialogs Resizable

APL2 provides several facilities for building resizable dialogs.

The SIZEBORDER dialog style enables a dialog to be resized. The MIN and MAX dialog styles add minimize and maximize buttons to the dialog's titlebar.

The MINIMUM SIZE and MAXIMUM SIZE properties are used to specify the limits to which a dialog can be resized. Typically, the MINIMUM SIZE property is set to the minimum size that displays the critical features of a dialog. The MAXIMUM SIZE property is set to the screen size. For example:

```
'MINIMUM SIZE' SET_PROPERTY DIALOG (400 200)
'MAXIMUM SIZE' SET_PROPERTY DIALOG SCREEN
```

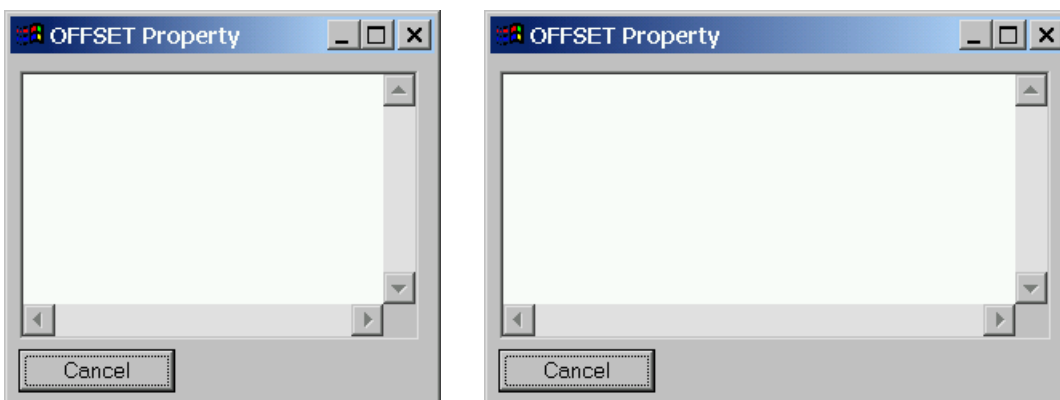
The STATE MINMAX property can be used to set or determine whether a dialog has a normal size or is maximized or minimized.

```
'STATE MINMAX' SET_PROPERTY DIALOG 0  A Normal
'STATE MINMAX' SET_PROPERTY DIALOG 1  A Maximized
'STATE MINMAX' SET_PROPERTY DIALOG 2  A Minimized
```

When a dialog is resized, the controls within it usually need to be repositioned and resized so they maintain their relative positions within the dialog. For a dialog with a single control, the control can be given the identifier of 32776 and it is automatically resized to fit the dialog's client area. For dialogs with multiple controls, the OFFSET property is used.

The OFFSET property is set for each control within a resizable dialog. Its value is four integers that specify the percentage of the change in the dialog's size that is added to the position of the control's edges.

For example, examine the two images below. They are both pictures of the same dialog. In the right image, the dialog has simply been enlarged by dragging the right edge of the dialog to the right.



Notice that the dialog contains two controls. As the dialog is resized, the push button remains the same size and at the same position in the lower left hand corner of the dialog. However, the MLE's width has changed so that its right edge remains the same distance from the dialog's right edge.

Consider what happened when the right edge of the dialog was dragged to the right. The dialog's width was increased by a specific amount. In order for the MLE's right edge to remain the same distance from the dialog's right edge, the width of the MLE had to change by the same amount that the dialog's width changed.

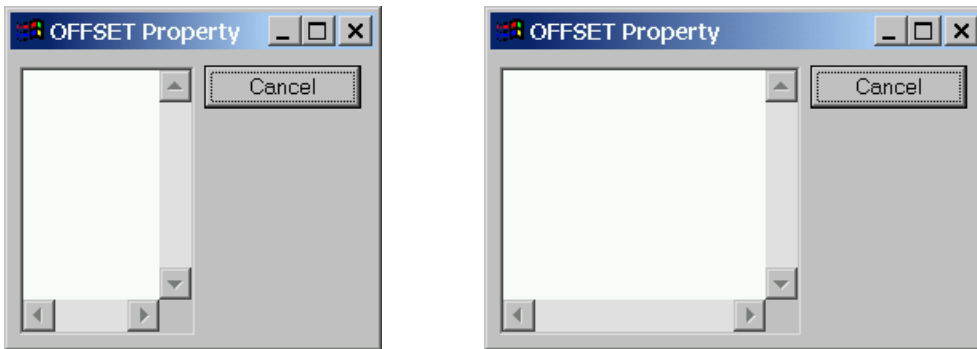
The following statements specified this behavior:

```
'OFFSET' SET_PROPERTY BUTTON (0 0 0 0)
'OFFSET' SET_PROPERTY MLE     (0 100 100 0)
```

The OFFSET property specifies the percentage of the change in the dialog's size that is added to the position of each of a control's left, top, right, and bottom edges. In the example above:

- The button's position and size remain the same as the dialog is resized so 0% of the change was added to the position of the button's edges.
- The MLE's edges all stay the same distance from the edges of the dialog. So 100% of the change in the dialog's height is added to the MLE's top edge and 100% of the change in the dialog's width is added to the MLE's right edge.

Consider another two images:



This is the same dialog with the position of the two controls reversed. Consider what happened when this dialog was resized

The MLE's behavior remains the same. The right edge still maintains the same distance from the dialog's right edge.

The button's behavior is quite different though.

The right edge of the button now behaves like the right edge of the MLE. It maintains the same distance from the right edge of the dialog.

And, the left edge of the button also maintains the same distance from the right edge. In other words, when the dialog was resized, the amount of the change was added to the position of both the button's left and right edges.

The following statements specified the new behavior:

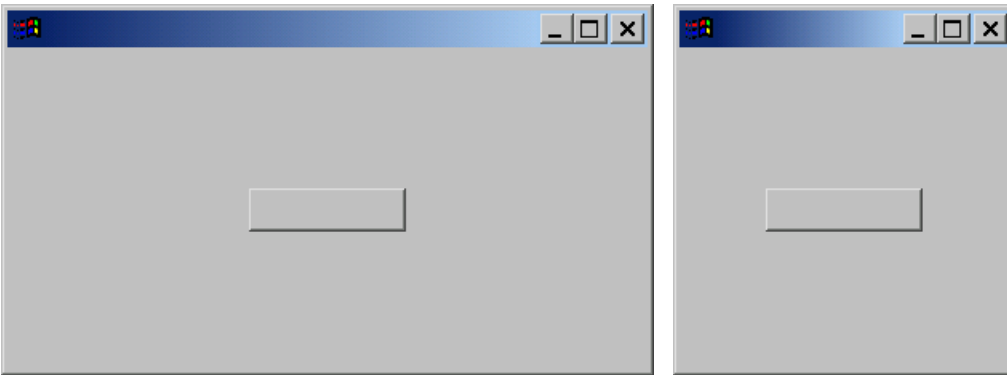
```
'OFFSET' SET_PROPERTY MLE     (0 100 100 0)
'OFFSET' SET_PROPERTY BUTTON (0 0 0 0)
```

```
'OFFSET' SET_PROPERTY BUTTON (100 100 100 100)
```

The OFFSET property still specifies the percentage of the change in the dialog's size that is added to the position of each of the controls' left, top, right, and bottom edges. In this example:

- The MLE's edges all stay the same distance from the edges of the dialog. So again, 100% of the change in the dialog's height is added to the MLE's top edge and 100% of the change in the dialog's width is added to the MLE's right edge.
- The button's position and size remain the same relative to the dialog's upper right hand corner. Both the left and right edges of the button remained the same distance from the dialog's right edge and both the top and bottom edges remained the same distance from the dialog's top edge. As the dialog was resized, 100% of the change was added to the position of the button's four edges.

Consider these two images:



The dialog was created with these statements:

```
DIALOG←CREATEDLG 'SIZEBORDER MIN MAX'  
BUTTON←CREATECTL DIALOG 'PUSH BUTTON' ''  
CENTER_CHILD BUTTON  
'OFFSET' SET_PROPERTY BUTTON (50 50 50 50)
```

The CENTER\_CHILD function was used to center the button in the dialog's client area. The OFFSET property was used to maintain the button's position in the center of the dialog.

The OFFSET property is very powerful. It can be used to maintain controls' positions relative to any of the edges of a dialog. For further information about using the OFFSET property, examine the DEMO\_RESIZE function in the DEMO145 workspace.

## Providing Online Help

You should strive to design dialogs that are easy to understand and use. To help your users understand your application, you can provide online help through the TOOL TIP and CONTEXT HELP properties.

The TOOL TIP property is a character vector that contains a one-line description of the purpose of a control. The tip is displayed when the user moves the mouse pointer over the control and leaves it there for a moment.

```
'TOOL TIP' SET_PROPERTY CONTROL 'Tool tip text'
```

The CONTEXT HELP property is a vector of character vectors and provides several lines of descriptive text about the purpose of a control or group of controls.

```
'CONTEXT HELP' SET_PROPERTY CONTROL ('Context' 'Help')
```

Setting the CONTEXT HELP property adds a question mark button to the dialog's titlebar. The user requests context help by using any of the following methods:

- Pressing the question mark button and then clicking on a control
- Right clicking on a control and selecting **What's This?** from the popup menu
- Giving focus to an input control and then pressing F1

Note: The question mark button is not added to the titlebar if the dialog contains a maximize button and it does not work if the dialog has either a maximize or minimize button. Do not use the Min and Max styles with dialogs that provide context help.

The system automatically displays the appropriate contextual help when the user performs any of these actions. Use the [CONTEXTHELP](#) function to display contextual help in response to other events.

To display the same contextual help text for all the controls in a group, the CONTEXT HELP property can be set for only the first control in the group. For example, this statement could be added to the example at the end of the [Arranging Controls](#) section:

```
'CONTEXT HELP' SET_PROPERTY OK ('CONTEXT' 'HELP')
```

When the user requests help for a control, if the control does not have context help and has the NOGROUP style, the system displays the context help for the first control in the group.

Users can control the font used for tool tips and context help by setting the **ToolTip** item on the **Appearance** page of the Windows **Display Properties** notebook.

## Fonts

Use the FONT property to change how a control's data is displayed. The FONT property is a character vector containing the font's size and face name. The size and face name are separated by a period. The size is expressed in points. A point is 1/72 inch.

```
DIALOG←CREATEDLG 'SIZEBORDER'  
BUTTON←CREATECTL DIALOG 'PUSH BUTTON' ''  
SET_PROPERTY DIALOG 'Font Sample'  
SET_PROPERTY BUTTON 'Sample Text'  
'FONT' SET_PROPERTY BUTTON '16.Cooper Black'  
SIZETOTEXT BUTTON
```



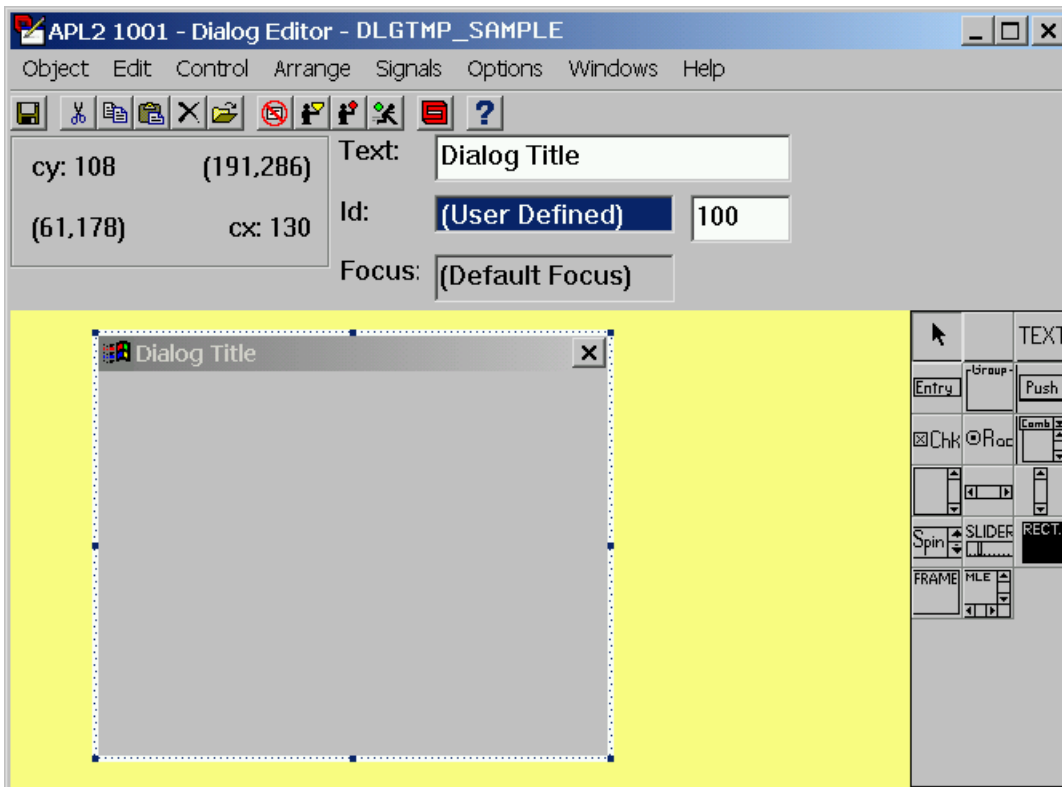
Notice that the SIZETOTEXT function uses the control's current font.

If no font is explicitly set for a control, it will use the same font as its parent. To set the font for all the controls in a dialog, set the dialog's FONT property before creating the controls.

For portability with older systems, AP 145 uses a default font that is different than the Windows default. To use the Windows default, use MS Shell Dlg as the face name.

## Building Dialogs using the Dialog Editor

The Dialog Editor is a graphical tool for designing dialogs. Here is a picture of a dialog editor window:



To open the Dialog Editor, select the **Open Object** option in the **Edit** pull-down menu. Select the **Type of Objects** choice **Variables**, check the **Dialog Template** option, type the template's name, and push **Ok**. You can also open an existing dialog template in any Session Manager or Object Editor window by double-clicking on the template's name.

The Dialog Editor includes extensive online help. Access the online help by selecting a choice from the **Help** menu or by pressing F1.

The Dialog Editor saves dialog designs in arrays called dialog templates. A dialog template can be used as the right argument of `CREATEDLG` to create a dialog and all its child controls. Dialog templates are saved in the workspace as character vectors.

## Starting a New Dialog

When you open a new dialog template, the Dialog Editor window appears and an empty dialog appears in the lower-left corner of the screen. A dotted border encloses the dialog. The dotted border indicates the dialog is selected for editing. Information about the selected dialog or control is displayed in the dialog editor's status area.

Change the dialog's title by typing in the **Text** entry field in the editor's status area.

The dotted border of the selected dialog or control includes drag handles at the corners and middles of the edges. Resize the dialog by dragging the drag handles. Move the dialog by dragging the titlebar or pressing the Arrow keys. Notice that as you resize and move the dialog, the values in the dialog editor's status area change. These values indicate the dialog's position and size.

Set the dialog's styles by double clicking on the dialog or selecting **Styles** from the **Edit** menu. This opens the **Dialog box styles** window. Press **Help** to learn about the different styles. Style changes will be visible when you finish and press **OK**.

## Adding and Deleting Controls

The **Control** menu lists all the classes of controls that you can put in a dialog. Many of the classes of controls are also represented in the control palette on the right hand side of the dialog editor's window.

Add a control by selecting a control, either from the Control menu or the control palette. When you select a control, the mouse pointer becomes a small plus sign (+). Position the pointer where you want the control and press mouse button 1. Release the button immediately to create a control with a default size. Drag the mouse before releasing the button to set the size as you create the control.

For grid and slider controls, a dialog will appear in which you must enter initialization data for the control. Complete this dialog and press **Ok**.

Select a control by clicking on it. The selected control, or the dialog if no control is selected, is enclosed in a dotted border. The dotted border indicates the window is selected. The Dialog Editor's status area displays information about the currently selected window. The choices on the menus affect the currently selected window.

Some types of controls support initial data. Type in the status area's **Text** entry field to set these controls' initial data.

Resize the selected control by dragging the corners and edges of the dotted border. Move the control by dragging the control or pressing the Arrow keys. Notice that as you resize and move the control, the values in the dialog editor's status area change.

The Dialog Editor automatically assigns identifiers to controls as they are created. The first control has identifier 101, the second has 102, and so on. Identifiers are used with the dialog's handle to refer to the control when using functions from GUITOOLS.

The selected control's identifier is displayed in the status area. Change a control's identifier by typing an integer in the **Id** entry field. Each non-zero identifier must be unique.

Several common identifiers can be accessed using the **Id** combo box. Click in the combo box and press the up and down arrow keys to view the common identifiers. `DID_OK` and `DID_CANCEL` are usually used for push buttons, `(Unused)` is used for non-input controls, and `FID_CLIENT` is used to make a control fill the dialog's client area.

See [Adding Controls](#) in the previous chapter for more information about control identifiers.

For detailed descriptions of individual controls and how they work, see the individual controls in the [Class Reference](#) or in the Dialog Editor's online help.



## Selecting and Arranging Controls

Select multiple controls by clicking on the dialog background and dragging the selection rectangle to enclose the desired controls. The selected controls are enclosed in dotted borders.

To move the selected control(s), use the arrow keys or click and drag any the control(s).

The **Arrange** menu allows you to arrange and align controls.

<b>Align</b>	Allows you to align controls along an edge
<b>Even spacing</b>	Allows you to evenly space controls
<b>Same size</b>	Allows you to set controls to the same size
<b>Push buttons</b>	Allows you to arrange push buttons
<b>Order groups</b>	Allows you to change the order of controls and groups.
<b>Settings</b>	Displays the Arrange Settings dialog, which allows the grid and spacing constants to be changed.

Notice that when multiple controls are selected, one of the controls has filled drag handles; this is the anchor control. The anchor control's information is displayed in the status area. The **Align**, **Even spacing**, and **Same Size** options arrange controls relative to the anchor control. Select which control is the anchor control by holding down Shift and clicking on a control.

## Controlling the Arrange Settings

Use the **Arrange Settings** dialog to specify how the dialog editor adjusts positions of dialog boxes and controls as you move them.

When you move dialog boxes and controls, the dialog editor adjusts the objects' positions so they are exact multiples of the **Grid** settings. Large grid settings make it easier to align controls, while small grid settings allow you to position controls more precisely.

When you use the choices on the **Even Spacing** menu, the dialog editor uses the **Control Spacing** settings.

When you use the choices on the **Push Button** menu, the dialog editor uses the **Margins** and **Push Button Spacing** settings.

Dialog and control positions and sizes and the **Arrange Settings** values are specified in dialog units. These units are device independent so that dialog boxes will have the same proportions and appearances on all displays despite different resolutions and aspect ratios.

Dialog units are fractions of the average size of the characters in the dialog's font. To specify the dialog's font, select the dialog and open the **Presentation Parameters** dialog. If no font is specified, the dialog editor uses the average size of the characters in the default font.

A horizontal dialog unit is one quarter of the average character width. A vertical dialog unit is one eighth of the average character height. For example, if you move a control to the left or the right (using the mouse or keyboard arrow keys) with the Grid horizontal value set at 20, it moves in steps of twenty dialog units or the width of 5 characters.

To avoid rounding errors and receive the most accurate results, specify settings that are whole multiples of a character. For example, using a horizontal value of 4 and a vertical value of 8 would produce a grid with cells the size of one character.

## Using Menus and the Status Area with Dialog Templates

The dialog editor does not provide built-in support for adding menu bars or displaying dialogs' status areas. When you position controls in the dialog editor, you should leave room at the top and bottom of the dialog if you intend to add a menu or use the status area in your dialog.

See [Using the Status Area](#) and [Adding a Menu Bar](#) for information about using these features.

## Ordering Controls

By default, the order in which you added the controls to the dialog determines the order in which focus moves between the controls when the user presses Tab. The **Order groups** option on the **Arrange** menu allows you to change the order in which the Tab and Arrow keys move focus between the controls.

### Notes:

- The position of controls in a dialog box does not affect the selection order.
- When you use group boxes to group controls, always create the group box before the controls that are to go inside it.

Selecting **Order groups** numbers the dialog's controls. The numbers indicate the order in which focus moves between the controls. Initially, the controls are numbered in the order in which they were created. To change the order, click on the controls in the order the cursor should move between them. To stop ordering the groups, click on the dialog's background area.

See [Focus, Dialog Navigation, Groups, and Tab Stops](#) for additional information.

## Event Handlers

The Event Handlers dialog allows you to associate an APL event handler expression with an event generated by the selected control or dialog.

The events for which you can specify event handlers are shown in the **Posted event** drop down combo box. The expression associated with the currently selected event is displayed in the **Handler** entry field.

To set up an APL event handler for an event, follow these steps:

1. Select a control or the dialog box.
2. Select **Event Handlers** on the **Edit** menu.
3. Select the event for which you would like to set up an event handler.
4. Type the APL expression that should be executed when the event occurs.

**Note:** Event handlers are usually character vectors. However, when the EVENTS property is set dynamically, event handlers can also be arbitrary arrays. The Event Handlers dialog can not be used to set arbitrary array event handlers. For more information, see [Using Arbitrary Arrays as Event Handlers](#).

## Changing a Dialog or Control

To change the properties of a dialog box or a single control, use the functions of the **Edit** menu.

The **Edit** functions require that you first select a control to edit. Once you select a control,

- Select **Cut** to copy the control to the clipboard and remove it from the dialog.
- Select **Copy** to copy the control to the clipboard.
- Select **Paste** to place a control you have copied to the clipboard with **Cut** or **Copy**.
- Select **Clear** to delete a control.
- Select **Duplicate** to create another control in the dialog identical with the selected control.
- Select **Styles** to define the style of the dialog or the selected control.
- Select **Presentation parameters** to select color and font.
- Select **Size to text** to adjust a control's size to fit its text.
- Select **Context help** or **Tool tip** to type contextual help and tool tips.
- Select **Event Handlers** to associate APL expressions with events generated by the dialog or control.

## Providing help to your users

While running your application, the user sometimes needs help. For example, the user may need assistance in recalling the use of a push button, making a choice among the items in a list box, or understanding what data is valid for an edit field.

Using the **Context help** and **Tool tip** functions of the **Edit** menu, you can provide your users with tool tips and contextual help for your dialogs' controls.

A tool tip is a single line of descriptive text that appears in a small popup window when the mouse pointer pauses over a control. To set a tool tip for a control, follow these steps:

1. Select a control.
2. Select **Tool tip** on the **Edit** menu.
3. Type the tool tip you want displayed.

Contextual help is descriptive text that appears in a popup window when the user requests help. Contextual help can be several paragraphs long and displays in a window roughly 40 characters wide.

To set the contextual help for a control, follow these steps:

1. Select a control.
2. Select **Context help** on the **Edit** menu.
3. Type the contextual help you want displayed.

Setting the context help for a control adds a question mark button to the dialog's titlebar.

The user can request contextual help in three ways:

1. Clicking first on the question mark button and then on a control.
2. Clicking first on a control with the right mouse button and then selecting **What's this?**
3. Tabbing to a control and pushing F1.

Note: The question mark button is not added to the titlebar if the dialog contains a maximize button and it does not work if the dialog has either a maximize or minimize button. Do not use the Min Button and Max Button dialog styles in dialogs that provide context help.

## Testing the Dialog

A good way to learn about the different types of windows and the effect of style settings on their appearance and behavior is to test them.

To test the dialog box, select **Test Mode** from the **Options** menu. The dialog box is displayed as it appears to the user from a program. In test mode, you can select and tab between controls, and their appearance changes in the same way as they do in an application. To return to work mode, click on **Test Mode** again to de-select it.

**Note:** The Dialog Editor's Test option does not execute APL event handlers. It merely displays the dialog as it would appear when run as part of an application.



## Using Dialog Templates

Use the **Save** or **Save and Close** choice on the **Object** menu to save the design into a dialog template.

Use the `CREATEDLG` function to create a dialog from a dialog template:

```
DIALOG←CREATEDLG DLGTMP
```

Use the dialog handle just as you would the handle of a dynamically created dialog.

## ***Dialog Box Design Guidelines***

The following are a few guidelines for designing dialog boxes:

- Clearly identify the information that the user is required to complete.
- Make dialogs initially invisible so initialization is not visible.
- Arrange the controls in the sequence in which the user would complete them.
- Arrange the controls in columns, starting at the upper-left corner, for left-to-right and top-to-bottom scanning.
- Align input controls horizontally and vertically so that the cursor moves in a straight line.
- If there are only a few entry fields, locate them at the top of the dialog box.
- Do not use the Max dialog style unless you also have a sizing border. The maximize button only positions the dialog, it does not size it unless you have a sizing border.
- Do not use the Max and Min dialog styles if you provide context help.
- Make groups of controls obvious by use of group boxes and white space.
- Align group boxes, where possible.
- Use the Tabstop and Group styles so that the Tab key moves focus between groups of controls and the Arrow keys move focus between controls within groups.
- Include a Cancel button with identifier of `DID_CANCEL` in every dialog. Make the Cancel button have the same event handler as the dialog's Close event. This is because by default, when the user presses Escape, Windows issues a Command event with an identifier of `DID_CANCEL` (2). The button enables you to respond to this event.
- If you have visible buttons, one of them should have the **Default** style. The default button will be pushed when the user hits **Enter**. This button should have an identifier of `DID_OK` (1).
- Set the initial focus. To set the initial focus in the dialog editor, double click with mouse button 2 on the appropriate control. The Focus field of the status area displays the identifier of the control with the initial focus. Use the `STATE_FOCUS` property to set the focus dynamically.
- Test all dialogs designed with the Dialog Editor on VGA displays. Dialogs templates use a coordinate system based on dialog units. Different operating systems use different fonts to define the sizes of dialog units. If you design dialogs on VGA displays, they will work on all other resolution displays.

## Executing Dialogs

Once you have created an application window, the EXECUTEDLG operator is used to wait for events and execute event handlers:

```
DEFAULTPROC EXECUTEDLG DIALOG
```

The EXECUTEDLG operator's right operand is the handle of a dialog. EXECUTEDLG will make the dialog visible, activate it, and wait for events. When an event occurs for which an event handler has been defined, EXECUTEDLG executes the event handler. If the event handler branches to zero, EXECUTEDLG will return. Otherwise, EXECUTEDLG will continue to wait for more events.

The EXECUTEDLG left operand is a function to process registered messages. Messages were registered on older systems. Registered messages are no longer used. Always code DEFAULTPROC or zero as the left operand.

EXECUTEDLG is defined so that it can return an explicit result named RESULT. However, EXECUTEDLG does not set RESULT. Event handlers can set the variable RESULT to cause EXECUTEDLG to return a value. For example:

```
D←CREATEDLG ''
B←CREATECTL D 'PUSH BUTTON' ''
'EVENTS' SET_PROPERTY D (1 2ρ 'CLOSE' 'RESULT←0 ◇ →0')
'EVENTS' SET_PROPERTY B (1 2ρ 'COMMAND' 'RESULT←1 ◇ →0')
RESULT←0 EXECUTEDLG D
```

If no event handler sets RESULT, EXECUTEDLG does not return an explicit result.

Note that referencing the result of EXECUTEDLG will yield VALUE ERROR if RESULT has not been set. Whenever writing code to use the result of EXECUTEDLG, care should be taken to ensure that RESULT is always set by the time EXECUTEDLG returns.

## Getting User Input

Event handlers are executed in response to user actions. An event handler typically retrieves some user input and processes that input. Use the `GET_PROPERTY` function to retrieve user input. For example, in the following dialog,



The event handler for the Ok button would probably need to retrieve the contents of the entry field:

```
GET_PROPERTY ENTRY  
John Doe
```

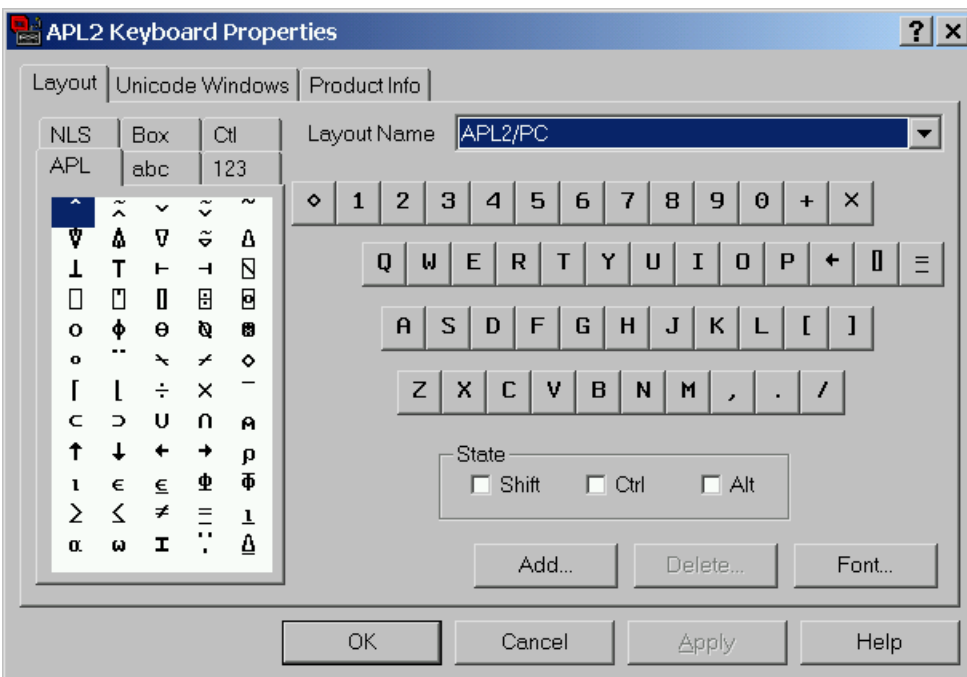
## Supporting APL Input

Use an APL font and the STATE APL property to enable APL characters in keyboard input.



```
ENTRY←CREATECTL DIALOG 'ENTRY FIELD' ''  
'STATE APL' SET_PROPERTY ENTRY 1  
'FONT' SET_PROPERTY ENTRY '10.APL2 Image'
```

Display the APL2 Keyboard Properties dialog by posting an APLKEY\_MSG\_MODIFY message to any window for which the STATE APL property has been set.



Use the GUITOOLS function POSTMSG and the APLKEY\_MSG\_MODIFY variable from the GUIVARS workspace:

```
POSTMSG ENTRY APLKEY_MSG_MODIFY 0 0
```

Provide a menu item to display the Keyboard Properties notebook if you want your users to be able to modify the APL2 keyboard layouts. For example, the APL2 Session Manager's Keyboard Properties system option displays the notebook.

## Sharing Variables with Window Properties

The `SET_PROPERTY` and `GET_PROPERTY` functions are usually used to set and get window properties. It is also possible to share a variable directly with a window property. Once a variable is shared with a property, setting the variable changes the property and referencing the variable retrieves the property's value. This can make it easier to write APL2 code when the application needs to set or get a property's value very often. Use `SHAREWINDOW` to share a variable with a window's property:

```
'DATA' SHAREWINDOW DIALOG 'TITLE'  
TITLE←'Sample Title'
```

The left argument of `SHAREWINDOW` is a property name. It defaults to `DATA`.

The right argument of `SHAREWINDOW` has 2 or 3 elements. The first element is a window handle and the second element is the name of the variable to share with the property. The optional third argument is the identifier of the child window with whose property the variable is to be shared.

Notes on using `SHAREWINDOW`:

Variables shared with properties are not automatically retracted when the window is destroyed. These variables should be localized or explicitly retracted when the application terminates. Use of `SHAREWINDOW` with a variable name that is already shared (for example from a previous execution of the application) will fail.

`GET_PROPERTY` and `SET_PROPERTY` provide significantly better performance than `SHAREWINDOW` when accessing a property just a few times. `SHAREWINDOW` should only be used when a property will be referenced or specified many times.

## ***Building More Complex Applications***

Most applications use several dialogs in addition to the main dialog. Use additional dialogs to display a message or to obtain a specific piece of information such as the name of a file to open. You can also imbed dialogs in dialogs to more easily arrange information. This section discusses the procedures for creating and managing multiple dialogs.

The simplest way to display another dialog is to use the MSGBOX function:

```
RC←DIALOG MSGBOX 'TITLE' 'MESSAGE'
```

The MSGBOX function creates and displays a message box. A message box is a simple dialog with a titlebar, some text, buttons, and optionally an icon. By default MSGBOX creates a message box with Ok and Cancel buttons. It automatically resizes the dialog to fit the message. It returns 1 if the box was closed with the Ok button and 2 if it was closed with the Cancel button (or the box's Close button.)

The right argument to MSGBOX contains two character vector elements: a title for the message box, and the text of the message. The text may contain linefeed characters to begin new paragraphs.

The left argument of MSGBOX is the handle of the message box's owner. If MSGBOX is used in an application with other windows, a dialog handle should always be supplied. Otherwise, the application may be closed while the message box is still on the screen. To understand why, read the next section about Owned, Modal, and Modeless Windows.

For information about more sophisticated uses of the MSGBOX function see [Message Boxes](#).

## Owned, Modal, and Modeless Windows

A dialog may own another dialog. A dialog that is owned has several special properties:

- An owned dialog is always displayed on top of its owner.
- An owned dialog is hidden when its owner is minimized.
- An owned dialog is automatically destroyed when its owner is destroyed.

As an example, the APL2 Session Manager's Find dialog is an owned window. The Session Manager or an Object Editor window owns the Find dialog. Owned dialogs are used when there is a logical connection with the owner window.

MSGBOX displays an owned dialog that also is modal. When a modal dialog is displayed, its owner is disabled. When the modal dialog is closed, the owner is enabled. For example, the APL2 Session Manager's Log Size dialog is a modal dialog.

Modal dialogs force the user to stop using the owner window until the modal dialog is closed. Modal dialogs are used for messages and to prompt for information the application needs in order to continue. The MSGBOX function should always be passed a dialog handle so that it can disable the dialog and prevent the application from being closed until the message box is closed.

Windows that are not modal are called modeless. Modeless windows do not disable other windows. The Session Manager's Find dialog is a modeless owned window.

Finally, windows can be modeless and also not owned. Modeless windows that are not owned operate completely independently. The APL2 Window List dialog is such a window. It operates independently of the Session Manager and Object Editor windows.



## Creating Owned Dialogs

Creating an owned dialog is easy. Simply use the handle of the owner as the left argument of `CREATEDLG`:

```
OWNED_DIALOG←OWNER_DIALOG CREATEDLG ''
```

## Executing Modal and Modeless Dialogs

To make an owned dialog modal, pass the owner's handle as the left argument of EXECUTEDLG. For example, here is a function that displays a modal dialog:

```
[0] MODAL OWNER;DLG;TEXT
[1] DLG←OWNER CREATEDLG ''
[2] 'EVENTS' SET_PROPERTY DLG(1 2p'Close' '→0')
[3] SET_PROPERTY DLG 'Modal Dialog'
[4] TEXT←CREATECTL DLG 'TEXT' '' 32776
[5] SET_PROPERTY TEXT 'This dialog is modal'
[6] OWNER(0 EXECUTEDLG)DLG
[7] DESTROYDLG DLG
```

Notice several things:

- The dialog handle OWNER is used to create an owned dialog.
- The function calls EXECUTEDLG. This means that **all** the modal dialog's events will be processed while this function is running.
- The owned dialog is made modal by passing the owner's handle, OWNER, as the left argument of EXECUTEDLG. EXECUTEDLG will disable the owner while executing the owned dialog's events.
- The modal dialog is destroyed before the function returns.

Consider the environment in which this function would be called.

Modal dialogs are usually displayed in response to a user action. For example, the user selects a menu choice and a modal dialog is displayed to prompt for some information. This means that modal dialogs are usually run inside event handlers. This further means that modal dialogs are created, executed, and destroyed all within the execution of an event handler by EXECUTEDLG. And, EXECUTEDLG was called by a function higher on the execution stack.

Applications usually call EXECUTEDLG once to execute the main window. This call to EXECUTEDLG typically does not return until the user closes the main window. When a modal dialog is needed, one of the application's event handlers calls EXECUTEDLG again to process the modal dialog. Because the owner window is disabled and the modal dialog ends before the event handler returns control to the main application's call to EXECUTEDLG, there is no interference between the main dialog's event handlers and the modal dialog's event handlers. Effectively, there is one call to EXECUTEDLG for the main dialog and one call for each modal dialog. This is not true for modeless dialogs. Modeless dialogs are executed by the same instance of EXECUTEDLG that executes the application's main dialog.

Here is a function that creates a modeless dialog:

```
[0] MODELESS OWNER;TEXT
[1] DLG←OWNER CREATEDLG ''
[2] SET_PROPERTY DLG 'Modeless Dialog'
[3] TEXT←CREATECTL DLG 'TEXT' '' 32776
[4] SET_PROPERTY TEXT 'This dialog is modeless'
[5] 'EVENTS' SET_PROPERTY DLG(1 2ρ'Close' 'DESTROYDLG DLG')
[6] 'STATE ACTIVE' SET_PROPERTY DLG 1
```

Notice several things:

- The dialog handle OWNER is again used to create an owned dialog.
- The function does not call EXECUTEDLG. The function simply creates the modeless dialog. The application's main call to EXECUTEDLG will execute the modeless dialog's events.
- The Close event handler is not →0. Rather, the event handler simply destroys the owned dialog. Only the main application window's event handlers should branch to zero and cause the main application's call to EXECUTEDLG to stop waiting for events.
- Setting the STATE ACTIVE property activates the dialog. EXECUTEDLG automatically activates dialogs, but since functions that create modeless dialogs do not call EXECUTEDLG, the functions must explicitly activate them.

The DEMO\_MODE function in the DEMO145 workspace demonstrates message boxes, owned, modal, and modeless dialogs.

## Using Arbitrary Arrays as Event Handlers

Most of the examples in this book have used character vectors as event handlers. Applications that use EXECUTEDLG, EXECUTEDLGX, STARTWAIT, and CHECK\_EVENTS to process events must use character vectors as event handlers. However, you may decide that in your application it is more convenient to use some other type of array as event handlers. The WAIT\_EVENT function supports using arbitrary arrays as event handlers.

The WAIT\_EVENT function waits for a specified number of seconds for an event. If an event does not happen during that time, it returns ' '. If an event does happen, it returns a five element array containing the handle of the window or object that signaled the event, the message identifier, message parameters 1 and 2, and the event handler.

Here is an example of a simple function that uses labels, which are integer scalars, as event handlers:

```
DEMO_ARBITRARY;SV145;DIALOG;OK;EVENT;HANDLE;MSG;MP1;MP2;LABEL

DIALOG←CREATEDLG ''
OK←CREATECTL DIALOG 'Push button' '' 1

SET_PROPERTY DIALOG 'Hello World'
'CLIENT POSITION' SET_PROPERTY OK (5 5)

'EVENTS' SET_PROPERTY DIALOG (1 2ρ'Close' LABEL_CLOSE)
'EVENTS' SET_PROPERTY OK (1 2ρ'Command' LABEL_OK)

'STATE ACTIVE' SET_PROPERTY DIALOG 1

WAIT:
EVENT←WAIT_EVENT 1
→(0=ρEVENT)/WAIT
(HANDLE MSG MP1 MP2 LABEL)←EVENT
→LABEL

LABEL_OK:
SET_PROPERTY DIALOG 'You pushed the Ok button'
→WAIT

LABEL_CLOSE:
DESTROYDLG DIALOG
```

Notice that the function explicitly sets the STATE ACTIVE property. This is because unlike the other event processing functions, WAIT\_EVENT does not activate the dialog.

## Message Boxes

The MSGBOX function displays a simple dialog called a message box. A message box includes a titlebar, a text message, one or more buttons, and optionally an icon. The function returns when the user presses a button or closes the message box.



Use either of the following two types of syntax to call the MSGBOX function:

```
CODE← [OWNER] MSGBOX TEXT  
CODE← [OWNER] MSGBOX TITLE TEXT [STYLE]
```

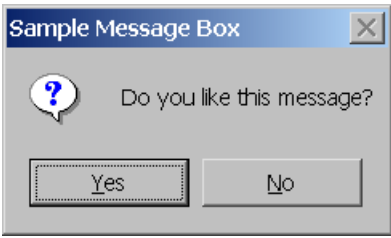
Parameter	Definition
OWNER	Integer scalar - Handle of the message box's owner.
TITLE	Character vector - Text to display in the titlebar. Default: 'APL2 Message'
TEXT	Character vector - Text of the message. Linefeed characters can be used as paragraph breaks.
STYLE	Integer scalar - Specifies the message box's contents and behavior. STYLE can be any BITWISE combination of <a href="#">Message Box Style Flags</a> . Default: MB_OKCANCEL, MB_ICONEXCLAMATION, and MB_MOVEABLE

The MSGBOX function returns a code indicating which button the user pressed to close the message box. See [Message Box Result Codes](#) for the valid codes.

Example:

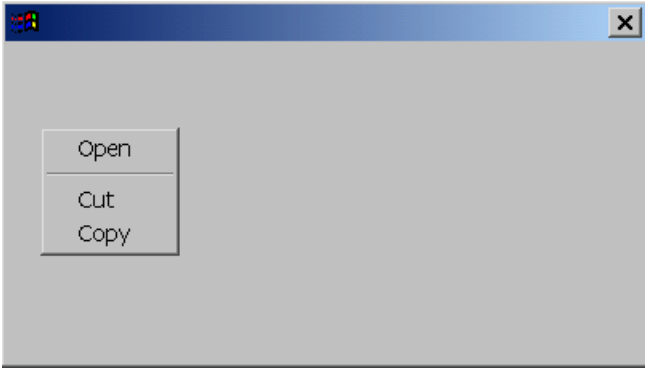
```
TITLE←'Sample Message Box'  
MESSAGE←'Do you like this message?'  
STYLE←vBITWISE/MB_YESNO MB_ICONQUESTION  
MDID_YES=DIALOG MSGBOX TITLE MESSAGE STYLE
```

1



## Popup Menus

The POPUPMENU function displays a floating pop-up menu at a specified position and returns the index of the selected item.



Use the following syntax to call the POPUPMENU function:

```
INDEX←POPUPMENU HANDLE X Y ITEMS
```

Parameters	Definition
HANDLE	Integer scalar - Handle of the window relative to which the menu is positioned.
X	Integer scalar - horizontal position of the menu
Y	Integer scalar - vertical position of the menu
ITEMS	Vector of character vectors - Vector of menu choices. Use a zero length vector to display a separator.

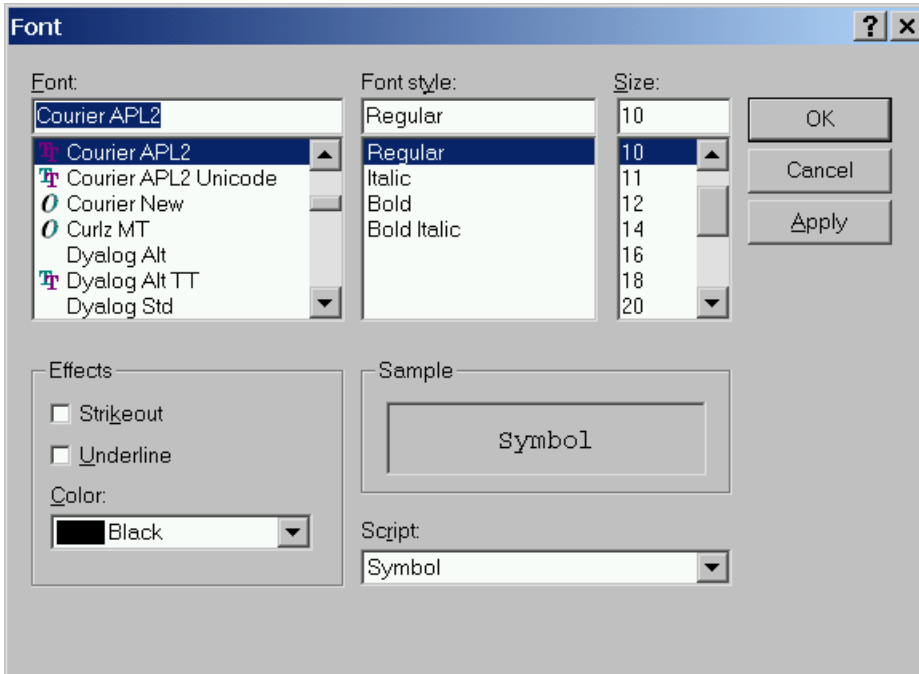
POPUPMENU returns the index of the user's selection. The index is index origin zero. If the user dismisses the menu without making a selection, -1 is returned. Separators do not affect selection indices.

Example:

```
ITEMS←'Open' '' 'Cut' 'Copy'  
INDEX←POPUPMENU DIALOG 25 150 ITEMS  
→(INDEX=̄1) /0  
→(INDEX+␣IO)▷OPEN CUT COPY
```

## Font Dialogs

The FONTDLG function displays a modal dialog that prompts the user to select a font. If the user presses Ok or Apply, the function applies the font to the window specified in the argument.



Use the following syntax to call the FONTDLG function:

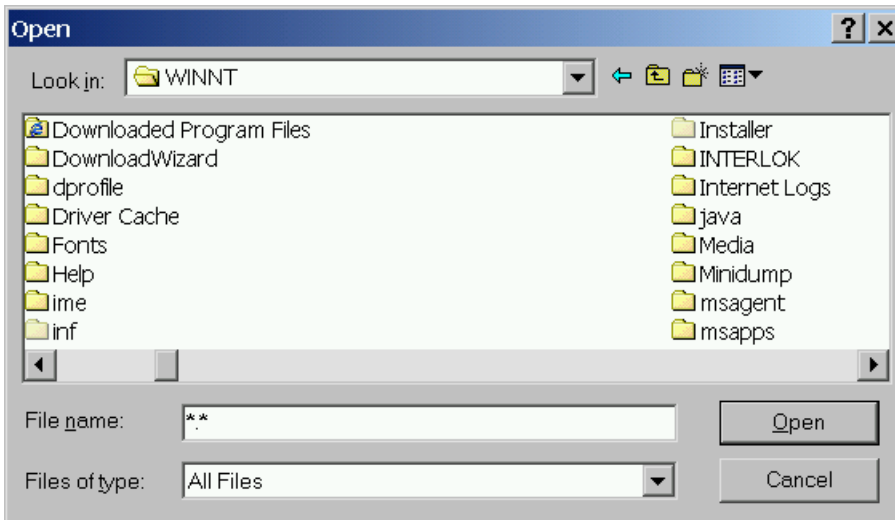
```
FONTDLG HANDLE
```

Parameter	Definition
HANDLE	Integer scalar - Handle of the window whose font should be set. The parent of this window is the font dialog's owner and is disabled while the dialog is visible.



## File Dialogs

The FILEDLG function displays a modal dialog that prompts the user to enter a filename.



Use the following syntax to call the FILEDLG function:

```
FILENAME ← [LIST INDEX [TITLE]] FILEDLG STYLE OWNER FILTER
```

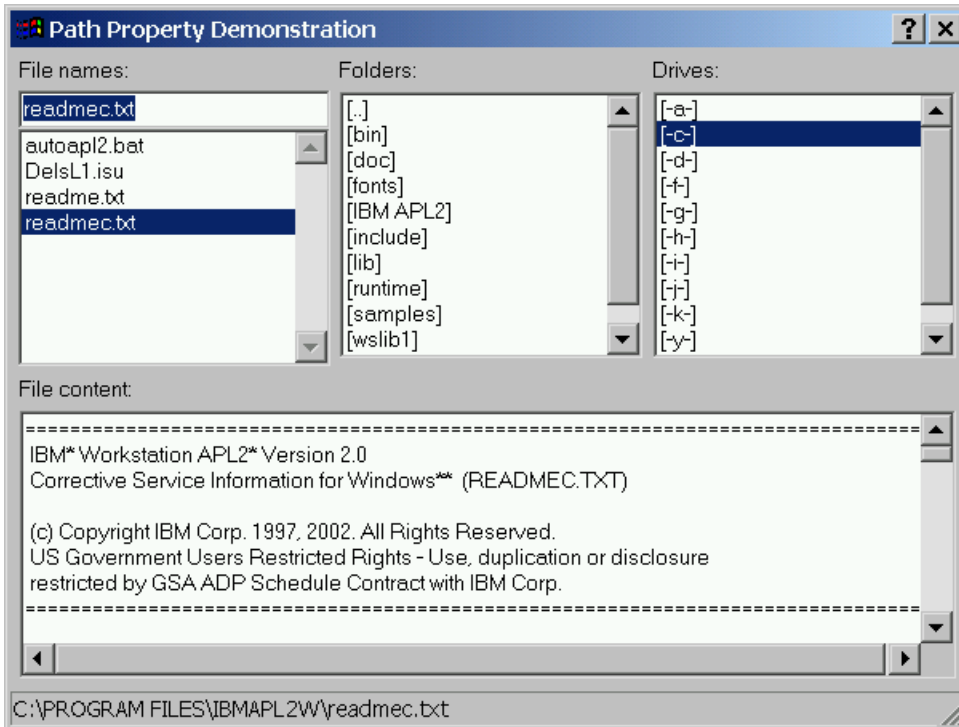
Parameter	Definition
STYLE	Boolean scalar - 0 for Open dialog, 1 for Save As dialog.
OWNER	Integer scalar - Handle of the dialog's owner.
FILTER	Character vector - Filter to initialize the dialog. For example, 'c:\*.*'.
LIST	2 column matrix - Specifies types to be listed in the dialog's <b>Files of type</b> window. Columns are:  [;1] Character vector - File type description. Example: 'All Files'. [;2] Character vector - File filter. Example: '*.*'.  If LIST is supplied, INDEX must also be supplied.
INDEX	Integer scalar - Specifies the index of the LIST element to be initially displayed.
TITLE	Character vector - Dialog box title
FILENAME	Character vector - The selected filename if the user pressed Open or Save. '' otherwise.

The EDIT, EDIT\_OPEN, and EDIT\_SAVEAS functions in the GUITOOLS workspace demonstrate using the FILEDLG function.

## Displaying File Information in Dialogs

Use the PATH, PATH SELECTION, and PATH STYLE properties to display file information in your dialogs.

The DEMO\_PATH function in the DEMO145 workspace demonstrates using the PATH, PATH SELECTION, and PATH STYLE properties to display file information. DEMO\_PATH produced the following dialog showing the readmec.txt file in the APL2 product's ibmapl2w folder:



The dialog uses a combo box to display file names, list boxes to display folders and drives, an MLE to display file contents, and the status area to display a fully qualified path and filename.

Combo boxes and list boxes support the PATH, PATH SELECTION, and PATH STYLE properties.

The PATH STYLE property controls the types of files that the window displays. The PATH STYLE property is a list of path styles. For example, DEMO\_PATH sets the Drives list box's PATH STYLE property to 'DRIVES'. See [Combo box](#) or [List box](#) for the list of supported path styles.

Use the PATH property to further restrict the displayed list of files. The PATH property is a character vector containing wildcard characters and optionally a drive and path. For example, the PATH property could be set to '\WINNT\\*.exe' to display all the programs in the root WINNT directory on the current drive. When the PATH property is set or the user selects a drive or a directory, the AP 145 current drive and directory are changed.

Use the PATH SELECTION property to reference the user's selection. The value is a character vector. The value is dependent on the window's PATH STYLE setting. For example, referencing the PATH SELECTION property of the controls in the DEMO\_PATH example above returned the following values:

<b>Drives</b> list box	'c: '
<b>Folders</b> list box	'ibmapl2w\ '
<b>File names</b> combo box	'readmec.txt '

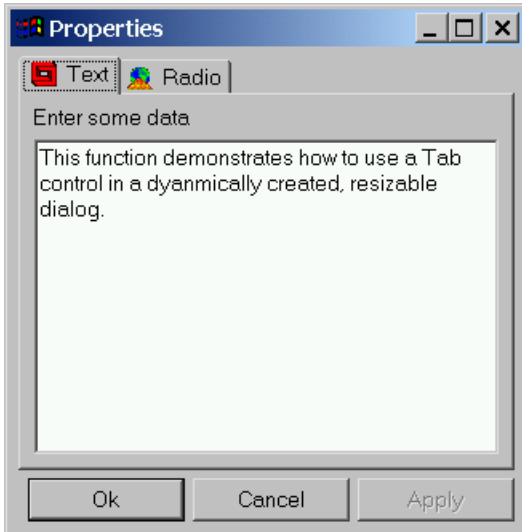
The desktop supports the PATH property. Use the desktop's PATH property to reference and specify the current AP 145 directory. For example, referencing the PATH property of the desktop in the DEMO\_PATH example above returned the following value:

```
'C:\PROGRAM FILES\IBMAPL2W'
```

Combine the value of the desktop's PATH property with the PATH SELECTION property of a combo box or list box to obtain a full path and file name. For example, DEMO\_PATH combines the value of the current directory and the value of the **File names** list box's PATH SELECTION property to produce the value shown in the dialog's status area.

## Tab Controls

The Tab control displays multiple dialogs on the same area of the screen. Each dialog is displayed on a page with a tab or a button.



To use a Tab control, follow these steps:

1. Create a main dialog
2. Create the main dialog's controls including a Tab control
3. Create a separate dialog for each page of the Tab control. For each page's dialog:
  - Use the main dialog's handle as the owner.
  - Use the Tab control's handle as the parent.
  - Create and arrange the page dialogs' controls.
  - Set the page dialog's CLIENT SIZE to fit the controls.
4. Set the Tab control's CLIENT SIZE property to fit the largest page dialog
5. Arrange the main dialog's controls.
6. Set the client size of the main dialog to surround its controls.
7. Set the Tab control's DATA property.

Use the main dialog's handle and the Tab control's handle in the left argument of `CREATEDLG` to create the page dialogs. The first element of the left argument specifies the dialog's owner. The second element specifies the parent. Also, create the page dialogs with the `NoBorder`, `NoSysMenu`, and `NoTitlebar` styles:

```
DIALOG←CREATEDLG ' '
TAB←CREATECTL DIALOG 'TAB' ' '
PAGE1←DIALOG TAB CREATEDLG 'NOBORDER NOSYSMENU NOTITLEBAR'
PAGE2←DIALOG TAB CREATEDLG 'NOBORDER NOSYSMENU NOTITLEBAR'
```

The Tab control's client area contains both the page tabs and dialogs. If the default size of the Tab control is not appropriate, use the following procedure to set the size of the Tab control:

1. Set the CLIENT SIZE property for each page dialog to fit its controls.
2. Set the OFFSET property for the page dialogs' controls to preserve their arrangement.
3. Use the SIZE property to calculate the size of the largest page dialog.
4. Increase the calculated height by the height of at least one row of tabs. To estimate the height of a row of tabs create a push button and query its default height.
5. Set the Tab control's CLIENT SIZE property to the calculated size.

After arranging the page and main dialogs' controls and setting their initial size, set the Tab control's DATA property:

```
DATA←0 3ρ''
DATA←DATA, [1] PAGE1 'Page 1 Tab Text' 'Page1Icon.ico'
DATA←DATA, [1] PAGE2 'Page 2 Tab Text' 'Page2Icon.ico'
'DATA' SET_PROPERTY TAB DATA
```

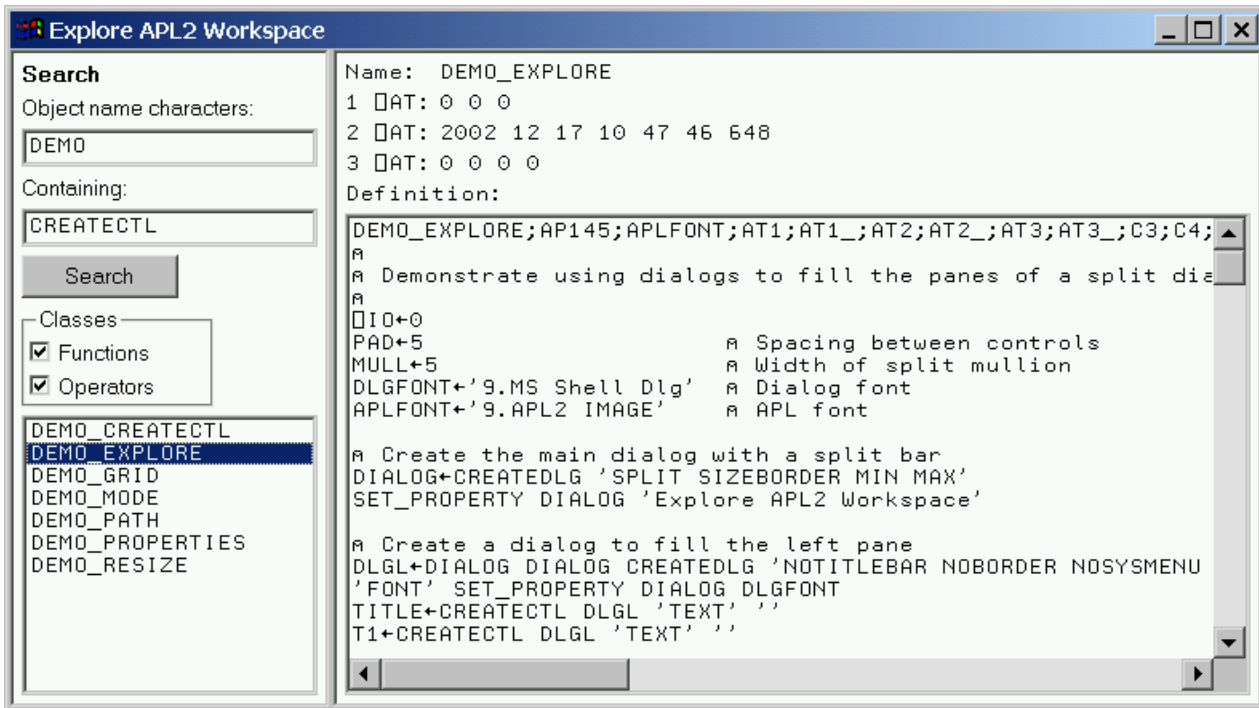
The Tab control's DATA property is a 3-column matrix. There is one row for each page. The columns are:

- [;1] Integer scalar - Handle of dialog to display on the page
- [;2] Character vector - Text to display on the page's tab. '' if none.
- [;3] Character vector - Name of bitmap or icon file to display on the page's tab. '' if none. If a full path is not supplied, files are searched for using the following sequence:
  1. The directory from which AP 145 was loaded
  2. The current directory
  3. The Windows system directory
  4. The directories listed in the PATH environment variable.

The DEMO\_PROPERTIES in the DEMO145 workspace demonstrates dynamically creating a dialog containing a Tab control. The DEMO\_TAB function demonstrates using dialog templates with a Tab control.

## Split Dialogs

The Split and SplitH dialog styles are used to create a dialog with two panes. The Split style separates the panes with a vertical split bar; the SplitH style separates the panes with a horizontal split bar. The first two visible children of the dialog automatically fill the panes. The children can be controls or dialogs. When the user moves the split bar, the dialog automatically resizes the two children to fill the panes.



When creating controls to fill the panes, simply use the split dialog's handle as the parent parameter of CREATECTL. The controls are resized to fill the panes.

```
DIALOG←CREATEDLG 'SIZEBORDER SPLIT'  
LB←CREATECTL DIALOG 'LISTBOX' ''  
MLE←CREATECTL DIALOG 'MLE' ''
```

When creating dialogs to fill the panes, use the split dialog's handle as owner and parent in the left argument of CREATEDLG. The first element of the left argument specifies the dialog's owner. The second element specifies the parent. Also, create the panes' dialogs with the NoBorder, NoSysMenu, and NoTitlebar styles:

```
DIALOG←CREATEDLG 'SIZEBORDER SPLIT'  
LEFT←DIALOG DIALOG CREATEDLG 'NOBORDER NOSYSMENU NOTITLEBAR '  
RIGHT←DIALOG DIALOG CREATEDLG 'NOBORDER NOSYSMENU NOTITLEBAR '
```

When a split dialog's split bar is moved, or when the dialog is resized, the dialog automatically resizes the children to fill the panes. Set the OFFSET property for the panes' dialogs' controls so all the controls maintain their relative positions.

For an example of using dialogs in the panes of a split dialog, see the DEMO\_EXPLORE function in the DEMO145 workspace.

## Displaying Graphics in a Dialog

An AP 207 graphics window can be used in an AP 145 dialog.

```
SV207←'OPEN' (HANDLE [ID [WIDTH HEIGHT [X Y]]])
RC←SV207
```

Parameter	Definition
HANDLE	Integer scalar - A window handle
ID	Integer scalar - A control window identifier. If ID is omitted or is zero, the graphics window is opened within the client area window identified by HANDLE. If ID is non-zero, the graphics window is opened within the client area of the child window whose parent is HANDLE and whose identifier is ID.
WIDTH HEIGHT	Integer scalars - The width and height of the graphic window in pixels. The default size fills the client area.
X Y	Integer scalar - The position of the graphic window within the client area relative to the origin at the lower left hand corner. The default position is the origin.

When a graphic window is opened in a dialog the AP 207 WAIT and POINT commands do not return Tab and Shift+Tab keystrokes. Dialog Tab key processing is performed instead.

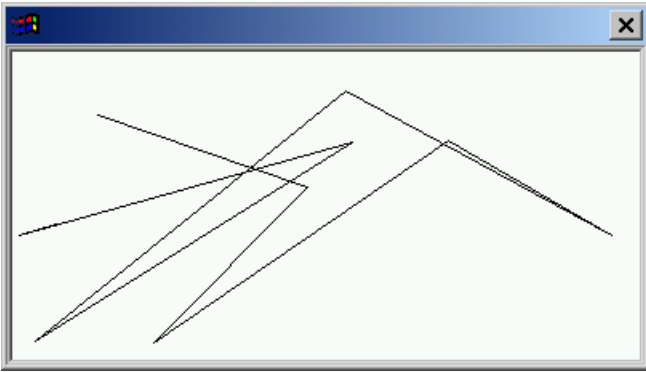
The graphic window is opened as a child of the specified window. The graphic window has identifier 32776.

Graphic windows support the CONTEXT HELP property. If a graphic window is opened within a control window, the control window's contextual help text will be displayed if no contextual help has been supplied for the graphic window.

The AP 207 CLOSE command must be issued before destroying a dialog containing a graphic window.

The following example creates a dialog with a child rectangle control that uses the identifier of 32776 so that it fills the dialog's client area. An AP 207 graphic window is then opened within the rectangle.

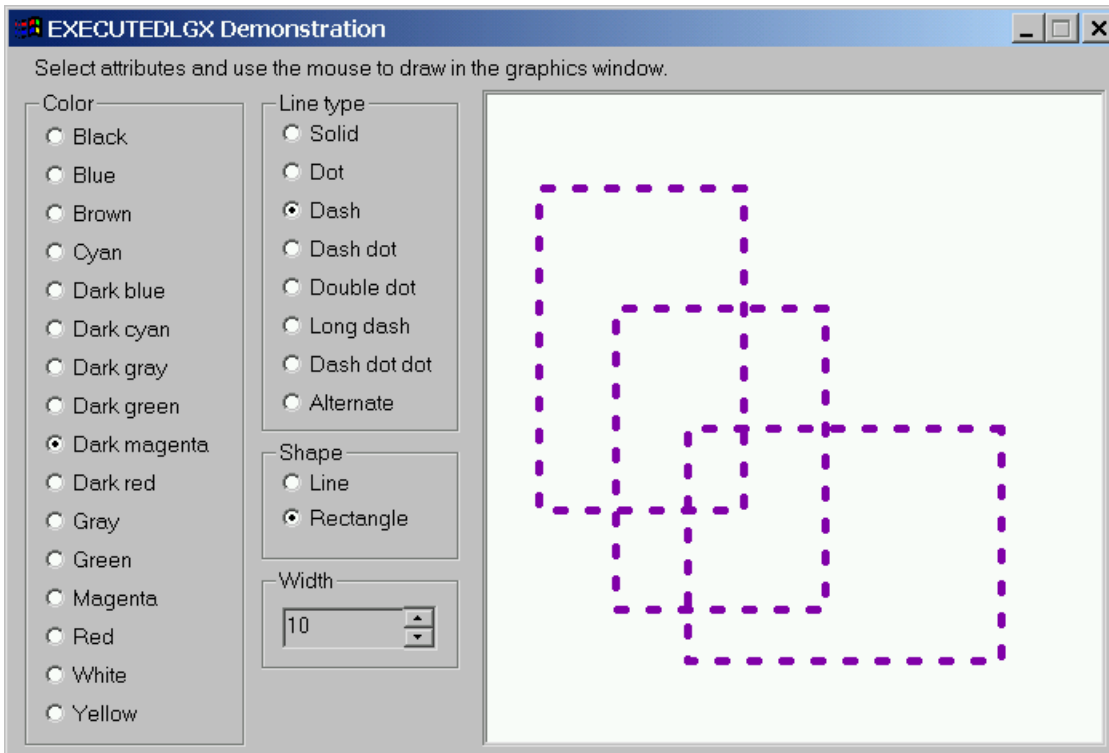
```
DIALOG←CREATEDLG ''
RECT←CREATECTL DIALOG 'RECTANGLE' 'BORDER' 32776
DES(2≠207 SVOFFER 'SV207')/'AP 207 share failed'
SV207←'OPEN' RECT
RC←SV207
SV207←'DRAW' (?10 2ρ 'CLIENT SIZE' GET_PROPERTY RECT)
RC←SV207
```



For an example of using AP 207 in a dialog, see the DEMO\_AP207 function in the DEMO145 workspace.

### Using AP 145 with other Auxiliary Processors

EXECUTEDLGX is used when an application needs to wait for events from both AP 145 and other processors. For example, use EXECUTEDLGX to handle user input through both AP 145 and AP 207, or to wait for user input through AP 145 and TCP/IP events through AP 119.



The EXECUTEDLGX operator has the following syntax:

```
[RESULT←] [OWNER] (SVPROC EXECUTEDLGX SVARS) HANDLE
```

<b>HANDLE</b>	A dialog handle.
<b>OWNER</b>	The handle of the dialog's owner. If OWNER is supplied, the dialog is modal.

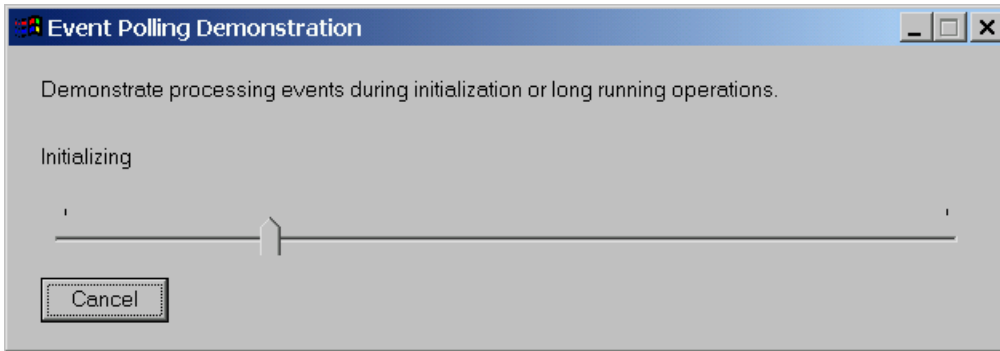


SVARS	<p>A list of one or more shared variable names to monitor.</p> <p>SVARS can be a character vector containing one shared variable name or a matrix containing one shared variable name per row.</p>
SVPROC	<p>An application supplied function for processing shared variable events.</p> <p>When a partner specifies any of the variables named in SVVARS, SVPROC is called. The list of variables currently being monitored is passed to SVPROC as a right argument. SVPROC should use <code>□SVS</code> to determine which variables have had events and process them. SVPROC must return a list of variables for EXECUTEDLGX to continue to monitor.</p>
RESULT	<p>An explicit result. RESULT is not set by EXECUTEDLGX, but can be set by event handlers.</p>

For an example of processing AP 145 and 207 events, see the DEMO\_EXECUTEDLGX function in the DEMO145 workspace.

## Remaining Responsive During Long Operations

GUI applications should process events in less than a tenth of a second. Otherwise, the user may think the application has stopped responding and has failed. Some applications include operations that take longer than a tenth of a second to complete. In order to remain responsive to user actions, applications that include long running operations should not use EXECUTEDLG. There are two techniques available to these applications: using START\_WAIT and CHECK\_EVENTS or using WAIT\_EVENT.



### Using START\_WAIT and CHECK\_EVENTS

The EXECUTEDLG operator uses the AP 145 AplWaitMsg service. This service tells AP 145 to start waiting for an event to occur. When an event occurs, the service completes.

EXECUTEDLG works by specifying the shared variable named SV145 with a call to AplWaitMsg and immediately referencing the variable. Because AplWaitMsg does not complete until an event has occurred, if no event has occurred, the reference hangs until one occurs. When an event does occur, AP 145 specifies the variable, the reference completes, and EXECUTEDLG executes the event handler.

During EXECUTEDLG's shared variable reference hang, your application is unable to perform any other work.

START\_WAIT, CHECK\_EVENTS, and WAIT\_EVENT can be used to process events without hanging. START\_WAIT specifies SV145 with a call to AplWaitMsg and immediately returns; it does not wait for the service to complete.

CHECK\_EVENTS checks whether SV145 has been specified, indicating an event has occurred. If SV145 has been specified, CHECK\_EVENTS references the variable, executes the event handler, and returns. If SV145 has not been specified, indicating no event has occurred, CHECK\_EVENTS simply returns.

Since CHECK\_EVENTS does not hang, it can be used to poll for events during a long running calculation.

The following example illustrates how to use START\_WAIT and CHECK\_EVENTS to perform a long running calculation while remaining responsive to user actions:

```

START_WAIT
LOOP:
  * Process any events that may have occurred.
  * CHECK_EVENTS returns 1 if an →0 event occurs, 0 otherwise.
  →CHECK_EVENTS/END
  * Perform part of the long running calculation here
  →LOOP
END:

```

When `START_WAIT` is called, AP 145 starts waiting for events. As they occur, they are saved for retrieval one by one using `CHECK_EVENTS`. If the `SV145` shared variable is used for any other purpose, any saved events are discarded. To avoid this loss of events caused by user interactions, the long running calculation must not use `SV145`. If the calculation must use AP 145 facilities, such as setting and getting properties, it should share another variable and use the AP 145 services directly rather than using the `GUITOOLS` cover functions.

For examples of using `START_WAIT` and `CHECK_EVENTS`, see the `DEMO_POLLING` and `DEMO_GDI` functions in the `DEMO145` workspace.

### Using `WAIT_EVENT`

The `WAIT_EVENT` function's right argument is the number of seconds to wait for an event. Use zero to query whether an event has been queued. If no event has been saved, `WAIT_EVENT` returns `' '`. Otherwise, it returns a 5 element array containing the following items:

1. The handle of the window or object that signaled the event
2. The message identifier
3. Message parameter 1
4. Message parameter 2
5. The event handler

Unlike `CHECK_EVENTS`, `WAIT_EVENT` does not execute the event handler. Your application must process the event handler itself. `WAIT_EVENT` always references `SV145`. So, you can freely use `SV145` and other `GUITOOLS` functions with `WAIT_EVENT`.

The following example illustrates how to use `WAIT_EVENT` to perform a long running calculation while remaining responsive to user actions:

```

DIALOG←CREATEDLG ''
'EVENTS' SET_PROPERTY DIALOG(1 2ρ'CLOSE' CLOSE)
'STATE ACTIVE' SET_PROPERTY DIALOG 1
LOOP:
  * Perform part of the long running calculation here
  * Check if there is an event to process
  EVENT←WAIT_EVENT 0
  →(0=ρEVENT)/LOOP
  * Process events
  (HANDLE MSG MP1 MP2 HANDLER)←EVENT
  →HANDLER
CLOSE:
  DESTROYDLG DIALOG

```

## Using AP 145 Services

The functions in the GUITOOLS workspace use Auxiliary Processor 145 to create and manage windows. The CREATEDLG function shares a variable named SV145 with AP 145. The rest of the workspace's functions use this shared variable. Cover functions are provided for many, but not all, of AP 145's services. To use the facilities discussed in the next few sections, you must use AP 145 directly. This section explains the process of calling AP 145 services.

To call an AP 145 service, assign a nested vector to the shared variable. The first element of the vector is a character vector containing the service name. The rest of the elements of the array are the service's parameters.

For example, the following statements call the AplSizeToText service and reference the return codes:

```
SV145←'AplSizeToText' DIALOG ID  
(APRC OSRC CMD)←SV145
```

Where DIALOG is the handle of a dialog and ID is a control's identifier.

Each result returned by AP 145 is a three element nested array. The first element is a scalar integer that is an auxiliary processor return code; it indicates whether the service was called successfully. The second element is a scalar integer and is the service's return code or result. The third element is the array specified to the shared variable; some elements of the array may have been updated by the service.

Here is an example that calls a service that returns a value:

```
SV145←'WinWindowFromID' DIALOG ID  
(APRC OSRC CMD)←SV145
```

This service is used by the WINDOWFROMID function. OSRC is the child window's handle.

AP 145 uses a general purpose parameter conversion service which is used both for built-in services and services in other libraries.

Each service parameter must be a numeric scalar, a one item numeric vector, a character scalar, or a character vector. Numeric parameters for built-in services must be either Boolean or integer.

AP 145 passes numeric items to services as four byte unsigned integers. Character vectors are padded with the string termination character, □AF 0. The address of the terminated vector is passed to the service. AP 145 automatically supplies indirection required by services that update their parameters.

Here is an example that calls a service that updates a parameter:

```
SV145←'AplGetProperty' HANDLE PROPERTY_NAME 0 ID  
(APRC OSRC CMD)←SV145  
(API HANDLE PROPERTY VALUE ID)←CMD
```

Notice the service's third parameter is coded as a zero. If successful, the service replaces this parameter with the property's value.

The CALLAPI function simplifies calling services. Consider the following group of statements:

```
SV145←'AplSizeToText' DIALOG ID  
(APRC OSRC CMD)←SV145  
□ES (APRC≠0) / 'AP 145 FAILED'
```

They can be replaced with the following single statement:

```
(OSRC CMD)←CALLAPI 'AplSizeToText' DIALOG ID
```

CALLAPI checks the AP 145 return code and if it is zero, returns the second and third elements of the result returned by AP 145.

AP 145 issues the following return codes:

0	Success
1	Invalid array
2	Unsupported service name
3	Incorrect number of parameters
4	Invalid parameter
5	System error
-1	Insufficient space

For information about the services supported by AP 145, consult [Appendix A: AP 145 Services](#).

## Using System Services

AP 145 maintains a table of named services for each shared variable. Each table is initially empty. When a service name is called, AP 145 searches the variable's named services table before searching for a built-in service.

Use `LOADAPI` to add services to a variable's table of named services. Use a library name and a service name to add a service from a dynamic link library. Use an address and a service name to add a service located using non-APL facilities. Supply a description of the service's parameters to indicate AP 145 should perform parameter type checking and conversion. Supply a description of the service's result to indicate AP 145 should perform result type conversion.

Use `CALLAPI` to call services. Use a service name to call a service in the variable's table of named services or use an address to directly call a service located using non-APL facilities using default parameter conversion.

Use `FREEAPI` to remove services from the variable's table of named services. All services are automatically removed when the shared variable is retracted.

AP 145 supports 32 bit services that use the Microsoft Visual Studio compilers' `__stdcall` calling convention. No checking of linkage convention is performed. Attempts to call services that use other linkage conventions can yield unpredictable results.

```
LOADAPI [LIBRARY | ADDRESS] SERVICE [DESCRIPTION [RESULT]]
(OSRC CMD)←CALLAPI [SERVICE | ADDRESS] parm1 parm2 ...
FREEAPI SERVICE
```

`LIBRARY` - Character vector - Filename of the DLL containing the routines

If a path is supplied but the file does not exist in the specified directory, the service fails. If a path is not supplied and the filename extension is omitted, the default library extension `.DLL` is appended. However, the library vector can include a trailing period (`.`) to indicate that the filename has no extension. When no path is specified, the following sequence is used:

- 1 The directory from which AP 145 was loaded.
- 2 The current directory.
- 3 The 32-bit Windows system directory. Use the `GetSystemDirectory` routine to get the path of this directory. The name of this directory is `SYSTEM32`.
- 4 The 16-bit Windows system directory. There is no `Win32` function that obtains the path of this directory, but it is searched. The name of this directory is `SYSTEM`.
- 5 The Windows directory. Use the `GetWindowsDirectory` routine to get the path of this directory.
- 6 The directories that are listed in the `PATH` environment variable.

`ADDRESS` - Integer scalar - Address of a routine located using non-APL facilities

SERVICE - Character vector or vector of character vectors - Case sensitive names of one or more routines

For LOADAPI:

If LIBRARY is supplied, SERVICE may be either a character vector containing a routine name or vector of character vectors each containing a routine name.

If ADDRESS is supplied, SERVICE must be a character vector containing a routine name.

If SERVICE is a character vector, then a DESCRIPTION argument may be supplied.

If the DESCRIPTION argument is omitted or if SERVICE is a vector of character vectors, then AP 145 will pass all the routines' parameters by value and as 4 byte signed integers or single byte character strings.

For CALLAPI:

SERVICE must be a character vector containing a routine name.

For FREEAPI:

SERVICE may be either a character vector or a vector of character vectors.

DESCRIPTION - Integer vector - Description of an routine's parameters - Optional

The length of the description corresponds to the number of parameters required by the routine. When the routine is called, AP 145 will validate that this number of parameters are passed. Each item corresponds to a parameter and describes that parameter. Descriptions may not contain more than 32 items. Pass a null vector to indicate that the service has no parameters.

Each item of the description specifies the type of the parameter and the amount of indirection that is used to pass the parameter to the routine.

Use values of 0 and 1 for automatic type handling. Numbers are passed as 4 byte signed integers. Character scalars and vectors are passed as null terminated single byte character strings. Data that cannot be converted to one of these representations is not supported. Code a 0 for parameters which will not be updated by the service. Code a 1 for parameters which will be updated; the address of the parameter will be passed.

Use other description values to specify the type of the parameter. Code a positive value for parameters which will not be updated by the service. Code a negative value for parameters which will be updated; the address of the parameter will be passed. The following absolute values are used to indicate parameter types:

- 0 - Automatic type handling. Parameter is passed by value
- 1 - Automatic type handling. Parameter is passed by reference
- 2 - Parameter is 1 byte signed integer
- 3 - Parameter is 2 byte signed integer
- 4 - Parameter is 4 byte signed integer
- 5 - Parameter is 4 byte floating point number
- 6 - Parameter is 8 byte floating point number
- 7 - Parameter is string of single byte characters, AP 145 null terminates the string.
- 8 - Parameter is string of 2 byte characters, AP 145 null terminates the string.

RESULT - Integer scalar – Description of single API's result - Optional

The value of `result` is interpreted as follows:

- 2 - Result is 1 byte signed integer
- 3 - Result is 2 byte signed integer
- 4 - Result is 4 byte signed integer (the default)
- 5 - Result is 4 byte floating point number
- 6 - Result is 8 byte floating point number
- 7 - Result is string of single byte characters
- 8 - Result is string of 2 byte Unicode characters

The following pages demonstrate using several types of services. For more demonstrations using routine addresses, consult the `DEMO_API` function in the DEMO145 workspace.



### Example using a service that updates an integer parameter:

Assume the following service is available in MYLIB.DLL, which resides in the current directory.

```
ULONG _System Sample(char * P1,unsigned long P2,unsigned long * P3);
```

The service expects the following parameters:

1. A null terminated character string
2. An integer
3. A pointer to an integer

The following code shows how to load and use the Sample service. Notice the description parameter passed to LOADAPI has 3 elements corresponding to the Sample service's 3 parameters. The last element is 1 indicating a pointer to this item should be passed.

```
LOADAPI 'MYLIB' 'Sample' (0 0 1)
(OSRC CMD)←CALLAPI 'Sample' 'String' 47 0
CMD
Sample String 47 123456
FREEAPI 'Sample'
```

Notice that the Sample service has updated the last parameter.

## Example using a Microsoft Windows service:

The following example demonstrates using the CharUpperBuff service that is included in Microsoft Windows. The Microsoft Platform SDK documentation defines the service like this:

### Syntax

```
DWORD CharUpperBuff(  
    LPTSTR lpsz,  
    DWORD cchLength  
);
```

### Parameters

lpsz	[in] Pointer to a buffer containing one or more characters to process.
cchLength	[in] Specifies the size, in TCHARs, of the buffer pointed to by lpsz. This refers to bytes for ANSI versions of the function or WCHARs for Unicode versions. The function examines each character, and converts lowercase characters to uppercase characters. The function examines the number of characters indicated by cchLength, even if one or more characters are null characters.

### Return Value

The return value is the number of TCHARs processed.

### Function Information

Header	Declared in Winuser.h, include Windows.h
Import library	User32.lib
Minimum operating systems	Windows 95, Windows NT 3.1
Unicode	Implemented as Unicode and ANSI versions on Windows NT, Windows 2000, Windows XP

Notice that the requirements state that the API is implemented as both Unicode and ANSI versions. This means that the Windows DLL actually contains two different APIs with different names. These versions are distinguished by the addition of the letter A or W at the end of the documented name. A is used for ANSI APIs; W is used for Unicode APIs. In this case, the actual names of the APIs in the Windows DLL are CharUpperBuffA and CharUpperBuffW.

Notice the SDK documentation also lists User32.lib as the Library. This file is used when linking compiled programs that use the API. It tells the linker how to find the API. It points to entries in the corresponding DLL. So, the actual DLL is named User32.Dll.

The following example loads the ANSI version of the API.

```

LOADAPI 'user32' 'CharUpperBuffA'
VECTOR←'This is a test.'
(OSRC CMD)←CALLAPI 'CharUpperBuffA' VECTOR (ρVECTOR)
OSRC
15
(API UPPER LENGTH)←CMD
UPPER
THIS IS A TEST.
FREEAPI 'CharUpperBuffA'

```

Notice that the example does not pass a service description to LOADAPI. No description is necessary because the service requires only integer and single byte character string arguments that are not updated.

## Example using a service with a parameter that is a structure:

The following example demonstrates using the GetWindowRect service that is included in Microsoft Windows. The Microsoft Platform SDK documentation defines the service like this:

### Syntax

```
BOOL GetWindowRect(  
    HWND hWnd,  
    LPRECT lpRect  
);
```

### Parameters

hWnd	[in] Handle to the window.
lpRect	[out] Pointer to a structure that receives the screen coordinates of the upper-left and lower-right corners of the window.

### Return Value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero. To get extended error information, call GetLastError.

### Function Information

Header	Declared in Winuser.h, include Windows.h
Import library	User32.lib
Minimum operating systems	Windows 95, Windows NT 3.1

Notice that the second parameter is a pointer to a RECT structure. The Microsoft Platform SDK defines the RECT structure like this:

```
typedef struct _RECT {  
    LONG left;  
    LONG top;  
    LONG right;  
    LONG bottom;  
} RECT, *PRECT;
```

### Members

left	Specifies the x-coordinate of the upper-left corner of the rectangle.
top	Specifies the y-coordinate of the upper-left corner of the rectangle.
right	Specifies the x-coordinate of the lower-right corner of the rectangle.
bottom	Specifies the y-coordinate of the lower-right corner of the rectangle.

Structures are used in the C language like nested and non-homogeneous arrays in APL2. You can define patterns to be used with the external functions ATR, RTA, and SIZEOF to convert APL2 arrays to character vectors that can be passed to C.

The following example uses the GetWindowRect API to get a window's rectangle:

```

3 11 □NA>'ATR' 'RTA' 'SIZEOF'
1 1 1
LONG←'(I4 0) '
RECT←'(G0 1 4) ',LONG,LONG,LONG,LONG

LOADAPI 'user32' 'GetWindowRect' (0 0)
(OSRC CMD)←CALLAPI 'GetWindowRect' HANDLE ((SIZEOF RECT)ρ□AF 0)
OSRC
1
(API HANDLE DATA)←CMD
RECT RTA DATA
200 624 606 854

FREEAPI 'GetWindowRect'

```

This example shows using an API that only requires a pointer to an empty structure. Therefore, SIZEOF is used to create the correct size vector of bytes containing zero. ATR can also be used to convert an APL2 array to a structure:

```
STRUCTURE←RECT ATR 0 0 0 0
```

Notice that although the example passes a service description to LOADAPI, no description is necessary because the service requires only integer and address arguments that are not updated.

## Example using a service that has no parameters:

The following example demonstrates using the GetLastError service to get the last error code set by Windows. The Microsoft Platform SDK documentation defines the service like this:

```
DWORD GetLastError(void);
```

### Parameters

This function has no parameters.

### Return Values

The return value is the calling thread's last-error code value.

### Requirements

Client	Included in Windows XP, Windows 2000 Professional, Windows NT Workstation, Windows Me, Windows 98, Windows 95.
Server	Included in Windows .NET Server 2003, Windows 2000 Server, Windows NT Server.
Header	Declared in Winbase.h; include Windows.h.
Library	Use Kernel32.lib.

The following example uses the GetLastError API to get the last error code:

```
LOADAPI 'kernel32' 'GetLastError' (10)
(OSRC CMD) ←CALLAPI 'GetLastError'
OSRC
1
FREEAPI 'GetLastError'
```

Notice the LOADAPI service description is zero length to indicate the GetLastError API has no parameters. This causes AP 145 to verify the number of parameters passed to GetLastError. If the service description was omitted, GetLastError could still be called, but AP 145 would not perform this verification.

## Example using a service that has floating point parameters:

The following example demonstrates using the OpenGL `glRectd` service to draw a rectangle. The Microsoft Platform SDK documentation defines the service like this:

```
Void glRectd(  
    GLdouble x1,  
    GLdouble y1,  
    GLdouble x2,  
    GLdouble y2  
);
```

### Parameters

`x1, y1`  
One vertex of a rectangle.

`x2, y2`  
The opposite vertex of the rectangle.

### Requirements

**Windows NT/2000:** Requires Windows NT 3.5 or later.

**Windows 95/98:** Requires Windows 95 or later. Available as a redistributable for Windows 95.

**Header:** Declared in `Gl.h`

**Library:** Use `Opengl32.lib`

The following example uses the OpenGL `glRectD` service to draw a rectangle:

```
LOADAPI 'OPENGL32' 'glRectd' (6 6 6 6)  
(OSRC CMD) ←CALLAPI 'glRectd' 12.34 56.78 101 34  
FREEAPI 'glRectd'
```

Notice the `LOADAPI` service description specifies that `glRectD` requires four 8 byte floating point parameters.

Note: `glRectd` requires a valid OpenGL rendering context.

### Example using a Unicode service:

The following example demonstrates passing a string of 2 byte Unicode characters to the CharUpperBuffW service:

```
LOADAPI 'user32' 'CharUpperBuffW' (8 3)
VECTOR←'This is a test.'
(OSRC CMD)←CALLAPI 'CharUpperBuffW' VECTOR (ρVECTOR)
OSRC
15
(API UPPER LENGTH)←CMD
UPPER
THIS IS A TEST.
FREEAPI 'CharUpperBuffW'
```

Notice the LOADAPI service description specifies that CharUpperBuffW requires two parameters: a null terminated string of 2 byte Unicode characters and an integer.

AP 145 automatically adds a 2 byte null character to the end of character vectors passed as 2 byte character string parameters.



## Examples using routines located using non-APL facilities:

The following examples demonstrate using non-APL facilities to locate services.

OpenGL extension functions are not exported by name from the OpenGL libraries and so LOADAPI can not be used to load them by name. Instead, to use OpenGL extension functions, use the wglGetProcAddress service which returns the address of an OpenGL extension function.

The following example demonstrates loading and using wglGetProcAddress to get the address of the glDrawBuffer extension function. LOADAPI is then used to add the address and a description of the extension function's parameters to the shared variable's table of services. The (,4) description indicates the function has a single integer parameter. Next, CALLAPI is used to call the extension function. Finally, FREEAPI is used to remove the two services from variable's table of services.

```
LOADAPI 'OPENGL32' 'wglGetProcAddress'  
ADDRESS←1>CALLAPI 'wglGetProcAddress' 'glDrawBuffer'  
LOADAPI ADDRESS 'glDrawBuffer' (,4)  
0 0ρCALLAPI 'glDrawBuffer' GL_NONE  
FREEAPI 'glDrawBuffer' 'wglGetProcAddress'
```

If AP 145's default parameter handling suffices for a routine, a description is not required. Furthermore, the routine does not need to be added to the variable's table of services. The service can be called directly using only the address. For example:

```
LOADAPI 'OPENGL32' 'wglGetProcAddress'  
ADDRESS←1>CALLAPI 'wglGetProcAddress' 'glDrawBuffer'  
0 0ρCALLAPI ADDRESS GL_NONE  
FREEAPI 'wglGetProcAddress'
```

The following example demonstrates how to locate and call the wglGetExtensionsStringARB OpenGL extension function.

```
⌘ Get pointer to list of WGL extensions  
LOADAPI 'OPENGL32' 'wglGetProcAddress'  
ADDRESS←1>CALLAPI 'wglGetProcAddress' 'wglGetExtensionsStringARB'  
LISTADDR←1>CALLAPI ADDRESS DC  
FREEAPI 'wglGetProcAddress'
```

wglGetExtensionsStringARB returns the address of a list of the OpenGL extension functions supported by the current OpenGL rendering context (a handle to which is in the variable DC in the previous example.) The list is a null terminated character string. The following example demonstrates how to copy the list into the workspace:

```
⌘ Extract list from memory  
LOADAPI 'KERNEL32' ('RtlMoveMemory' 'lstrlenA')  
LENGTH←1>CALLAPI 'lstrlenA' LISTADDR  
LIST←2 2>CALLAPI 'RtlMoveMemory' (LENGTHρ□AF 0) LISTADDR LENGTH  
FREEAPI 'RtlMoveMemory' 'lstrlenA'
```

## ***Objects: Timers and DDE***

Dialogs and control windows manage communication between your application and users. AP 145 objects manage communication between your application and system facilities. Objects support use of the system clock to signal events at specified intervals and communication with other applications using Dynamic Data Exchange (DDE).

Use the CREATEOBJ function to create objects. For detailed information about creating objects, see [CREATEOBJ](#) and [Object Reference](#).

### **Note:**

The DDE technology uses 16 bit integers to store data lengths. Therefore, data passed through DDE is limited in size. Use the COM external function or the GUITOOLS function CALLCOM to pass larger arrays.

## Timers

Timers are the simplest objects. They are used to signal events at specified intervals. For example,

```
TIMER←CREATEOBJ 'TIMER' .5 '□TS'  
DEFAULTPROC EXECUTEDLG TIMER  
2003 12 6 15 40 52 590  
2003 12 6 15 40 53 60  
2003 12 6 15 40 53 560
```

The first argument of CREATEOBJ is a character vector that contains the class of object to create. Subsequent arguments are class-specific initialization data. Timer objects require two initialization data items: an interval in seconds, and an event handler expression. When the interval has elapsed, the AplWaitMsg service returns the event handler expression.

CREATEOBJ returns a handle that uniquely identifies the object. Like window handles, object handles are integer scalars and are used to refer to objects in calls to other functions. For example, the SET\_PROPERTY function can be used to set timer object properties:

```
'INTERVAL' SET_PROPERTY TIMER .75  
'EVENTS' SET_PROPERTY TIMER (1 2ρ'Timer' '3↓□TS')
```

For more information about creating timers, consult [Timer](#). The DEMO\_TIMER function in the DEMO145 workspace demonstrates using a timer.

## Dynamic Data Exchange

Use DDE objects to manage communication between APL2 and other DDE-enabled applications. You can use DDE objects to communicate with spread-sheets, databases, and even Windows itself. This chapter provides a very brief introduction to using APL2 DDE objects. The DDESHARE workspace in library 2 contains a variety of utility and demonstration functions. For more detailed information consult the chapter on the DDESHARE workspace in the User's Guide.

There are two types of DDE applications: servers and clients. DDE server applications manage repositories of information. Client applications can make requests to reference and update items in the repositories. Client applications can also request that servers execute commands.

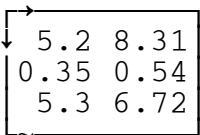
For example, Microsoft Excel can operate as a DDE server and APL2 can access spreadsheet data as a DDE client. The following statements demonstrate how a DDE connection can be established between APL2 and Excel:

```
APPNAME←'Excel '  
TOPIC←' [Book1] Sheet1 '  
ITEM←'R1C1:R3C2 '  
HDATA←CREATEOBJ 'DDE DATA' APPNAME TOPIC ITEM
```

DDE DATA objects manage the communications between APL2 and DDE servers. Three initialization data items are required: the name of the DDE server application, the name of a topic, and the name of an item. In the case of Excel, the topic is a spreadsheet name and the item name is a range of cells. You can either refer to a range of cells explicitly as shown above or use cell range names defined in Excel.

Once a DDE DATA object is created, APL2 can make requests of the Excel DDE server. For example, use GET\_PROPERTY to reference the value of the item:

```
VALUE←'XLTABLE DATA' GET_PROPERTY HDATA  
DISPLAY VALUE
```



5.2	8.31
0.35	0.54
5.3	6.72

Use SET\_PROPERTY to specify new values:

```
'XLTABLE DATA' SET_PROPERTY HDATA (3 2) 6)
```

Use DDE COMMAND objects to request that DDE server applications execute commands. For example:

```
HCMD←CREATEOBJ 'DDE COMMAND' 'Excel' 'Sheet1 '  
SET_PROPERTY HCMD '[RUN("MACRO")] '
```

DDE COMMAND objects require two initialization items: the name of a DDE server application and a topic name. Commands apply to the topic with which the DDE COMMAND object is communicating.

Further examples of using DDE DATA and DDE COMMAND objects can be found in [Appendix C: Using APL2, DDE and Microsoft Excel](#) and [Appendix D: Using APL2, DDE and Microsoft Access](#).

## DDE Servers

Most APL2 applications that use DDE are clients and only use DDE DATA and DDE COMMAND objects. However, it is also possible to build DDE server applications in APL2.

DDE server applications manage repositories of information that are arranged in a hierarchical structure:

- DDE SERVER object
- DDE TOPIC objects
- DDE ITEM objects

DDE server, topic, and item objects all have names.

- Each DDE SERVER object has an application name.
- Each DDE TOPIC object has a topic name.
- Each DDE ITEM object has an item name.

Each APL2 DDE server application has a DDE SERVER object. AP 145 uses the server object's name to identify the server in connection requests from other applications.

Each DDE SERVER object contains one or more DDE TOPIC objects. Topic names identify sections of the repository. For example, Excel uses spreadsheet names as topic names.

Each DDE TOPIC object contains one or more DDE ITEM objects. Each DDE ITEM object has a value.

Clients use the application, topic, and item names to locate an item's value and the application and topic names to locate a topic for executing commands.

AP 145 automatically fulfills client requests to reference server applications' DDE ITEM object values. Server applications use DDE TOPIC objects to respond to client requests to update DDE ITEM object values and execute commands.

Here is an example that creates the basic components of a DDE server application:

```
[0]  SERVER;SV145;HSERVER;HTOPIC;HITEM
[1]  HSERVER←CREATEOBJ 'DDE SERVER' 'App Name'
[2]  HTOPIC←CREATEOBJ 'DDE TOPIC' HSERVER 'Topic Name'
[3]  HITEM←CREATEOBJ 'DDE ITEM' HTOPIC 'Item Name'
[4]  EVENTS←>('Write value' 'WRITE')('Execute value' 'EXECUTE')
[5]  SET_PROPERTY HITEM 'Initial value'
[6]  'EVENTS' SET_PROPERTY HTOPIC EVENTS
[7]  0 EXECUTEDLG HSERVER
[8]  DESTROYOBJ HSERVER
```

The example first creates a hierarchy of server, topic, and item objects. The server object's initialization data is the application's name. The topic object's initialization data is the server's handle and the topic's name. The item object's initialization data is the topic's handle and the item's name.

After creating the object hierarchy, the function specifies event handlers for the topic object's Write value and Execute value events. It then specifies an initial value for the item and waits for events.

When a client requests to specify a new value for an item, AP 145 sets the topic object's DDE WRITE property with the name and new value of the item. The Write value event is then signaled. Here is the WRITE function that handles the Write value event:

```
[0] WRITE; ITEM; VALUE
[1] (ITEM VALUE) ← 'DDE WRITE' GET_PROPERTY HTOPIC
[2] SET_PROPERTY HITEM VALUE
[3] 'DDE WRITE' SET_PROPERTY HTOPIC 0
```

The WRITE function retrieves the name and new value of the item from the DDE WRITE property and sets the new value for the item. It then sets the DDE WRITE property with a 0 return code to indicate the request was processed successfully. Note this example does not validate the item name.

When a client requests that a command be executed, AP 145 sets the topic object's DDE EXECUTE property with the command to be executed. The Execute value event is signaled. Here is the EXECUTE function that handles the Execute value event:

```
[0] EXECUTE; COMMAND
[1] COMMAND ← 'DDE EXECUTE' GET_PROPERTY HTOPIC
[2] ⚡COMMAND
[3] 'DDE EXECUTE' SET_PROPERTY HTOPIC (0 0)
```

The EXECUTE function retrieves and executes the command. It then sets the DDE EXECUTE property with a 2 element integer return code. The first element is 0 or 1. 0 indicates the command was executed successfully. 1 indicates the command was not executed. If the command was executed successfully, the second element is a return code from 0 to 255. Otherwise, the second element is ignored.

More information about DDE is available in the following locations:

- The chapter on DDESHARE in the User's Guide
- The DDESHARE workspace in public library 2
- The [DDESHARE](#) section of Appendix B

## **Class Reference**

Use the GUITOOLS workspace functions `CREATEDLG` and `CREATECTL` to create dialogs and controls.

AP 145 supports the following classes:

- [Dialog](#)
- [ActiveX](#)
- [Check box](#)
- [Combo box](#)
- [Custom](#)
- [Date](#)
- [Entry field](#)
- [Frame](#)
- [Graphic Windows \(AP 207\)](#)
- [Grid](#)
- [Group box](#)
- [List box](#)
- [Listview](#)
- [Menus and Menu Items](#)
- [MLE](#)
- [Month](#)
- [Progress bar](#)
- [Push button](#)
- [Radio button](#)
- [Rectangle](#)
- [Scroll bar](#)
- [Slider](#)
- [Spin button](#)
- [Tab](#)
- [Time](#)
- [Treeview](#)

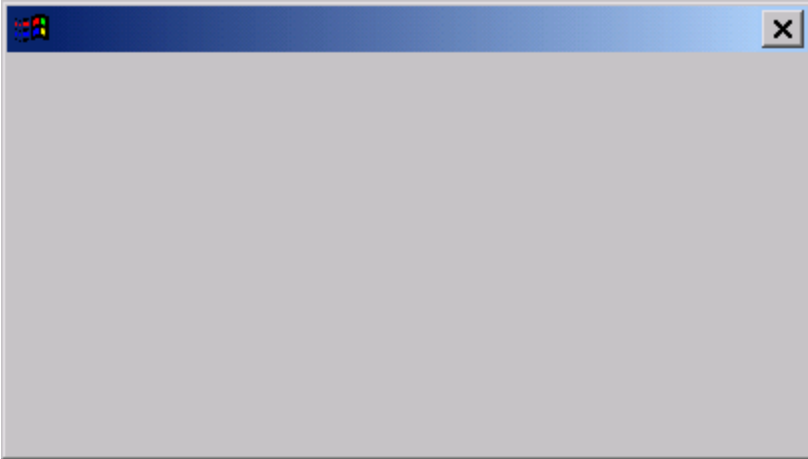
AP 145 also provides support for the following system window:

- [Desktop](#)

The following sections list the styles, properties, and events supported for dialogs and each control class. Style, property, and event names are case insensitive. Mutually exclusive styles are listed together; the defaults are shown in **bold**.

## Dialog

**Dialogs** are top-level windows that provide a consistent user interface for application management. Dialog windows typically include a border, a titlebar, and a system menu containing Move, Size, and Close. Dialogs are created with the `CREATEDLG` function.



### Styles:

Styles	Definitions
NoBorder Border <b>DlgBorder</b> SizeBorder	Has no border Has a thin border Has a 3 dimensional dialog border Has a sizing border  Note: The NoBorder and Border styles only have an effect when the NoSysMenu and NoTitlebar styles are used.
HScroll	Has a horizontal scroll bar
Max	Has a maximize button on titlebar
Min	Has a minimize button on titlebar
Modal	Is application modal; all other application windows are disabled
NoIcon	Dialog with system menu does not have an icon
NoIgnore	Do not ignore titlebar styles if the dialog is a child
NoSysMenu	Does not have a system menu on titlebar
NoTitleBar	Does not have a title bar
Shell	System supplies default size and position
Split	The dialog contains two panes arranged side by side that display the first two visible children.



SplitH	The dialog contains two panes arranged one above the other that display the first two visible children.
VScroll	Has a vertical scroll bar
Visible	Initially visible

**Class specific properties:**

Property	Description
'ACCELERATOR'	<p>4 column matrix - Defines keyboard accelerators. Accelerators provide a shortcut method for users to issue application commands.</p> <p>[;1] Integer scalar - BITWISE combination of <a href="#">Accelerator Flags</a> that indicate the required shift state and whether the code in the second column is a character, an integer, or a virtual key code.</p> <p>[;2] A character scalar, an integer scalar, or one of the <a href="#">Accelerator Virtual Key Codes</a> - An integer scalar is simply the <math>\square</math>AF value of a character. Virtual key codes are used for keys such as F3 and Home.</p> <p>[;3] Integer scalar - Identifier of the accelerator. Each identifier must be unique.</p> <p>[;4] Arbitrary array - Event handler expression</p> <p>For further information about using accelerators, see the DEMO_ACCEL function.</p>
'DATA'	Character vector - Dialog title
'MAXIMUM SIZE'	2 integers - Maximum dialog width and height
'MINIMUM SIZE'	2 integers - Minimum dialog width and height
'SPLIT'	Integer scalar - Position of bar between split style dialog's panes. Expressed as a percentage between 0 and 100 of the width or height of the dialog.
'STATE ACTIVE'	Boolean scalar - 0 inactive, 1 active
'STATE MINMAX'	Integer scalar - 0 normal size, 1 maximized, 2 minimized
'STATE RESIZING'	Read-only Boolean scalar - 1 user is resizing the dialog, 0 otherwise
'STATUS DATA'	<p>Character vector - single status bar message</p> <p>Vector of character vectors - multiple status bar messages. Must have same number of elements as STATUS PARTS property.</p>

'STATUS PARTS'	<p>2 row matrix - Defines number of parts of status area and how they are drawn. The number of columns defines the number of parts.</p> <p>[1;] Integer scalar - Specifies part width. Expressed as position of the right edge of the corresponding part in pixels. If -1, the right edge extends to the border of the window.</p> <p>[2;] Integer scalar - Specifies whether the part is drawn:</p> <p>-1 Drawn below the plane of the window  0 Drawn without borders  1 Drawn above the plane of the window</p>
'STATUS VISIBLE'	Boolean scalar - 0 invisible, 1 visible
'TEMPLATE'	<p>Character vector – Dialog template</p> <p>A template returned by the TEMPLATE property includes the child controls, presentation properties, and event handlers included in the dialog. It can be used to recreate the dialog just as if the template were created with the dialog editor. The template does not include child dialogs, graphic windows, or the dialog's menu or status area. The template property can only be referenced.</p>
'UNICODE DATA'	See Data
'UNICODE STATUS DATA'	See Status Data

Common properties:

- 'APPLICATION DATA'
- 'CLASS'
- 'CLIENT POSITION'
- 'CLIENT SIZE'
- 'COLOR BACKGROUND'
- 'COLOR FOREGROUND'
- 'CURSOR'
- 'CURSOR POSITION'
- 'DROPPED FILES'
- 'EVENTS'
- 'FONT'
- 'HORIZONTAL LIMITS'
- 'HORIZONTAL SELECTION'
- 'OFFSET'
- 'PICTURE'
- 'POSITION'
- 'RGB COLOR BACKGROUND'
- 'RGB COLOR FOREGROUND'
- 'SIZE'
- 'STATE ENABLE'
- 'STATE ENABLE DRAGDROP'

- 'STATE FOCUS'
- 'STATE VISIBLE'
- 'STYLE'
- 'UNICODE FONT'
- 'USER DATA'
- 'VERTICAL LIMITS'
- 'VERTICAL SELECTION'

**Events:**

<b>Event</b>	<b>Description</b>
'Activate'	The dialog is being activated or deactivated.
'Close'	The user is closing the window.
'Context menu'	The user has clicked the right mouse button in the window.
'Files dropped'	The user has dropped one or more files on the window
'Horizontal slider position'	The user has stopped dragging the horizontal scroll bar slider.
'Horizontal slider track'	User moves the horizontal scroll bar slider with the pointer device.
'Line down'	User clicks on the down arrow of the vertical scroll bar.
'Line left'	User clicks on the left arrow of the horizontal scroll bar.
'Line right'	User clicks on the right arrow of the horizontal scroll bar.
'Line up'	User clicks on the up arrow of the vertical scroll bar.
'Maximize'	The dialog is being maximized.
'Minimize'	The dialog is being minimized.
'Move'	The dialog is being moved.
'Mouse wheel'	The user has rotated the mouse wheel.
'Open'	Undocumented
'Page down'	User clicks on the area below slider in the vertical scroll bar.
'Page left'	User clicks on the area to the left of the slider in the horizontal scroll bar.
'Page right'	User clicks on the area to the right of the slider in the horizontal scroll bar.
'Page up'	User clicks on the area above slider in the vertical scroll bar.
'Paint'	The window needs to be painted.
'Restore'	The window is being restored.
'Size'	The dialog is being resized.
'Start menu'	The user has pulled down an application menu
'Start system menu'	The user is starting to use the system menu.
'Vertical slider position'	The user has stopped dragging the vertical scroll bar slider.

'Vertical slider track'

User moves the vertical scroll bar slider with the pointer device.

## ActiveX

An ActiveX control is a reusable software component based on the Component Object Model (COM). Many ActiveX controls are available, both as part of Windows and as products from other vendors.

ActiveX controls only operate in a special environment called a container. The container is a COM object that manages the interface between the ActiveX control and the rest of the application. AP 145 uses Microsoft Active Template Library (ATL) host windows as containers for ActiveX controls. When using an ActiveX control in AP 145, an ATL host window is a child of the dialog and the ActiveX control is within the ATL host container.

Use `CREATECTL` to add an ActiveX control to a dialog. Specify `ACTIVEX` as the control class and supply the ActiveX class name as the control data. `CREATECTL` will create an ATL host window and the ATL host window will in turn create the ActiveX control. `CREATECTL` will then return the handle of ATL window.

The ATL host window manages the interface between the dialog and the ActiveX control. For example, the host manages the ActiveX control's size and position. Use the `GET_PROPERTY` and `SET_PROPERTY` functions to access the host's properties. The host automatically propagates AP 145 property specifications to the ActiveX control.

To access the ActiveX control's events, methods, and properties directly, you need the ActiveX control's COM handle. Reference the host's 'COM OBJECT' property to retrieve the ActiveX control's COM handle. Use the COM handle with the [CALLCOM](#) function in the `GUITOOLS` workspace to access the ActiveX controls' COM methods, properties and event handlers. For more information on using COM components, see the `APL2 User's Guide` section on `COM` under `Supplied External Routines`.

### Styles:

Styles	Definitions
Border	The control is drawn with a border
Invisible	Is initially invisible
NoGroup	Does not begin a group
NoTabstop	Tab does not move focus to the control

### Notes:

- When creating an ActiveX grid control, control data must be supplied. The control data is one or two character vectors. An ActiveX class name and an optional class license.

### Class specific properties:

Property	Description
----------	-------------

'COM OBJECT'	Integer scalar - COM handle of ActiveX control. Use the value of this property with the CALLCOM function to access the ActiveX objects events, methods, and properties. When you reference the COM OBJECT property, the object's reference count will be incremented. Use the CALLCOM function's RELEASE command to decrement the reference count. The COM OBJECT property is read-only.
'CONTROL DATA'	One or two character vectors: the ActiveX class name and an optional class license key. The CONTROL DATA property is read-only.

Common properties:

- 'APPLICATION DATA'
- 'CLASS'
- 'CLIENT POSITION'
- 'CLIENT SIZE'
- 'CONTEXT HELP'
- 'CURSOR'
- 'CURSOR POSITION'
- 'EVENTS'
- 'ID'
- 'NAME'
- 'OFFSET'
- 'POSITION'
- 'SIZE'
- 'STATE ENABLE'
- 'STATE VISIBLE'
- 'STYLE'
- 'TOOL TIP'
- 'UNICODE CONTEXT HELP'
- 'UNICODE TOOL TIP'
- 'USER DATA'

**Events:**

Two types of events are supported for ActiveX controls: AP 145 events and COM events.

Use GET\_PROPERTY and SET\_PROPERTY to reference and specify AP 145 events. Use the CALLCOM function's QUERY\_EVENTS and HANDLERS commands to reference and specify COM events.

Use WAIT\_EVENT to wait for events in applications that use ActiveX controls. The EXECUTEDLG and EXECUTEDLGX operators and the CHECK\_EVENTS function do not support COM events and the CALLCOM function does not support the WAIT command.

The following table lists the AP 145 events supported for ActiveX controls:

<b>Event</b>	<b>Description</b>
'Hover'	The mouse pointer paused over the control

## Check box

A **check box** control is a small square with a text label to the right. It is normally used in groups that allow many items (or none) to be selected. Clicking on the square or the text checks or unchecks the control.



### Styles:

Styles	Definitions
3State	Has checked, unchecked, and indeterminate state
NoAuto	STATE CHECKED property is not automatically changed when user clicks
Invisible	Is initially invisible
NoGroup	Does not begin a group
NoTabstop	Tab does not move focus to the control

### Class specific properties:

Property	Description
'DATA'	Character vector - Control text
'STATE CHECKED'	Boolean scalar - 0 not checked, 1 checked
'UNICODE DATA'	See Data

### [Common properties:](#)

- 'APPLICATION DATA'
- 'CLASS'
- 'CLIENT POSITION'
- 'CLIENT SIZE'
- 'COLOR BACKGROUND'
- 'COLOR FOREGROUND'
- 'CONTEXT HELP'
- 'CURSOR'
- 'CURSOR POSITION'
- 'EVENTS'
- 'FONT'
- 'ID'
- 'NAME'
- 'OFFSET'
- 'PICTURE'
- 'POSITION'
- 'RGB COLOR BACKGROUND'



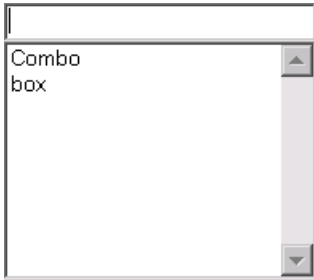
- 'RGB COLOR FOREGROUND'
- 'SIZE'
- 'STATE ENABLE'
- 'STATE FOCUS'
- 'STATE VISIBLE'
- 'STYLE'
- 'TOOL TIP'
- 'UNICODE CONTEXT HELP'
- 'UNICODE DATA'
- 'UNICODE FONT'
- 'UNICODE TOOL TIP'
- 'USER DATA'

**Events:**

<b>Event</b>	<b>Description</b>
'Button clicked'	User clicks the button.
'Button double clicked'	User double clicks the button.
'Button down'	User presses a key or mouse button.
'Button up'	User releases the key or mouse button.
'Context menu'	The user has clicked the right mouse button in the window.
'Hover'	The mouse pointer paused over the control

## Combo box

A **combo box** control is a combination of an entry field and list box, allowing the user to select an item from a list, placing the text of that item into the entry field. The list may be hidden, saving room in the dialog.



### Styles:

Styles	Definitions
<b>Simple</b>	The list box is always visible
Dropdown	The list box is initially collapsed
Dropdownlist	The list box is initially collapsed and the user cannot type a new entry in the entry field.
Sort	The list items are sorted alphabetically
HScroll	Has a horizontal scroll bar
Invisible	Is initially invisible
NoGroup	Does not begin a group
NoTabstop	Tab does not move focus to the control

### Class specific properties:

Property	Description
'COLLAPSED CLIENT POSITION'	2 integers - Position of window relative to lower left corner of parent's client area. Same as CLIENT POSITION, but based on collapsed state of drop down combo boxes.
'COLLAPSED POSITION'	2 integers - Position of window relative to lower left hand corner of parent. Same as POSITION, but based on collapsed state of drop down combo boxes.
'COLLAPSED SIZE'	2 integers - Width and height of window in pixels. Same as SIZE, but based on collapsed state of drop down combo boxes. Note: It is not possible to alter the height of drop down combo boxes. When using the COLLAPSED SIZE or SIZE properties to alter the width, specification of any height other than the existing height of the window will result in repositioning of the window without the corresponding height change.
'DATA'	Character vector - Entry field text

'DATA LIST'	Vector of character vectors - List box items
'LIMITS'	Integer scalar - Number of characters allowed in entry field
'PATH'	<p>Character vector - Path. The control displays the files in the path that match the styles specified in the path style property. For example, 'c:\temp\*.txt'.</p> <p>If a drive is not supplied, the current drive is used</p> <p>If a path is not supplied, the current directory of AP 145 is used</p> <p>If a drive or path is supplied, the current directory of AP 145 is changed.</p> <p>If a filename is supplied, it must contain at least one wildcard character, ? or *. If no filename is supplied, *.* is used.</p> <p>Combo boxes must have a non-zero identifier to use the path style property.</p>
'PATH SELECTION'	Character vector - Contains the selected item. The path selection property may be referenced but not specified. The property has a null value if no item is selected.
'PATH STYLE'	<p>Character vector - Specifies the attributes of the filenames to be displayed. Path style can contain one or more of the following words:</p> <p><b>ARCHIVE</b>      Include archived files.</p> <p><b>DIRECTORY</b>    Include subdirectories. Subdirectory names are enclosed in square brackets ([ ]).</p> <p><b>DRIVES</b>        Include drives. Drives are listed in the form [-x-], where x is the drive letter.</p> <p><b>EXCLUSIVE</b>     Include only files with the specified attributes. By default, read-write files are listed even if READWRITE is not specified.</p> <p><b>HIDDEN</b>        Include hidden files.</p> <p><b>READWRITE</b>    Include read-write files with no additional attributes.</p> <p><b>SYSTEM</b>        Include system files.</p> <p>Combo boxes must have a non-zero identifier to use the path style property.</p>
'SELECTION'	Integer vector - Indices of the selected items in origin 0
'STATE LIST VISIBLE'	<p>Boolean scalar - 0 drop down list is invisible, 1 visible.</p> <p>Note: This property has no effect on simple combo boxes.</p>

'UNICODE DATA'	See Data
'UNICODE DATA LIST'	See Data list

[Common properties:](#)

- 'APPLICATION DATA'
- 'CLASS'
- 'CLIENT POSITION'
- 'COLOR BACKGROUND'
- 'COLOR FOREGROUND'
- 'CONTEXT HELP'
- 'CURSOR'
- 'CURSOR POSITION'
- 'EVENTS'
- 'FONT'
- 'ID'
- 'NAME'
- 'OFFSET'
- 'POSITION'
- 'RGB COLOR BACKGROUND'
- 'RGB COLOR FOREGROUND'
- 'SIZE'
- 'STATE APL'
- 'STATE ENABLE'
- 'STATE FOCUS'
- 'STATE VISIBLE'
- 'STYLE'
- 'TOOL TIP'
- 'UNICODE CONTEXT HELP'
- 'UNICODE FONT'
- 'UNICODE TOOL TIP'
- 'USER DATA'

**Events:**

Event	Description
'Context menu'	The user has clicked the right mouse button in the window.
'Enter or double click'	The user has depressed the Enter key or double clicked (single clicked in the case of a drop-down list) on an item in the list box control.
'Entry field change'	The content of the entry field control has changed, and the change has been displayed on the screen.
'Hover'	The mouse pointer paused over the control
'List box select'	An item in the list box control has been selected.
'Memory error'	The control encountered an error allocating memory.

'Show list box'

The list box is about to be displayed.

## Custom

A **Custom** control is a window whose processing is performed entirely by the APL2 application. The application draws the contents of the window and handles all mouse and keyboard interactions. Custom controls can be written to fulfill specialized input or display requirements.

### Notes:

Writing custom controls requires detailed knowledge of Microsoft Windows programming using messages and the Graphics Device Interface (GDI). Information about these topics can be found in the Platform SDK documentation available at the Microsoft web site.

AP 145 automatically calls `BeginPaint` and `EndPaint` when Windows sends `WM_PAINT` messages to custom controls. Therefore, when an APL2 application receives the Paint event for a custom control, the control's invalid region has already been validated and `BeginPaint` will return a device context containing an empty invalid region. APL2 applications should use `GetDC` and `ReleaseDC` rather than `BeginPaint` and `EndPaint`.

### Styles:

Styles	Definitions
Border	The control is drawn with a border
HScroll	Has a horizontal scroll bar
Invisible	Is initially invisible
NoGroup	Does not begin a group
NoTabstop	Tab does not move focus to the control
VScroll	Has a vertical scroll bar

**Class specific properties:** None

### [Common properties:](#)

Custom controls provide normal support for most common properties. Custom controls provide special behavior for the color and font properties as described below.

If the color background or RGB color background property is set to a non-null value, the custom control provides normal support and fills the background with the specified color. If either property is set to null, the custom control does not fill the background; the application should fill the client area.

Custom controls do not use the color foreground, RGB color foreground, font, and Unicode font properties. Specifications to these properties have no automatic effect. Applications can use these properties during message processing to store color and font information.

Custom controls support the following common properties:

- 'APPLICATION DATA'
- 'CLASS'
- 'CLIENT POSITION'
- 'CLIENT SIZE'
- 'COLOR BACKGROUND'
- 'COLOR FOREGROUND'
- 'CONTEXT HELP'
- 'CURSOR'
- 'CURSOR POSITION'
- 'DROPPED FILES'
- 'EVENTS'
- 'FONT'
- 'HORIZONTAL LIMITS'
- 'HORIZONTAL SELECTION'
- 'ID'
- 'NAME'
- 'OFFSET'
- 'POSITION'
- 'RGB COLOR BACKGROUND'
- 'RGB COLOR FOREGROUND'
- 'SIZE'
- 'STATE APL'
- 'STATE ENABLE'
- 'STATE ENABLE DRAGDROP'
- 'STATE FOCUS'
- 'STATE VISIBLE'
- 'STYLE'
- 'TOOL TIP'
- 'UNICODE CONTEXT HELP'
- 'UNICODE FONT'
- 'UNICODE TOOL TIP'
- 'USER DATA'
- 'VERTICAL LIMITS'
- 'VERTICAL SELECTION'

## Events:

Custom controls' events provide applications with general classifications of Windows messages. By specifying handlers for an event, your application will be passed all the Windows messages related to that event classification. When the AP 145 `AplWaitMsg` service returns an event, the `EXECUTEDLG` operator assigns the control handle, the message identifier, and the message parameters to the `HANDLE`, `MSG`, `MP1`, and `MP2` variables before executing the event handler. The event handler can then examine these variables to determine the exact cause of the event.

The following table lists the custom control events, descriptions, and the identifiers of the Windows messages that can be returned for each type of event.

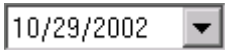
<b>Event</b>	<b>Description</b>	<b>Messages</b>
'Files dropped'	The user has dropped one or more files on the window	No message data
'Hover'	The mouse pointer paused over the control	No message data
'Horizontal slider position'	The user has stopped dragging the horizontal scroll bar slider.	WM_HSCROLL
'Horizontal slider track'	User moves the horizontal scroll bar slider with the pointer device.	WM_HSCROLL
'Keystroke'	The user has pressed a key.	WM_KEYDOWN WM_KEYUP WM_SYSKEYDOWN WM_SYSKEYUP WM_CHAR WM_SYSCHAR WM_DEADCHAR WM_SYSDEADCHAR WM_IME_CHAR
'Kill focus'	The control is losing the focus.	WM_KILLFOCUS
'Line down'	User clicks on the down arrow of the vertical scroll bar.	WM_VSCROLL
'Line left'	User clicks on the left arrow of the horizontal scroll bar.	WM_HSCROLL
'Line right'	User clicks on the right arrow of the horizontal scroll bar.	WM_HSCROLL
'Line up'	User clicks on the up arrow of the vertical scroll bar.	WM_VSCROLL
'Mouse click'	The user has pressed a mouse button.	WM_LBUTTONDOWN WM_MBUTTONDOWN WM_RBUTTONDOWN WM_LBUTTONUP WM_MBUTTONUP WM_RBUTTONUP WM_LBUTTONDBLCLK WM_MBUTTONDBLCLK WM_RBUTTONDBLCLK
'Mouse movement'	The user has moved the mouse over the control.	WM_MOUSEMOVE
'Mouse wheel'	The user has rotated the mouse wheel.	WM_MOUSEWHEEL
'Page down'	User clicks on the area below slider in the vertical scroll bar.	WM_VSCROLL
'Page left'	User clicks on the area to the left of the slider in the horizontal scroll bar.	WM_HSCROLL



'Page right'	User clicks on the area to the right of the slider in the horizontal scroll bar.	WM_HSCROLL
'Page up'	User clicks on the area above slider in the vertical scroll bar.	WM_VSCROLL
'Paint'	The control needs to be repainted.	WM_PAINT
'Set focus'	The control receives the focus.	WM_SETFOCUS
'Size'	The control has been resized.	WM_SIZE
'System color change'	The definition of one or more system colors has changed.	WM_SYSCOLORCHANGE
'Vertical slider position'	The user has stopped dragging the vertical scroll bar slider.	WM_VSCROLL
'Vertical slider track'	User moves the vertical scroll bar slider with the pointer device.	WM_VSCROLL

## Date

A **Date** control provides an interface through which to exchange date information with a user.



### Styles:

Styles	Definitions
<b>ShortDate</b> ShortDateCentury LongDate	Displays the date in short, short century, or long format
Checkbox	Checkbox included which grays date
Right	Drop down month control is right aligned
UpDown	Spin button style rather than drop down month
Invisible	Is initially invisible
NoGroup	Does not begin a group
NoTabstop	Tab does not move focus to the control

### Class specific properties:

Property	Description
'DATA'	3 Integers - Year, Month, and Day
'STATE CHECKED'	Boolean scalar - 0 not checked, 1 checked

### [Common properties:](#)

- 'APPLICATION DATA'
- 'CLASS'
- 'CLIENT POSITION'
- 'CLIENT SIZE'
- 'COLOR BACKGROUND'
- 'COLOR FOREGROUND'
- 'CONTEXT HELP'
- 'CURSOR'
- 'CURSOR POSITION'
- 'EVENTS'
- 'FONT'
- 'ID'
- 'NAME'
- 'OFFSET'

- 'POSITION'
- 'RGB COLOR BACKGROUND'
- 'RGB COLOR FOREGROUND'
- 'SIZE'
- 'STATE ENABLE'
- 'STATE FOCUS'
- 'STATE VISIBLE'
- 'STYLE'
- 'TOOL TIP'
- 'UNICODE CONTEXT HELP'
- 'UNICODE FONT'
- 'UNICODE TOOL TIP'
- 'USER DATA'

**Events:**

<b>Event</b>	<b>Description</b>
'Change'	The user has changed the date.
'Context menu'	The user has clicked the right mouse button in the window.
'Hover'	The mouse pointer paused over the control
'Kill focus'	The control is losing the focus.
'Set focus'	The control receives the focus.

The Date control class is a member of the [Microsoft Windows Common Control Library](#).

## Entry field

An **entry field** control is a rectangle in which a user can enter text.



### Styles:

Styles	Definitions
Left Center Right	The data is left justified, centered, or right justified
NoAutoscroll	Does not automatically scroll text horizontally
NoMargin	Does not have a margin
ReadOnly	The user can not modify the data
Unreadable	Displays each character as an asterisk
Invisible	Is initially invisible
NoGroup	Does not begin a group
NoTabstop	Tab does not move focus to the control

### Class specific properties:

Property	Description
'DATA'	Character vector - Control text
'LIMITS'	Integer scalar - Number of characters allowed in entry field
'SELECTION'	2 integers - 0-origin indices of the first and last characters of the selection range. If they are equal, they describe the cursor position.
'UNICODE DATA'	See Data

### [Common properties:](#)

- 'APPLICATION DATA'
- 'CLASS'
- 'CLIENT POSITION'
- 'CLIENT SIZE'
- 'COLOR BACKGROUND'
- 'COLOR FOREGROUND'
- 'CONTEXT HELP'
- 'CURSOR'
- 'CURSOR POSITION'

- 'EVENTS'
- 'FONT'
- 'ID'
- 'NAME'
- 'OFFSET'
- 'POSITION'
- 'RGB COLOR BACKGROUND'
- 'RGB COLOR FOREGROUND'
- 'SIZE'
- 'STATE APL'
- 'STATE ENABLE'
- 'STATE FOCUS'
- 'STATE READONLY'
- 'STATE VISIBLE'
- 'STYLE'
- 'TOOL TIP'
- UNICODE CONTEXT HELP
- UNICODE FONT
- UNICODE TOOL TIP
- USER DATA

**Events:**

Event	Description
'Change'	The content of the entry field control has changed, and the change has been displayed on the screen.
'Context menu'	The user has clicked the right mouse button in the window.
'Hover'	The mouse pointer paused over the control
'Kill focus'	The entry field control is losing the focus.
'Memory error'	The control encountered an error allocating memory.
'Overflow'	The entry field control cannot insert more text than the current text limit.
'Scroll'	The entry field control is about to scroll horizontally.
'Set focus'	The entry field control is receiving the focus.

## Frame

A **frame** control is a rectangular frame, used in simple graphics.



### Styles:

Styles	Definitions
<b>Foreground</b> Background Halftone	The frame is drawn using the background, the foreground, or the halftone color.
Invisible	Is initially invisible
Border	Draws a 3 dimensional border around the frame

### Common properties:

- 'APPLICATION DATA'
- 'CLASS'
- 'CLIENT POSITION'
- 'CLIENT SIZE'
- 'COLOR BACKGROUND'
- 'COLOR FOREGROUND'
- 'CONTEXT HELP'
- 'CURSOR'
- 'CURSOR POSITION'
- 'ID'
- 'NAME'
- 'OFFSET'
- PICTURE
- 'POSITION'
- 'RGB COLOR BACKGROUND'
- 'RGB COLOR FOREGROUND'
- 'SIZE'
- 'STATE ENABLE'
- 'STATE VISIBLE'
- 'STYLE'
- 'TOOL TIP'
- 'UNICODE CONTEXT HELP'
- 'UNICODE TOOL TIP'
- 'USER DATA'

## Graphic Windows (AP 207)

The AP 207 OPEN command can be used to create a graphic window in an AP 145 dialog.



Graphic windows support the following common properties:

- 'CONTEXT HELP'
- 'STATE FOCUS'
- 'UNICODE CONTEXT HELP'

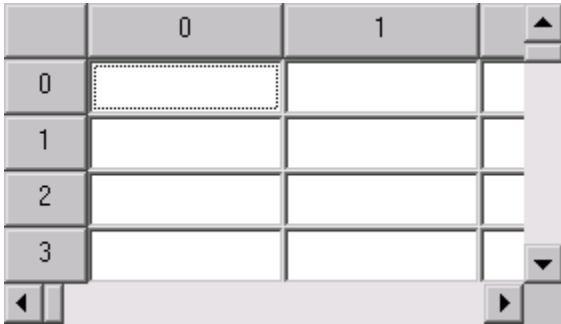
See [Displaying Graphics in a Dialog](#) for more information.

### Events:

Event	Description
'Kill focus'	The graphic window is losing the focus.
'Set focus'	The graphic window is receiving the focus.

## Grid

A **grid** is a rectangular arrangement of cells.



### Styles:

Styles	Definitions
NoMixed	Does not support numeric or non-homogeneous data
NoHScroll	Does not have a horizontal scroll bar
NoVScroll	Does not have a vertical scroll bar
NoColHead	Does not have column heading buttons
NoColSelect	Column heading buttons do not select columns
NoColResize	User can not resize columns
NoRowHead	Does not have row heading buttons
NoRowSelect	Row heading buttons do not select rows
NoRowResize	User can not resize rows
Invisible	Is initially invisible
NoGroup	Does not begin a group
NoTabstop	Tab does not move focus to the control



Resizable	<p>The user can add and delete cells using the keyboard. This style enables the following key combinations:</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <td>Alt+Left</td> <td>Insert a column before the cell that has focus.</td> </tr> <tr> <td>Alt+Right</td> <td>Insert a column after the cell that has focus.</td> </tr> <tr> <td>Alt+Up</td> <td>Insert a row before the cell that has focus.</td> </tr> <tr> <td>Alt+Down</td> <td>Insert a row after the cell that has focus.</td> </tr> <tr> <td>Ctrl+Alt+Left</td> <td>Delete the current column and move the focus left</td> </tr> <tr> <td>Ctrl+Alt+Right</td> <td>Delete the current column and move the focus right</td> </tr> <tr> <td>Ctrl+Alt+Up</td> <td>Delete the current row and move the focus up</td> </tr> <tr> <td>Ctrl+Alt+Down</td> <td>Delete the current row and move the focus down</td> </tr> </table> <p>Note: These keystrokes are not supported if the <a href="#">AplSetCellSize</a> service has been used to resize cells.</p>	Alt+Left	Insert a column before the cell that has focus.	Alt+Right	Insert a column after the cell that has focus.	Alt+Up	Insert a row before the cell that has focus.	Alt+Down	Insert a row after the cell that has focus.	Ctrl+Alt+Left	Delete the current column and move the focus left	Ctrl+Alt+Right	Delete the current column and move the focus right	Ctrl+Alt+Up	Delete the current row and move the focus up	Ctrl+Alt+Down	Delete the current row and move the focus down
Alt+Left	Insert a column before the cell that has focus.																
Alt+Right	Insert a column after the cell that has focus.																
Alt+Up	Insert a row before the cell that has focus.																
Alt+Down	Insert a row after the cell that has focus.																
Ctrl+Alt+Left	Delete the current column and move the focus left																
Ctrl+Alt+Right	Delete the current column and move the focus right																
Ctrl+Alt+Up	Delete the current row and move the focus up																
Ctrl+Alt+Down	Delete the current row and move the focus down																

Notes:

- When creating a grid control, control data can be supplied. The control data is 2 integers specifying the number of rows and columns. The default is 100 rows and columns.
- If NoRowHead is specified, NoRowSelect and NoRowResize are assumed.
- If NoColHead is specified, NoColSelect and NoColResize are assumed.
- If the CREATECTL control data argument is supplied, it is two positive integers that specify the number of rows and columns. If control data is omitted, there are 100 rows and columns.

**Class specific properties:**

Property	Description
'COLUMN HEADINGS'	Vector of character vectors - Text of the column heading buttons. Vector must have the same number of elements as the grid has columns.
'COLUMN WIDTHS'	Integer vector - Widths of the columns in pixels. Vector must have the same number of elements as the grid has columns.
'CONTROL DATA'	2 integers - Number of rows and columns
'FOCUS CELL'	2 integers - Row and column 0 origin index of the cell that has focus.
'HEIGHT COLUMN HEADINGS'	Integer scalar - Height of column heading buttons in pixels

'HORIZONTAL SLAVE'	<p>Integer scalar - Handle of a grid control whose horizontal scrolling should be controlled.</p> <p>The horizontal slave property can only be set once.</p> <p>The master and slave grid controls must have the same number of columns.</p> <p>When the horizontal slave property is set, the slave control's width is set to the same as the master control's width.</p>
'ROW HEADINGS'	<p>Vector of character vectors - Text of the row heading buttons. Vector must have the same number of elements as the grid has rows.</p>
'ROW HEIGHTS'	<p>Integer vector - Heights of the rows in pixels. Vector must have the same number of elements as the grid has rows.</p>
'SELECTION'	<p>4 integers - Selection range. The first two integers are the index of the beginning of the range. The second two integers are the number of selected rows and columns.</p>
'UNICODE COLUMN HEADINGS'	<p>See column headings</p>
'UNICODE ROW HEADINGS'	<p>See row headings</p>
'VERTICAL SLAVE'	<p>Integer scalar - handle of a grid control whose vertical scrolling should be controlled.</p> <p>The vertical slave property can only be set once.</p> <p>The master and slave grid controls must have the same number of rows.</p> <p>When the vertical slave property is set, the slave control's height is set to the same as the master control's height.</p>
'WIDTH ROW HEADINGS'	<p>Integer scalar - width of the row heading buttons in pixels</p>

When setting the size of a grid control, use the sum of the COLUMN WIDTHS and WIDTH ROW HEADINGS properties to determine the width and the sum of the ROW HEIGHTS and HEIGHT COLUMN HEADINGS to determine the height. If the grid has scroll bars, create horizontal and vertical style scroll bars and query their default sizes to determine the width and height of the grid control's scroll bars. To set a column's width accurately, create a push button, set the button's text to the longest string in a column, SIZETOTEXT the button, and query the button's size.

## Range properties:

Range Property	Description
'CELL BACKGROUND COLOR' 'CELL FOREGROUND COLOR'	Integer scalar - Background color code:  0 Default entry field color 1 Default button color 2 Default read-only entry field color 3 Black 4 Blue 5 Brown 6 Cyan 7 Dark blue 8 Dark cyan 9 Dark grey 10 Dark green 11 Dark green 12 Dark red 13 Green 14 Grey 15 Pink 16 Red 17 White 18 Yellow
'CELL DATA'	Numeric scalar, a character vector, or a matrix of numeric scalars and character vectors. If the grid has the NoMixed style, only character vectors are supported.
'CELL EDGE'	Integer scalar - Specifies which edges of cell are drawn. BITWISE combination of following values:  1 Left 2 Top 4 Right 8 Bottom

'CELL FORMAT'	Integer scalar – Specifies whether cell’s data is character or numeric.  -1 Cell is being edited (not valid with SET_RANGE) 0 Character 1 Numeric
'CELL JUSTIFICATION'	Integer scalar - Specifies how data is justified in the cell. One of the following values:  1 Left 2 Centered 3 Right
'CELL STATE PUSHED'	Boolean scalar - Push Button cell state. 0 not pushed, 1 pushed
'CELL STATE READONLY'	Boolean scalar - Entry field and button enable state. 0 input allowed, 1 read-only
'CELL STATE WORDWRAP'	Boolean scalar - 0 text drawn in single line, 1 text wrapped onto multiple lines
'CELL TYPE'	Integer scalar - One of the following values:  0 Static text 1 Entry field 2 Push button
'UNICODE CELL DATA'	See cell data

Common properties:

- 'APPLICATION DATA'
- 'CLASS'
- 'CLIENT POSITION'
- 'CLIENT SIZE'
- 'COLOR BACKGROUND'
- 'CONTEXT HELP'
- 'CURSOR'
- 'CURSOR POSITION'
- 'EVENTS'
- 'FONT'
- 'ID'
- 'NAME'
- 'OFFSET'
- 'POSITION'
- 'RGB COLOR BACKGROUND'

- 'SIZE'
- 'STATE APL'
- 'STATE ENABLE'
- 'STATE FOCUS'
- 'STATE VISIBLE'
- 'STYLE'
- 'TOOL TIP'
- 'UNICODE CONTEXT HELP'
- 'UNICODE FONT'
- 'UNICODE TOOL TIP'
- 'USER DATA'

**Events:**

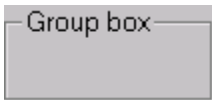
Event	Description
'Begin edit'	The user has started editing a cell. The cell has been replaced with an MLE. The handle of the MLE may be retrieved using WINDOWFROMID; the grid is the MLE's parent and the MLE's id is zero.
'Button'	A button cell has been pushed.
'Change'	The user has changed a cell's value. Query the MLE's DATA property to retrieve the changed value. <b>Note:</b> This event is signaled before the End Edit event.
'Column button'	The user has pushed a column heading button. This event is only signaled if the NoColSelect style is used.
'Context menu'	The user has clicked the right mouse button in the window.
'End edit'	The user has finished editing a cell and saved any changes made. <b>Note:</b> To determine the index of the edited cell, do not query the focus cell property in response to the end edit event, because the focus may move before the end edit event is received. To determine the index of the edited cell, use the MP1 and MP2 message parameters or query the focus cell property in response to the begin edit event.
'Focus move'	The user has moved the focus to another cell.
'Format change'	The user has hit enter to end editing a cell with the numeric CELL FORMAT property and the new value can not be converted to a number. The cell's format has been changed to character. <b>Note:</b> This event is signaled before the End Edit event.
'Hover'	The mouse pointer paused over the control
'Row button'	The user has pushed a row heading button. This event is only signaled if the NoRowSelect style is used.

When a cell event occurs and an event handler is supplied, EXECUTEDLG sets MP1 and MP2 to the row and column index of the focus cell.

Normally **Grid** controls handle heading button presses. However, when the no row selection or no column selection styles are used, and a heading button event occurs, EXECUTEDLG sets the variable MP1 to the index of the heading button.

## Group box

A **group box** is a frame used for grouping controls together. It has a text label in its top edge.



### Styles:

Styles	Definitions
Invisible	Is initially invisible

### Class specific properties:

Property	Description
'DATA'	Character vector - Control text
'CLIENT ORIGIN'	2 integers - Position of the lower left hand corner of the group box's client area relative to the lower left hand corner of parent dialog's client area. Reference the CLIENT ORIGIN property to determine where to position controls within the group box. Specify the CLIENT ORIGIN to position the group box over other controls. The DEMO_GROUP function demonstrates using the CLIENT ORIGIN property.
'UNICODE DATA'	See data

### [Common properties:](#)

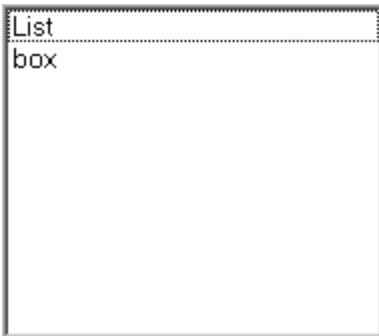
- 'APPLICATION DATA'
- 'CLASS'
- 'CLIENT POSITION'
- 'CLIENT SIZE'
- 'COLOR BACKGROUND'
- 'COLOR FOREGROUND'
- 'CONTEXT HELP'
- 'CURSOR'
- 'CURSOR POSITION'
- 'FONT'
- 'ID'
- 'NAME'
- 'OFFSET'
- 'POSITION'
- 'RGB COLOR BACKGROUND'
- 'RGB COLOR FOREGROUND'
- 'SIZE'

- 'STATE ENABLE'
- 'STATE VISIBLE'
- 'STYLE'
- 'TOOL TIP'
- 'UNICODE CONTEXT HELP'
- 'UNICODE FONT'
- 'UNICODE TOOL TIP'
- 'USER DATA'



## List box

A **list box** control is a rectangle with a vertical scroll bar on its right side. It is used to display a list of character vectors, such as file names.



### Styles:

Styles	Definitions
MultipleSel	The user can select multiple items
ExtendedSel	The user can select multiple ranges of items
Sort	The list items are sort alphabetically
HScroll	Has a horizontal scroll bar
Invisible	Is initially invisible
NoGroup	Does not begin a group
NoTabstop	Tab does not move focus to the control

### Class specific properties:

Property	Description
'DATA'	Vector of character vectors - List box items

'PATH'	<p>Character vector - Path. The control displays the files in the path that match the styles specified in the path style property. For example, 'c:\temp\*.txt'.</p> <p>If a drive is not supplied, the current drive is used</p> <p>If a path is not supplied, the current directory of AP 145 is used</p> <p>If a drive or path is supplied, the current directory of AP 145 is changed.</p> <p>If a filename is supplied, it must contain at least one wildcard character, ? or *. If no filename is supplied, *.* is used.</p> <p>List boxes must have a non-zero identifier to use the path style property.</p>
'PATH SELECTION'	<p>Character vector - Contains the selected item. The path selection property may be referenced but not specified. The property has a null value if no item is selected.</p>
'PATH STYLE'	<p>Character vector - Specifies the attributes of the filenames to be displayed. Path style can contain one or more of the following words:</p> <p>ARCHIVE      Include archived files.</p> <p>DIRECTORY    Include subdirectories. Subdirectory names are enclosed in square brackets ([ ]).</p> <p>DRIVES        Include drives. Drives are listed in the form [-x-], where x is the drive letter.</p> <p>EXCLUSIVE    Include only files with the specified attributes. By default, read-write files are listed even if READWRITE is not specified.</p> <p>HIDDEN        Include hidden files.</p> <p>READWRITE    Include read-write files with no additional attributes.</p> <p>SYSTEM        Include system files.</p> <p>List boxes must have a non-zero identifier to use the path style property.</p>
'SELECTION'	<p>Integer vector - 0 origin indices of selected items</p>
'TOP INDEX'	<p>Integer scalar - 0 origin index of item displayed at top of control</p>
'UNICODE DATA'	<p>See Data</p>

[Common properties:](#)

➤ 'APPLICATION DATA'

- 'CLASS'
- 'CLIENT POSITION'
- 'CLIENT SIZE'
- 'COLOR BACKGROUND'
- 'COLOR FOREGROUND'
- 'CONTEXT HELP'
- 'CURSOR'
- 'CURSOR POSITION'
- 'EVENTS'
- 'FONT'
- 'ID'
- 'NAME'
- 'OFFSET'
- 'POSITION'
- 'RGB COLOR BACKGROUND'
- 'RGB COLOR FOREGROUND'
- 'SIZE'
- 'STATE ENABLE'
- 'STATE FOCUS'
- 'STATE VISIBLE'
- 'STYLE'
- 'TOOL TIP'
- 'UNICODE CONTEXT HELP'
- 'UNICODE FONT'
- 'UNICODE TOOL TIP'
- 'USER DATA'

**Events:**

<b>Event</b>	<b>Description</b>
'Context menu'	The user has clicked the right mouse button in the window.
'Enter or double click'	The user has depressed the Enter key or double clicked on an item in the list box control.
'Hover'	The mouse pointer paused over the control
'Kill focus'	The list box control is losing the focus.
'Select'	An item is being selected or deselected.
'Set focus'	The list box control is gaining the focus.

## Listview

A **Listview** control is a window that displays a collection of items; each item consists of an icon and a label. Listview controls provide several ways to arrange and display items. For example, additional information about each item can be displayed in columns to the right of the icon and label.



### Styles:

Styles	Definitions
NoHeadings	Does not have column headings
StaticHeadings	Column headings are drawn as static text
DragHeadings	The user can rearrange the order of the columns
Editlabels	Allows the user to edit item labels
Left	Align items along the left in icon view
NoWrap	Icon text does not wrap into multiple lines
SingleSel	Prevents selection of multiple items
FullRowSel	Highlight entire rows in details view
GridLines	Display grid lines in details view
CheckBoxes	Display check boxes next to items
Invisible	Is initially invisible
NoGroup	Does not begin a group
NoTabstop	Tab does not move focus to the control

Notes:

- If NoHeadings is specified, StaticHeadings and DragHeadings cannot be used.

### Class specific properties:

Property	Description
'DATA'	Matrix of character vectors - Displayed data
'FOCUS'	Integer scalar or empty vector - 0 origin index of item with focus
'HEADING SELECTION'	Integer scalar - 0 origin index of the last selected listview column heading. Specifications are discarded.

Property	Description																		
'HEADINGS'	Vector of character vectors - Column headings																		
'JUSTIFY'	<p>Integer vector - Specifies the justification of column headings and data. Must have the same number of elements as the headings property. Use the following justification values:</p> <p>-1 Left justified  0 Centered  1 Right justified</p>																		
'PICTURE INDEXES'	<p>Integer vector - 0 origin indices, into the listview control's picture list, of the pictures to be displayed next to each item. Must have the same number of elements as the number of rows in the listview control's data property. Use -1 for no picture.</p>																		
'PICTURE LIST'	<p>Array of pictures used in a listview control. Each element can be any of 4 types of arrays: character vectors containing names of image files, enclosed character vectors containing the contents of image files, 3 element arrays containing image resource information, or integer scalars that are built-in picture codes.</p> <p>Character vector - Name of image file. The following image file types are supported:</p> <table border="1" data-bbox="651 1073 1344 1415"> <tbody> <tr> <td>Bitmap</td> <td>BMP</td> </tr> <tr> <td>Enhanced Metafile</td> <td>EMF</td> </tr> <tr> <td>Exchangeable Image File</td> <td>Exif</td> </tr> <tr> <td>Graphics Interchange Format</td> <td>GIF</td> </tr> <tr> <td>Icon</td> <td>ICO</td> </tr> <tr> <td>Joint Photographic Experts Group</td> <td>JPG or JPEG</td> </tr> <tr> <td>Portable Network Graphics</td> <td>PNG</td> </tr> <tr> <td>Tag Image File Format</td> <td>TIF or TIFF</td> </tr> <tr> <td>Windows Metafile</td> <td>WMF</td> </tr> </tbody> </table> <p><a href="#">GDI+</a> is required for file types other than bitmap and icon.</p> <p>If a full path is not supplied, files are searched for using the following sequence:</p> <ol style="list-style-type: none"> <li>1. The directory from which AP 145 was loaded</li> <li>2. The current directory</li> <li>3. The Windows system directory</li> <li>4. The directories listed in the PATH environment variable</li> </ol>	Bitmap	BMP	Enhanced Metafile	EMF	Exchangeable Image File	Exif	Graphics Interchange Format	GIF	Icon	ICO	Joint Photographic Experts Group	JPG or JPEG	Portable Network Graphics	PNG	Tag Image File Format	TIF or TIFF	Windows Metafile	WMF
Bitmap	BMP																		
Enhanced Metafile	EMF																		
Exchangeable Image File	Exif																		
Graphics Interchange Format	GIF																		
Icon	ICO																		
Joint Photographic Experts Group	JPG or JPEG																		
Portable Network Graphics	PNG																		
Tag Image File Format	TIF or TIFF																		
Windows Metafile	WMF																		

Property	Description
	<p>Enclosed character vector - Contents of image file. Icon file images are not supported. <a href="#">GDI+</a> is required for this type of array.</p> <p>Three element array - Image resource</p> <p>[1] Character vector - Resource type: 'BITMAP' or 'ICON'  [2] Character vector – Name of executable containing resource.  For example: 'SHELL32 .DLL'  [3] Integer scalar - Resource identifier</p> <p>Integer scalar – Built-in picture code:</p> <ul style="list-style-type: none"> <li>0 Closed folder</li> <li>1 Open folder</li> <li>2 Closed book</li> <li>3 Open book</li> <li>4 Page</li> </ul> <p>Note: The listview picture list property does not respect transparent backgrounds in pictures. For best results, use bitmaps and icons with white backgrounds.</p>
'SELECTION'	Integer vector - 0 origin indices of selected items
'STATE CHECKED'	<p>Integer vector - Specifies whether a check box appears next to an item and whether it is checked or unchecked. The state checked property is only used with listview controls with the CheckBoxes style. The state checked property must have the same number of elements as the number of rows in the listview's data property. Its values mean:</p> <ul style="list-style-type: none"> <li>0 No check box</li> <li>1 Empty check box</li> <li>2 Checked check box</li> </ul>
'TOP INDEX'	Integer scalar - 0 origin index of item displayed at top of control
'UNICODE DATA' 'UNICODE HEADINGS'	<p>See Data and Headings</p> <p>APL characters can not be displayed in listview controls in dialogs created by the CREATEDLG function.</p> <p>Listview controls always store data in Unicode. When a listview's DATA or HEADINGS property is specified, the control uses Windows' facilities</p>

Property	Description
	<p>to convert the data to Unicode. Windows uses the current input locale to interpret the data. Since Windows locales do not include APL characters, characters at those □AV positions are converted to Unicode national language characters.</p> <p>To display APL characters in listview controls, use the UNICREATEDLG function to create a Unicode dialog and set the UNICODE DATA and UNICODE HEADINGS properties. AP 145 will convert the APL data to Unicode.</p> <p>For more information about using Unicode, see the Double Byte Character Set Support appendix in the APL2 User’s Guide.</p>
'VIEW'	<p>Integer scalar - Current view. Use one of the following values:</p> <p>0 Icons view  1 Small icons view  2 List view  3 Details view</p>
'WIDTHS'	<p>Integer vector - Widths of column headings in pixels. Must have the same number of elements as the headings property.</p>

Common properties:

- 'APPLICATION DATA'
- 'CLASS'
- 'CLIENT POSITION'
- 'CLIENT SIZE'
- 'COLOR BACKGROUND'
- 'COLOR FOREGROUND'
- 'CONTEXT HELP'
- 'CURSOR POSITION'
- 'CURSOR'
- 'DROPPED FILES'
- 'EVENTS'
- 'FONT'
- 'ID'
- 'NAME'
- 'OFFSET'
- 'POSITION'
- 'RGB COLOR BACKGROUND'
- 'RGB COLOR FOREGROUND'
- 'SIZE'
- 'STATE APL'

- 'STATE ENABLE'
- 'STATE ENABLE DRAGDROP'
- 'STATE FOCUS'
- 'STATE VISIBLE'
- 'STYLE'
- 'TOOL TIP'
- 'UNICODE CONTEXT HELP'
- 'UNICODE DATA'
- 'UNICODE FONT'
- 'UNICODE TOOL TOP'

**Events:**

Event	Description
'Check state change'	The user checked or cleared an item's checkbox
'Column click'	The user has clicked on a column heading.
'Files dropped'	The user has dropped one or more files on the window
'Hover'	The mouse pointer paused over the control
'Kill focus'	The listview control is losing the focus.
'Enter or double click'	The user has pressed Enter or double clicked.
'Label changed'	The user has edited an item's label. Reference the control's DATA property to retrieve the item's modified label.
'Right click'	The user has right clicked on an item.
'Select'	One or more items has been selected or deselected.
'Set focus'	The listview control is gaining the focus.

The Listview control class is a member of the [Microsoft Windows Common Control Library](#).



## Menus and Menu Items

Menus are created with the CREATEMENU function. CREATEMENU returns a menu handle.

Menus support the following properties.

- 'APPLICATION DATA'
- 'EVENTS'
- 'USER DATA'

Menu items support the following properties:

- 'CONTEXT HELP'
- 'DATA'
- 'STATE CHECKED'
- 'STATE ENABLE'
- 'UNICODE CONTEXT HELP'
- 'UNICODE DATA'

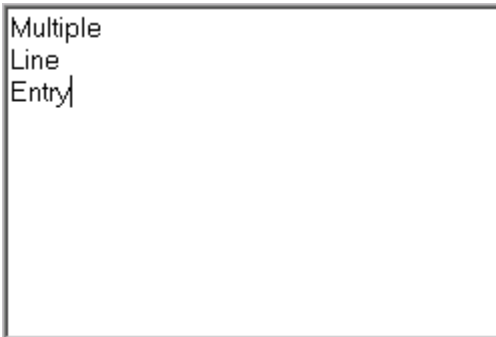
Set a menu item's property using the menu's handle and the menu item's identifier. For example, the following statements create a menu and disable the Quit menu item:

```
MENU←CREATEMENU DIALOG(,c((100 'File')(120 'Quit')))  
'STATE ENABLE' SET_PROPERTY MENU 0 120
```

See the DEMO\_MENU function and [Adding a Menu Bar](#) for further information about using menus.

## MLE

An **MLE** control is an entry field that allows the text to be entered on multiple lines.



### Styles:

Styles	Definitions
NoBorder	Does not have a border
ReadOnly	The user can not modify the data
WordWrap	Wraps data onto multiple lines
Hscroll	Has a horizontal scroll bar
Vscroll	Has a vertical scroll bar
Invisible	Is initially invisible
NoGroup	Does not begin a group
NoTabstop	Tab does not move focus to the control

### Class specific properties:

Property	Description
'DATA'	Vector of character vectors - Control text
'LIMITS'	Integer scalar - Number of characters allowed in control
'SELECTION'	Pair of integer pairs - Defines the first and last characters of the selection range. The first number in each pair is a record index. The second number in each pair is an index of a character within the record. Indices are in index origin zero. If the pairs are the same, they describe the cursor position.
'TOP INDEX'	Integer scalar - 0 origin index of item displayed at top of control
'UNICODE DATA'	See Data

### [Common properties:](#)

- 'APPLICATION DATA'
- 'CLASS'
- 'CLIENT POSITION'
- 'CLIENT SIZE'
- 'COLOR BACKGROUND'
- 'COLOR FOREGROUND'
- 'CONTEXT HELP'
- 'CURSOR'
- 'CURSOR POSITION'
- 'DROPPED FILES'
- 'EVENTS'
- 'FONT'
- 'ID'
- 'NAME'
- 'OFFSET'
- 'POSITION'
- 'RGB COLOR BACKGROUND'
- 'RGB COLOR FOREGROUND'
- 'SIZE'
- 'STATE APL'
- 'STATE ENABLE'
- 'STATE ENABLE DRAGDROP'
- 'STATE FOCUS'
- 'STATE READONLY'
- 'STATE VISIBLE'
- 'STYLE'
- 'TOOL TIP'
- 'UNICODE CONTEXT HELP'
- 'UNICODE FONT'
- 'UNICODE TOOL TIP'
- 'USER DATA'

**Events:**

Event	Description
'Change'	The content of the MLE has changed, and the change has been displayed on the screen.
'Context menu'	The user has clicked the right mouse button in the window.
'Files dropped'	The user has dropped one or more files on the window
'Horizontal scroll'	The MLE has completed a scrolling calculation and is about to update the display accordingly. All queries return values as if the scrolling were complete. However, no scrolling action is visible on the user interface.
'Hover'	The mouse pointer paused over the control
'Kill focus'	The MLE is losing the focus.

'Memory error'	The control encountered an error allocating memory.
'Set focus'	The MLE is receiving the focus.
'Vertical scroll'	The MLE has completed a scrolling calculation and is about to update the display accordingly. All queries return values as if the scrolling were complete. However, no scrolling action is visible on the user interface.

## Month

A **Month** control implements a calendar-like interface. This provides the user with a way to select a date or a range of dates.



### Styles:

Styles	Definitions
MultipleSel	User can select range of dates
NoToday	Do not display today's date at bottom
NoTodayCircle	Do not circle today's date in calendar
WeekNumbers	Display week numbers
Invisible	Is initially invisible
NoGroup	Does not begin a group
NoTabstop	Tab does not move focus to the control

### Class specific properties:

Property	Description
'DATA'	For single selection month controls, the data property is a 3-element vector containing the year, month, and day. For multiple selection month controls, the data property is a 2 by 3 matrix containing the beginning and ending years, months, and days.

Property	Description
'FIRST DAY OF WEEK'	Integer scalar - Specifies which day is the first day of the week. One of the following values:  0 Monday 1 Tuesday 2 Wednesday 3 Thursday 4 Friday 5 Saturday 6 Sunday
'HIGHLIGHTED DAYS'	Integer matrix, 3 columns - Specifies days that should be highlighted. The columns are:  [;1] Year [;2] Month [;3] Day
'MAXIMUM SELECTION COUNT'	Integer scalar - Maximum number of days that may be selected
'MONTH DELTA'	Integer scalar - Specifies how many months the control should scroll when the user presses a scroll button.
'RANGE'	Integer matrix, 2 rows, 3 columns - Specifies minimum and maximum allowable dates in control. The first row is the minimum date. The second row is the maximum date. The columns are:  [;1] Year [;2] Month [;3] Day
'TODAY'	Integer vector - Year, month, and day that should be specified as today.

Common properties:

- 'APPLICATION DATA'
- 'CLASS'
- 'CLIENT POSITION'
- 'CLIENT SIZE'
- 'COLOR BACKGROUND'
- 'COLOR FOREGROUND'
- 'CONTEXT HELP'

- 'CURSOR POSITION'
- 'CURSOR'
- 'EVENTS'
- 'FONT'
- 'ID'
- 'NAME'
- 'OFFSET'
- 'POSITION'
- 'RGB COLOR BACKGROUND'
- 'RGB COLOR FOREGROUND'
- 'SIZE'
- 'STATE ENABLE'
- 'STATE FOCUS'
- 'STATE VISIBLE'
- 'STYLE'
- 'TOOL TIP'
- 'UNICODE CONTEXT HELP'
- 'UNICODE FONT'
- 'UNICODE TOOL TIP'
- 'USER DATA'

**Events:**

Event	Description
'Context menu'	The user has clicked the right mouse button in the window.
'Hover'	The mouse pointer paused over the control
'Selection change'	The selected date or range of dates has changed.
'Select'	The user has selected a date.

The Month control class is a member of the [Microsoft Common Control Library](#).

## Progress Bar

A **Progress Bar** control displays the progress of a lengthy operation.



### Styles:

Styles	Definitions
Marquee	The progress bar moves like a marquee.
Smooth	Displays progress status in a smooth scrolling bar rather than the default segmented bar.
Vertical	Displays progress status vertically, from bottom to top.
Invisible	Is initially invisible
NoGroup	Does not begin a group
NoTabstop	Tab does not move focus to the control

### Class specific properties:

Property	Description
'LIMITS'	2 integers - Specifies range of progress bar
'SELECTION'	Integer scalar - Value of progress bar

### [Common properties:](#)

- 'APPLICATION DATA'
- 'CLASS'
- 'CLIENT POSITION'
- 'CLIENT SIZE'
- 'COLOR BACKGROUND'
- 'COLOR FOREGROUND'
- 'CONTEXT HELP'
- 'CURSOR'
- 'CURSOR POSITION'
- 'ID'
- 'NAME'
- 'OFFSET'
- 'PICTURE'
- 'POSITION'
- 'RGB COLOR BACKGROUND'
- 'RGB COLOR FOREGROUND'
- 'SIZE'



- 'STATE ENABLE'
- 'STATE VISIBLE'
- 'STYLE'
- 'TOOL TIP'
- 'UNICODE CONTEXT HELP'
- 'UNICODE TOOL TIP'
- 'USER DATA'

The Progress Bar control class is a member of the [Microsoft Common Control Library](#).

## Push button

A **push button** control is a rectangle with rounded corners, containing a text label or picture. Clicking on it with the mouse or pressing Enter or the Space bar while it has focus signals an event.



### Styles:

Styles	Definitions
Default	Button is pressed when Enter is hit. This style requires a non-zero id.
Invisible	Is initially invisible
NoGroup	Does not begin a group
NoTabstop	Tab does not move focus to the control

### Notes:

- If CREATECTL Id parameter is 1, button text is set to Ok
- If CREATECTL Id parameter is 2, button text is set to Cancel
- Windows issues an id 1 button push if the user presses Enter
- Windows issues an id 2 button push if the user presses Escape

### Class specific properties:

Property	Description
'DATA'	Character vector - Control text
'UNICODE DATA'	See data

### [Common properties:](#)

- 'APPLICATION DATA'
- 'CLASS'
- 'CLIENT POSITION'
- 'CLIENT SIZE'
- 'COLOR BACKGROUND'
- 'COLOR FOREGROUND'
- 'CONTEXT HELP'
- 'CURSOR'
- 'CURSOR POSITION'
- 'EVENTS'
- 'FONT'
- 'ID'

- 'NAME'
- 'OFFSET'
- 'PICTURE'
- 'POSITION'
- 'RGB COLOR BACKGROUND'
- 'RGB COLOR FOREGROUND'
- 'SIZE'
- 'STATE ENABLE'
- 'STATE FOCUS'
- 'STATE VISIBLE'
- 'STYLE'
- 'TOOL TIP'
- 'UNICODE CONTEXT HELP'
- 'UNICODE FONT'
- 'UNICODE TOOL TIP'
- 'USER DATA'

**Events:**

<b>Event</b>	<b>Description</b>
'Command'	User presses the button.
'Context menu'	The user has clicked the right mouse button in the window.
'Button down'	User presses a key or mouse button.
'Button up'	User releases the key or mouse button.
'Hover'	The mouse pointer paused over the control

## Radio button

A **radio button** control is a small circle with a text label to the right. It is normally used in groups that allow only one item at a time to be selected. Clicking on the circle or the text chooses that option instead of any other one in the group.



Styles:

Styles	Definitions
NoAuto	STATE CHECKED property is not automatically changed when user clicks
Invisible	Is initially invisible
NoGroup	Does not begin a group
NoTabstop	Tab does not move focus to the control

### Class specific properties:

Property	Description
'DATA'	Character vector - Control text
'STATE CHECKED'	Boolean scalar - 0 not checked, 1 checked
'UNICODE DATA'	See Data

### [Common properties:](#)

- 'APPLICATION DATA'
- 'CLASS'
- 'CLIENT POSITION'
- 'CLIENT SIZE'
- 'COLOR BACKGROUND'
- 'COLOR FOREGROUND'
- 'CONTEXT HELP'
- 'CURSOR'
- 'CURSOR POSITION'
- 'EVENTS'
- 'FONT'
- 'ID'
- 'NAME'
- 'OFFSET'
- 'PICTURE'
- 'POSITION'
- 'RGB COLOR BACKGROUND'
- 'RGB COLOR FOREGROUND'

- 'SIZE'
- 'STATE CHECKED'
- 'STATE ENABLE'
- 'STATE FOCUS'
- 'STATE VISIBLE'
- 'STYLE'
- 'TOOL TIP'
- 'UNICODE CONTEXT HELP'
- 'UNICODE FONT'
- 'UNICODE TOOL TIP'
- 'USER DATA'

**Events:**

<b>Event</b>	<b>Description</b>
'Button clicked'	User clicks the button.
'Button double clicked'	User double clicks the button.
'Button down'	User presses a key or mouse button.
'Button up'	User releases the key or mouse button.
'Context menu'	The user has clicked the right mouse button in the window.
'Hover'	The mouse pointer paused over the control

## Rectangle

A **rectangle** control is a colored rectangle, typically used in simple graphics. By flattening the rectangle to a line, it is also useful for making a separating line or bar between controls.



### Styles:

Styles	Definitions
Foreground Background Halftone	The frame is drawn using the Foreground, the background, or the halftone color.
Invisible	Is initially invisible
Border	Draws a 3 dimensional border around the rectangle

### Common properties:

- 'APPLICATION DATA'
- 'CLASS'
- 'CLIENT POSITION'
- 'CLIENT SIZE'
- 'COLOR BACKGROUND'
- 'COLOR FOREGROUND'
- 'CONTEXT HELP'
- 'CURSOR'
- 'CURSOR POSITION'
- 'ID'
- 'NAME'
- 'OFFSET'
- 'PICTURE'
- 'POSITION'
- 'RGB COLOR BACKGROUND'
- 'RGB COLOR FOREGROUND'
- 'SIZE'
- 'STATE ENABLE'
- 'STATE VISIBLE'
- 'STYLE'
- 'TOOL TIP'
- 'UNICODE CONTEXT HELP'
- 'UNICODE TOOL TIP'
- 'USER DATA'

## Scroll bar

A **scroll bar** lets the user scroll the contents of a window. Scroll bars have a scrolling arrow at each end, and a slider that can be set at any point in the bar.



### Styles:

Styles	Definitions
Horizontal Vertical	The scroll bar is drawn horizontally or vertically
Invisible	Is initially invisible
NoGroup	Does not begin a group
NoTabstop	Tab does not move focus to the control

### Class specific properties:

Property	Description
'LIMITS'	2 integers - Specifies range of scroll bar's value
'SELECTION'	Integer scalar - Value of scroll bar

### [Common properties:](#)

- 'APPLICATION DATA'
- 'CLASS'
- 'CLIENT POSITION'
- 'CLIENT SIZE'
- 'COLOR BACKGROUND'
- 'COLOR FOREGROUND'
- 'CONTEXT HELP'
- 'CURSOR'
- 'CURSOR POSITION'
- 'EVENTS'
- 'ID'
- 'NAME'
- 'OFFSET'
- 'POSITION'
- 'RGB COLOR BACKGROUND'
- 'RGB COLOR FOREGROUND'
- 'SIZE'
- 'STATE ENABLE'
- 'STATE FOCUS'

- 'STATE VISIBLE'
- 'STYLE'
- 'TOOL TIP'
- 'UNICODE CONTEXT HELP'
- 'UNICODE TOOL TIP'
- 'USER DATA'

**Events:**

<b>Event</b>	<b>Description</b>
'Context menu'	The user has clicked the right mouse button in the window.
'Hover'	The mouse pointer paused over the control
'Line left'	User clicks on the left arrow of the scroll bar, or depresses the left arrow key.
'Line right'	User clicks on the right arrow of the scroll bar, or depresses the right arrow key.
'Page left'	User clicks on the area to the left of the slider, or depresses the page up key.
'Page right'	User clicks on the area to the right of the slider, or depresses the page right key.
'Line up'	User clicks on the up arrow of the scroll bar, or depresses the up arrow key.
'Line down'	User clicks on the down arrow of the scroll bar, or depresses the down arrow key.
'Page up'	User clicks on the area above the slider, or depresses the page up key.
'Page down'	User clicks on the area below the slider, or depresses the page down key.
'Slider track'	User moves the slider with the pointer device.
'End scroll'	User has finished scrolling, but only if the user has not been doing any absolute slider positioning.
'Slider position'	Indicates the final position of the slider.
'Mouse wheel'	The user has rotated the mouse wheel.



## Slider

A **slider** control is a window that allows selection of a value from a range.



### Styles:

Styles	Definitions
<b>Horizontal</b> Vertical	The scroll bar is drawn horizontally or vertically
Top <b>Bottom</b>	For horizontal sliders, the tick marks are drawn above or below the slider
Left <b>Right</b>	For vertical sliders, the tick marks are drawn on the left or right side of the slider
<b>Ruler1</b> Ruler2	The first or second set of increments is used.
Invisible	Is initially invisible
NoGroup	Does not begin a group
NoTabstop	Tab does not move focus to the control

### Notes:

- If the CREATECTL CtlData argument is supplied, it is two positive integers that specify the number of increments in scales 1 and 2. If CtlData is omitted, both scales 1 and 2 have 100 increments.

### Class specific properties:

Property	Description
'CONTROL DATA'	Integer vector - 4 elements:  [1] Scale 1 number of increments [2] Scale 1 spacing [3] Scale 2 number of increments [4] Scale 2 spacing

<b>Property</b>	<b>Description</b>
'DIMENSIONS'	Integer vector - 3 elements  [1] Shaft breadth [2] Arm breadth [3] Arm length  Ignored on Windows
'SCALE'	Integer scalar - Active scale. Value is 1 or 2
'SCALE 1 TICKMARK SIZES'	Integer vector - Scale 1 tick mark sizes in pixels. 0 indicates no tick mark. Length must equal number of increments specified in Control Data.  Tick marks are fixed size on Windows.
'SCALE 1 TICKMARK TEXT'	Vector of character vectors - Scale 1 tick mark labels. Zero length character vectors indicate no label. Length must equal number of increments specified in Control Data.  Tick mark text properties are ignored on Windows.
'SCALE 2 TICKMARK SIZES'	Integer vector - Scale 2 tick mark sizes in pixels. 0 indicates no tick mark. Length must equal number of increments specified in Control Data.  Tick marks are fixed size on Windows.
'SCALE 2 TICKMARK TEXT'	Vector of character vectors - Scale 1 tick mark labels. Zero length character vectors indicate no label. Length must equal number of increments specified in Control Data.  Tick mark text properties are ignored on Windows.
'SELECTION'	Integer scalar - Value of slider
'SHAFT POSITION'	Two Integers - Specifies position of the lower left corner of the shaft relative to the slider window in pixels.  Shaft position property is ignored on Windows.

[Common properties:](#)

- 'APPLICATION DATA'
- 'CLASS'
- 'CLIENT POSITION'

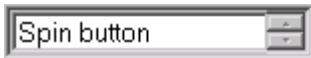
- 'CLIENT SIZE'
- 'COLOR BACKGROUND'
- 'COLOR FOREGROUND'
- 'CONTEXT HELP'
- 'CURSOR'
- 'CURSOR POSITION'
- 'EVENTS'
- 'ID'
- 'NAME'
- 'OFFSET'
- 'POSITION'
- 'RGB COLOR BACKGROUND'
- 'RGB COLOR FOREGROUND'
- 'SIZE'
- 'STATE ENABLE'
- 'STATE FOCUS'
- 'STATE READONLY'
- 'STATE VISIBLE'
- 'STYLE'
- 'TOOL TIP'
- 'UNICODE CONTEXT HELP'
- 'UNICODE TOOL TIP'
- 'USER DATA'

**Events:**

<b>Event</b>	<b>Description</b>
'Change'	The slider arm position has changed.
'Context menu'	The user has clicked the right mouse button in the window.
'Hover'	The mouse pointer paused over the control
'Kill focus'	The slider control is losing the focus.
'Set focus'	The slider control is receiving the focus.
'Slider track'	The slider arm is being dragged, but has not been released.

## Spin button

A **spin button** control is a window which allows access to a list of data values, but which occupies less space in the dialog than a list box.



### Styles:

Styles	Definitions
All Numeric ReadOnly	The spin button accepts all characters, only digits 0-9, or no characters at all.
Left Center Right	The data is left justified, centered, or right justified
NoBorder	Does not have a border
Invisible	Is initially invisible
NoGroup	Does not begin a group
NoTabstop	Tab does not move focus to the control

### Class specific properties:

Property	Description
'DATA'	The value current selected item. For character spin buttons, a character vector. For numeric spin buttons, an integer scalar. Specifications of the Data property are ignored.
'DATA LIST'	For character spin buttons, a vector of character vectors.  Not used for numeric spin buttons.
'LIMITS'	For numeric spin buttons, 2 integers: the lower and upper limits of the range to display.  Not used for character spin buttons.
'SELECTION'	Integer scalar - 0 origin index of selected item or -1 if the value typed in by the user is not in the spin button's list of items.
'UNICODE DATA'	See Data

Property	Description
'UNICODE DATA LIST'	See Data List

Common properties:

- 'APPLICATION DATA'
- 'CLASS'
- 'CLIENT POSITION'
- 'CLIENT SIZE'
- 'COLOR BACKGROUND'
- 'COLOR FOREGROUND'
- 'CONTEXT HELP'
- 'CURSOR'
- 'CURSOR POSITION'
- 'EVENTS'
- 'FONT'
- 'ID'
- 'NAME'
- 'OFFSET'
- 'POSITION'
- 'RGB COLOR BACKGROUND'
- 'RGB COLOR FOREGROUND'
- 'SIZE'
- 'STATE APL'
- 'STATE ENABLE'
- 'STATE FOCUS'
- 'STATE VISIBLE'
- 'STYLE'
- 'TOOL TIP'
- 'UNICODE CONTEXT HELP'
- 'UNICODE FONT'
- 'UNICODE TOOL TIP'
- 'USER DATA'

**Events:**

Event	Description
'Changed'	Tells the application that the contents of the spin field changed.
'Context menu'	The user has clicked the right mouse button in the window.
'End spin'	Tells the application that the user released the select button or one of the arrow keys while spinning a button.
'Hover'	The mouse pointer paused over the control
'Kill focus'	Tells the application that the spin field lost focus.
'Set focus'	Tells the application that the spin button gained focus.

The Spin button control class uses a member of the [Microsoft Windows Common Control Library](#).

## Tab

A **Tab** control is analogous to the dividers in a notebook or the labels in a file cabinet. By using a tab control, an application can define multiple pages for the same area of a window. Each page consists of a dialog that the application displays when the user selects the corresponding tab.



### Styles:

Styles	Definitions
MultipleLines	Displays tabs in multiple lines
FixedWidthTabs	All tabs are the same width
Buttons	Tabs are drawn as buttons
FlatButtons	Selected tab appears indented
Invisible	Is initially invisible
NoGroup	Does not begin a group
NoTabstop	Tab does not move focus to the control

### Notes:

- FlatButtons implies Buttons.

### Class specific properties:

Property	Description
'DATA'	Three-column matrix - Page dialog information:  [;1] Dialog handle [;2] Tab text [;3] Picture information  The picture information can be any of 3 types of arrays: a character vector containing the name of an image file, an enclosed character vector containing

Property	Description																		
	<p>the content of an image file, or a 3 element array containing image resource information. Use "" for no picture.</p> <p>Character vector - Name of image file. The following image file types are supported:</p> <table border="1" data-bbox="605 436 1297 779"> <tr> <td>Bitmap</td> <td>BMP</td> </tr> <tr> <td>Enhanced Metafile</td> <td>EMF</td> </tr> <tr> <td>Exchangeable Image File</td> <td>Exif</td> </tr> <tr> <td>Graphics Interchange Format</td> <td>GIF</td> </tr> <tr> <td>Icon</td> <td>ICO</td> </tr> <tr> <td>Joint Photographic Experts Group</td> <td>JPG or JPEG</td> </tr> <tr> <td>Portable Network Graphics</td> <td>PNG</td> </tr> <tr> <td>Tag Image File Format</td> <td>TIF or TIFF</td> </tr> <tr> <td>Windows Metafile</td> <td>WMF</td> </tr> </table> <p><a href="#">GDI+</a> is required for file types other than bitmap and icon.</p> <p>If a full path is not supplied, files are searched for using the following sequence:</p> <ol style="list-style-type: none"> <li>1. The directory from which AP 145 was loaded</li> <li>2. The current directory</li> <li>3. The Windows system directory</li> <li>4. The directories listed in the PATH environment variable</li> </ol> <p>Enclosed character vector - Contents of image file. Icon file images are not supported. <a href="#">GDI+</a> is required for this type of array.</p> <p>Three element array - Image resource</p> <p>[1] Character vector - Resource type: 'BITMAP' or 'ICON'  [2] Character vector – Name of executable containing resource.  For example: 'SHELL32.DLL'  [3] Integer scalar - Resource identifier</p> <p>See <a href="#">Tab Controls - Property Windows</a> for more information.</p>	Bitmap	BMP	Enhanced Metafile	EMF	Exchangeable Image File	Exif	Graphics Interchange Format	GIF	Icon	ICO	Joint Photographic Experts Group	JPG or JPEG	Portable Network Graphics	PNG	Tag Image File Format	TIF or TIFF	Windows Metafile	WMF
Bitmap	BMP																		
Enhanced Metafile	EMF																		
Exchangeable Image File	Exif																		
Graphics Interchange Format	GIF																		
Icon	ICO																		
Joint Photographic Experts Group	JPG or JPEG																		
Portable Network Graphics	PNG																		
Tag Image File Format	TIF or TIFF																		
Windows Metafile	WMF																		
'SELECTION'	Integer scalar - 0 origin index of selected page																		
'UNICODE DATA'	See Data																		



### Common properties:

- 'APPLICATION DATA'
- 'CLASS'
- 'CLIENT POSITION'
- 'CLIENT SIZE'
- 'COLOR BACKGROUND'
- 'COLOR FOREGROUND'
- 'CONTEXT HELP'
- 'CURSOR'
- 'CURSOR POSITION'
- 'EVENTS'
- 'FONT'
- 'ID'
- 'NAME'
- 'OFFSET'
- 'POSITION'
- 'RGB COLOR BACKGROUND'
- 'RGB COLOR FOREGROUND'
- 'SIZE'
- 'STATE ENABLE'
- 'STATE FOCUS'
- 'STATE VISIBLE'
- 'STYLE'
- 'TOOL TIP'
- 'UNICODE CONTEXT HELP'
- 'UNICODE FONT'
- 'UNICODE TOOL TIP'
- 'USER DATA'

### **Events:**

<b>Event</b>	<b>Description</b>
'Context menu'	The user has clicked the right mouse button in the window.
'Hover'	The mouse pointer paused over the control
'Select'	A tab has been selected.

The Tab control class is a member of the [Microsoft Windows Common Control Library](#).

## Text

A **text** control is used to display a text label within a dialog.

### Text

#### Styles:

Styles	Definitions
Left Center Right	The data is left justified, centered, or right justified
Border	Draws a 3 dimensional border around the text
Wordbreak	Wraps data onto multiple lines
Invisible	Is initially invisible
NoGroup	Does not begin a group

#### Class specific properties:

Property	Description
'DATA'	Character vector - Control text
'UNICODE DATA'	See Data

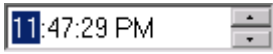
#### [Common properties:](#)

- 'APPLICATION DATA'
- 'CLASS'
- 'CLIENT POSITION'
- 'CLIENT SIZE'
- 'COLOR BACKGROUND'
- 'COLOR FOREGROUND'
- 'CONTEXT HELP'
- 'CURSOR'
- 'CURSOR POSITION'
- 'FONT'
- 'ID'
- 'NAME'
- 'OFFSET'
- 'POSITION'
- 'RGB COLOR BACKGROUND'
- 'RGB COLOR FOREGROUND'
- 'SIZE'
- 'STATE ENABLE'

- 'STATE VISIBLE'
- 'STYLE'
- 'TOOL TIP'
- 'UNICODE CONTEXT HELP'
- 'UNICODE FONT'
- 'UNICODE TOOL TIP'
- 'USER DATA'

## Time

A **Time** control provides an interface through which to exchange time information with a user.



### Styles:

Styles	Definitions
Checkbox	Checkbox included which grays time
Invisible	Is initially invisible
NoGroup	Does not begin a group
NoTabstop	Tab does not move focus to the control

### Class specific properties:

Property	Description
'DATA'	Four integers - Time of data  Hour Minute Second Millisecond
'STATE CHECKED'	Boolean scalar - 0 not checked, 1 checked

### [Common properties:](#)

- 'APPLICATION DATA'
- 'CLASS'
- 'CLIENT POSITION'
- 'CLIENT SIZE'
- 'COLOR BACKGROUND'
- 'COLOR FOREGROUND'
- 'CONTEXT HELP'
- 'CURSOR'
- 'CURSOR POSITION'
- 'EVENTS'
- 'FONT'
- 'ID'
- 'NAME'
- 'OFFSET'
- 'POSITION'

- 'RGB COLOR BACKGROUND'
- 'RGB COLOR FOREGROUND'
- 'SIZE'
- 'STATE ENABLE'
- 'STATE FOCUS'
- 'STATE VISIBLE'
- 'STYLE'
- 'TOOL TIP'
- 'UNICODE CONTEXT HELP'
- 'UNICODE FONT'
- 'UNICODE TOOL TIP'
- 'USER DATA'

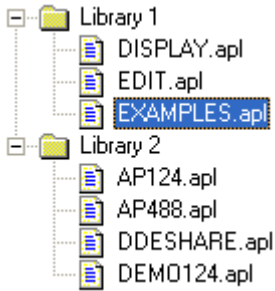
**Events:**

<b>Event</b>	<b>Description</b>
'Change'	The user has changed the time.
'Context menu'	The user has clicked the right mouse button in the window.
'Hover'	The mouse pointer paused over the control
'Kill focus'	The control is losing the focus.
'Set focus'	The control receives the focus.

The Time control class is a member of the [Microsoft Windows Common Control Library](#).

## Treeview

A **Treeview** control is a window that displays a hierarchical list of items, such as the headings in a document, the entries in an index, or the files and directories on a disk. Each item consists of a label and an optional image, and each item can have a list of subitems associated with it. By clicking an item, the user can expand or collapse the associated list of subitems.



### Styles:

Styles	Definitions
Border	The control is drawn with a border
Checkboxes	Displays checkboxes next to the items
Editlabels	Allows the user to edit item labels
Fullrowselect	Enables clicking anywhere on an item's row to select the item
Hasbuttons	Displays plus (+) and minus (-) buttons next to parent items
Haslines	Uses lines to show the hierarchy of items
Lineatroot	Uses lines to link items at the root of the tree-view control. This value is ignored if Haslines is not also specified
Singleexpand	Causes the item being selected to expand and the item being unselected to collapse upon selection in the tree view. If the user holds down the CTRL key while selecting an item, the item being unselected will not be collapsed.
Invisible	Is initially invisible
NoGroup	Does not begin a group
NoTabstop	Tab does not move focus to the control

**Class specific properties:**

Property	Description																		
'DATA'	<p>4 column matrix - Defines data to display in control.</p> <p>[;1] Nonnegative integer scalar – Depth of the item. The first item must be 0.</p> <p>[;2] Character vector – The item label</p> <p>[;3] Integer scalar – Index of the control’s picture list image to display when the item is not selected. Use -1 for no picture.</p> <p>[;4] Integer scalar – Index of the control’s picture list image to display when the item is selected. Use -1 for no picture.</p>																		
'PICTURE LIST'	<p>Array of pictures used in a treeview control. Each element can be any of 4 types of arrays: character vectors containing names of image files, enclosed character vectors containing the contents of image files, 3 element arrays containing image resource information, or integer scalars that are built-in picture codes.</p> <p>Character vector - Name of image file. The following image file types are supported:</p> <table border="1" data-bbox="594 974 1289 1318"> <tbody> <tr> <td>Bitmap</td> <td>BMP</td> </tr> <tr> <td>Enhanced Metafile</td> <td>EMF</td> </tr> <tr> <td>Exchangeable Image File</td> <td>Exif</td> </tr> <tr> <td>Graphics Interchange Format</td> <td>GIF</td> </tr> <tr> <td>Icon</td> <td>ICO</td> </tr> <tr> <td>Joint Photographic Experts Group</td> <td>JPG or JPEG</td> </tr> <tr> <td>Portable Network Graphics</td> <td>PNG</td> </tr> <tr> <td>Tag Image File Format</td> <td>TIF or TIFF</td> </tr> <tr> <td>Windows Metafile</td> <td>WMF</td> </tr> </tbody> </table> <p><a href="#">GDI+</a> is required for file types other than bitmap and icon.</p> <p>If a full path is not supplied, files are searched for using the following sequence:</p> <ol style="list-style-type: none"> <li>1. The directory from which AP 145 was loaded</li> <li>2. The current directory</li> <li>3. The Windows system directory</li> <li>4. The directories listed in the PATH environment variable</li> </ol> <p>Enclosed character vector - Contents of image file. Icon file images are not supported. <a href="#">GDI+</a> is required for this type of array.</p>	Bitmap	BMP	Enhanced Metafile	EMF	Exchangeable Image File	Exif	Graphics Interchange Format	GIF	Icon	ICO	Joint Photographic Experts Group	JPG or JPEG	Portable Network Graphics	PNG	Tag Image File Format	TIF or TIFF	Windows Metafile	WMF
Bitmap	BMP																		
Enhanced Metafile	EMF																		
Exchangeable Image File	Exif																		
Graphics Interchange Format	GIF																		
Icon	ICO																		
Joint Photographic Experts Group	JPG or JPEG																		
Portable Network Graphics	PNG																		
Tag Image File Format	TIF or TIFF																		
Windows Metafile	WMF																		

	<p>Three element array - Image resource</p> <p>[1] Character vector - Resource type: 'BITMAP' or 'ICON'</p> <p>[2] Character vector – Name of executable containing resource. For example: 'SHELL32.DLL'</p> <p>[3] Integer scalar - Resource identifier</p> <p>Integer scalar - Built-in picture code:</p> <ul style="list-style-type: none"> <li>0 Closed folder</li> <li>1 Open folder</li> <li>2 Closed book</li> <li>3 Open book</li> <li>4 Page</li> </ul>
'SELECTION'	Integer scalar- Index of selected item
'STATE CHECKED'	<p>Integer vector - Specifies whether a check box appears next to an item and whether it is checked or unchecked. The state checked property is only used with treeview controls with the CheckBoxes style. The state checked property must have the same number of elements as the number of rows in the treeview's data property. Its values mean:</p> <ul style="list-style-type: none"> <li>0 No check box</li> <li>1 Empty check box</li> <li>2 Checked check box</li> </ul>
'STATE EXPANDED'	Boolean vector – Length equals number of items. 0 collapsed, 1 expanded
'TOP INDEX'	Integer scalar - 0 origin index of item displayed at top of control
'UNICODE DATA'	See Data

Common properties:

- 'APPLICATION DATA'
- 'CLASS'
- 'CLIENT POSITION'
- 'CLIENT SIZE'
- 'COLOR BACKGROUND'
- 'COLOR FOREGROUND'
- 'CONTEXT HELP'
- 'CURSOR'
- 'CURSOR POSITION'
- 'DROPPED FILES'



- 'EVENTS'
- 'FONT'
- 'ID'
- 'NAME'
- 'OFFSET'
- 'POSITION'
- 'RGB COLOR BACKGROUND'
- 'RGB COLOR FOREGROUND'
- 'SIZE'
- 'STATE APL'
- 'STATE ENABLE'
- 'STATE ENABLE DRAGDROP'
- 'STATE FOCUS'
- 'STATE VISIBLE'
- 'STYLE'
- 'TOOL TIP'
- 'UNICODE CONTEXT HELP'
- 'UNICODE FONT'
- 'UNICODE TOOL TIP'
- 'USER DATA'

**Events:**

Event	Description
'Check state change'	The user checked or cleared an item's checkbox
'Context menu'	The user has clicked the right mouse button in the window.
'Enter or double click'	The user has pressed Enter or double clicked.
'Files dropped'	The user has dropped one or more files on the window
'Hover'	The mouse pointer paused over the control
'Label changed'	The user has edited an item's label. Reference the control's DATA property to retrieve the item's modified label.
'Select'	The user has selected an item.

The Treeview control class is a member of the [Microsoft Windows Common Control Library](#).

## Desktop

The desktop supports the following properties. Use a handle of 1 or the `HWND_DESKTOP` constant from `GUIVARS` to refer to the desktop. The `HANDLE_DESKTOP` function returns the actual handle of the desktop window.

Property	Description
'CLIENT SIZE'	Two integers - Size of the desktop's client area in pixels  The desktop's Client Size property can only be referenced.
'CLIPBOARD DATA'	Use the Clipboard Data property to move data between APL2 and the clipboard. Any arbitrary APL2 array can be set on the clipboard and pasted into APL2. Only the following type arrays can be pasted into other applications: <ul style="list-style-type: none"> <li>• Character scalar</li> <li>• Character vector</li> <li>• Character matrix</li> <li>• Vector of character vectors</li> </ul> Use a Tab character to delimit columns for other applications.
'CLIPBOARD LINK'	Vector of 3 character vectors - Identifies DDE server that placed data on the clipboard.  Application name Topic name Item name  The clipboard link property can only be referenced.
'CLIPBOARD TEXT'	Character vector - APL2 does not parse Clipboard Text property values. They may contain Tabs, Carriage Returns, and Line Feeds to delimit data.
'CLIPBOARD UNICODE DATA'	See Clipboard Data
'PATH'	Character vector - Current path. i.e. 'c:\dir'
'POSITION'	The desktop's Position property can only be referenced.
'SIZE'	Two integers - Size of the desktop in pixels  The desktop's Size property can only be referenced.

<b>Property</b>	<b>Description</b>
'STATE ENABLE'	Boolean scalar - 0 disabled, 1 enabled  Specifications of the desktop's State Enable property are ignored.

## Common Property Reference

The following table defines properties used with multiple classes of windows:

Property	Description																								
'APPLICATION DATA'	<p>Arbitrary APL2 array - Use the application data property to store an APL2 array within a window.</p> <p>No more than 500,000 bytes can be stored in a window's application data property.</p> <p>Menus do not support the application data property.</p> <p>DDE DATA, DDE COMMAND, DDE SERVER, DDE TOPIC, DDE ITEM objects do not support the application data property.</p>																								
'CLASS'	<p>Character vector - Class of window.</p> <p>Specifications of the CLASS property are ignored.</p>																								
'CLIENT POSITION'	2 integers - Position of window relative to lower left corner of parent's client area																								
'CLIENT SIZE'	2 integers - Sets size of window to yield specified size client area																								
'COLOR BACKGROUND' 'COLOR FOREGROUND'	<p>Character vector - Any of the following colors:</p> <table> <tbody> <tr> <td>'NEUTRAL'</td> <td>Color not BACKGROUND</td> </tr> <tr> <td>'DEFAULT'</td> <td>Color not BACKGROUND</td> </tr> <tr> <td>'FALSE'</td> <td>All bits are set to 0.</td> </tr> <tr> <td>'TRUE'</td> <td>All bits are set to 1.</td> </tr> <tr> <td>'BLACK'</td> <td>Black</td> </tr> <tr> <td>'BLUE'</td> <td>Blue</td> </tr> <tr> <td>'BROWN'</td> <td>Brown</td> </tr> <tr> <td>'CYAN'</td> <td>Cyan</td> </tr> <tr> <td>'DARKBLUE'</td> <td>Dark blue</td> </tr> <tr> <td>'DARKCYAN'</td> <td>Dark cyan</td> </tr> <tr> <td>'DARKGRAY'</td> <td>Dark gray</td> </tr> <tr> <td>'DARKGREEN'</td> <td>Dark green</td> </tr> </tbody> </table>	'NEUTRAL'	Color not BACKGROUND	'DEFAULT'	Color not BACKGROUND	'FALSE'	All bits are set to 0.	'TRUE'	All bits are set to 1.	'BLACK'	Black	'BLUE'	Blue	'BROWN'	Brown	'CYAN'	Cyan	'DARKBLUE'	Dark blue	'DARKCYAN'	Dark cyan	'DARKGRAY'	Dark gray	'DARKGREEN'	Dark green
'NEUTRAL'	Color not BACKGROUND																								
'DEFAULT'	Color not BACKGROUND																								
'FALSE'	All bits are set to 0.																								
'TRUE'	All bits are set to 1.																								
'BLACK'	Black																								
'BLUE'	Blue																								
'BROWN'	Brown																								
'CYAN'	Cyan																								
'DARKBLUE'	Dark blue																								
'DARKCYAN'	Dark cyan																								
'DARKGRAY'	Dark gray																								
'DARKGREEN'	Dark green																								

Property	Description
	<p>'DARKPINK'    Dark pink</p> <p>'DARKRED'    Dark red</p> <p>'GREEN'    Green</p> <p>'PALEGRAY'    Pale gray</p> <p>'PINK'    Pink</p> <p>'RED'    Red</p> <p>'WHITE'    White</p> <p>'YELLOW'    Yellow</p> <p>A null length vector resets the window's color to its default.</p> <p>Setting a window's named color property resets the corresponding RGB color property to null.</p>
'CONTEXT HELP'	<p>Vector of character vectors - Descriptive text that is displayed when control has focus and user presses F1 or clicks on question mark button and then clicks on control.</p>
'CURSOR'	<p>Specifies the mouse pointer to be used when the pointer is over the window.</p> <p>Character vector - Name of a cursor (.CUR or .ANI) file. If a full path is not supplied, files are searched for using the following sequence:</p> <ol style="list-style-type: none"> <li>1. The directory from which AP 145 was loaded</li> <li>2. The current directory</li> <li>3. The Windows system directory</li> <li>4. The directories listed in the PATH environment variable.</li> </ol> <p>Scalar integer - A built-in cursor code:</p> <ol style="list-style-type: none"> <li>1 Small arrow and hour glass</li> <li>2 Arrow</li> <li>3 Cross hairs</li> <li>4 Text input I-beam</li> <li>5 Prohibited</li> <li>6 Four pointed arrow</li> <li>7 Upper right sizing arrow</li> <li>8 Top sizing arrow</li> <li>9 Upper left sizing arrow</li> <li>10 Left sizing arrow</li> <li>11 Up arrow</li> </ol>

Property	Description						
	12 Hour glass 13 Hand 14 Arrow and question mark						
'CURSOR POSITION'	Two integers - Position of mouse pointer relative to lower left corner of window.						
'DROPPED FILES'	Vector of character vectors - List of names of files user dropped on window						
'EVENTS'	2 column matrix - Event handler array For windows, [;1] Character vector - Event name [;2] Arbitrary array - Event handler For menus, [;1] Integer scalar - Menu item identifier [;2] Arbitrary array - Event handler Specifying a zero element array removes all event handlers.						
'FONT'	Character vector - Font size and name separated by period. Size expressed in points (1/72 inch)						
'HORIZONTAL LIMITS'	2 integers - Range of horizontal scrollbar						
' HORIZONTAL SELECTION'	Integer scalar - Horizontal scrollbar position						
'ID'	Integer scalar - Identifier of control						
'NAME'	Character vector - Name of control						
'OFFSET'	4 integers - Percentage of parent size change added to position of left, top, right, and bottom edges of window						
'PICTURE'	The PICTURE property can be any of 4 types of arrays: character vectors containing names of image files, enclosed character vectors containing the contents of image files, 3 element arrays containing image resource information, or integer scalars that are built-in picture codes. Character vector - Name of image file. The following image file types are supported: <table border="1" data-bbox="690 1759 1385 1877" style="margin-left: auto; margin-right: auto;"> <tbody> <tr> <td>Bitmap</td> <td>BMP</td> </tr> <tr> <td>Enhanced Metafile</td> <td>EMF</td> </tr> <tr> <td>Exchangeable Image File</td> <td>Exif</td> </tr> </tbody> </table>	Bitmap	BMP	Enhanced Metafile	EMF	Exchangeable Image File	Exif
Bitmap	BMP						
Enhanced Metafile	EMF						
Exchangeable Image File	Exif						

Property	Description												
	<table border="1" data-bbox="690 210 1385 436"> <tr> <td>Graphics Interchange Format</td> <td>GIF</td> </tr> <tr> <td>Icon</td> <td>ICO</td> </tr> <tr> <td>Joint Photographic Experts Group</td> <td>JPG or JPEG</td> </tr> <tr> <td>Portable Network Graphics</td> <td>PNG</td> </tr> <tr> <td>Tag Image File Format</td> <td>TIF or TIFF</td> </tr> <tr> <td>Windows Metafile</td> <td>WMF</td> </tr> </table> <p data-bbox="636 474 1385 510"><a href="#">GDI+</a> is required for file types other than bitmap and icon.</p> <p data-bbox="636 548 1528 619">If a full path is not supplied, files are searched for using the following sequence:</p> <ol data-bbox="656 657 1422 800" style="list-style-type: none"> <li>1. The directory from which AP 145 was loaded</li> <li>2. The current directory</li> <li>3. The Windows system directory</li> <li>4. The directories listed in the PATH environment variable</li> </ol> <p data-bbox="636 840 1515 911">Enclosed character vector - Contents of image file. Icon file images are not supported. <a href="#">GDI+</a> is required for this type of array.</p> <p data-bbox="636 951 1115 984">Three element array - Image resource</p> <p data-bbox="636 1024 1377 1058">[1] Character vector - Resource type: 'BITMAP' or 'ICON'</p> <p data-bbox="636 1060 1446 1094">[2] Character vector – Name of executable containing resource.</p> <p data-bbox="680 1096 1083 1129">For example: 'SHELL32.DLL'</p> <p data-bbox="636 1131 1125 1165">[3] Integer scalar - Resource identifier</p> <p data-bbox="636 1205 1107 1239">Integer scalar - Built-in picture code:</p> <ol data-bbox="777 1241 1024 1860" style="list-style-type: none"> <li>1 Application</li> <li>2 Information</li> <li>3 Warning</li> <li>4 Error</li> <li>5 Question</li> <li>6 Windows Logo</li> <li>7 Cut</li> <li>8 Copy</li> <li>9 Paste</li> <li>10 Undo</li> <li>11 Redo</li> <li>12 Delete</li> <li>13 New</li> <li>14 Open</li> <li>15 Save</li> <li>16 Print Preview</li> <li>17 Properties</li> </ol>	Graphics Interchange Format	GIF	Icon	ICO	Joint Photographic Experts Group	JPG or JPEG	Portable Network Graphics	PNG	Tag Image File Format	TIF or TIFF	Windows Metafile	WMF
Graphics Interchange Format	GIF												
Icon	ICO												
Joint Photographic Experts Group	JPG or JPEG												
Portable Network Graphics	PNG												
Tag Image File Format	TIF or TIFF												
Windows Metafile	WMF												

Property	Description
	<ul style="list-style-type: none"> <li>18 Help</li> <li>19 Find</li> <li>20 Replace</li> <li>21 Print</li> <li>22 Large Icons View</li> <li>23 Small Icons View</li> <li>24 List View</li> <li>25 Details view</li> <li>26 Sort by name</li> <li>27 Sort by size</li> <li>28 Sort by date</li> <li>29 Sort by type</li> <li>30 Parent folder</li> <li>31 Net connect</li> <li>32 Net disconnect</li> <li>33 New folder</li> <li>34 Backward</li> <li>35 Forward</li> <li>36 Favorites</li> <li>37 Add to favorites</li> <li>38 View tree</li> </ul>
'POSITION'	2 integers - Position of window relative to lower left hand corner of parent
'RGB COLOR BACKGROUND' 'RGB COLOR FOREGROUND'	<p>3 integers from 0 to 255 - Red, Green, and Blue intensity. Setting a null vector resets the window's color to its default.</p> <p>Setting a window's RGB color property resets the corresponding named color property.</p>
'SIZE'	2 integers - Width and height of window in pixels
'STATE APL'	Boolean scalar - 0 normal keyboard, 1 APL keyboard enabled
'STATE CHECKED'	Boolean scalar - 0 unchecked, 1 checked
'STATE ENABLE'	Boolean scalar - 0 disabled, 1 enabled
'STATE ENABLE DRAGDROP'	Boolean scalar - 0 disabled, 1 enabled
'STATE FOCUS'	Boolean scalar - 0 not focus, 1 has focus
'STATE READONLY'	Boolean scalar - 0 input enabled, 1 Read only
'STATE VISIBLE'	Boolean scalar - 0 invisible, 1 visible
'STYLE'	Character vector - Styles used to create window.



<b>Property</b>	<b>Description</b>
	Specifications of the STYLE property are ignored.
'TOOL TIP'	Character vector - Text that is displayed when mouse pointer pauses over control.
'UNICODE CONTEXT HELP'	See Context help
'UNICODE FONT'	See Font
'UNICODE TOOL TIP'	See Tool tip
'USER DATA'	Integer scalar - Arbitrary value
'VERTICAL LIMITS'	2 integers - Range of vertical scrollbar
' VERTICAL SELECTION'	Integer scalar - Horizontal vertical position

## ***Object Reference***

The GUITOOLS workspace function CREATEOBJ is used to create objects.

AP 145 supports the following object classes:

- ['DDE DATA'](#)
- ['DDE COMMAND'](#)
- ['DDE SERVER'](#)
- ['DDE TOPIC'](#)
- ['DDE ITEM'](#)
- ['TIMER'](#)

The following sections list the initialization data, properties, and events supported for each object class. Property and event names are case insensitive.

## DDE DATA

**DDE DATA** objects are used by DDE client applications to retrieve and set values of items in DDE server applications.

**Class name:** DDE DATA

**Syntax:** HDATA←CREATEOBJ 'DDE DATA' application topic item

**Initialization data:**

Data	Description
application	Character vector - Name of a running DDE server application
topic	Character vector - Name of a topic supported by the server
item	Character vector - Name of an item within the topic

**Properties:**

Property	Description
'DATA'	When connected to an APL2 server - Any arbitrary APL2 array When connected to non-APL servers - Line feed delimited character vectors.
'EVENTS'	See the events table
'XLTABLE DATA'	Used when connected to Excel - Matrix of character scalars, character vectors, numeric scalars, and 1 element integer vectors. 1 element integer vectors are Excel error codes.

**Events:**

Event	Description
'New value'	A new value is available at the server.
'Server close'	The server has closed.

The following Excel error code variables are available in the DDESHARE workspace:

```
XLERROR_DIVZERO←7
XLERROR_NA      ←42
XLERROR_NAME    ←29
XLERROR_NULL    ←0
XLERROR_NUM     ←36
XLERROR_REF     ←23
XLERROR_VALUE   ←15
```

## DDE COMMAND

**DDE COMMAND** objects are used by DDE client applications to send commands to DDE server applications.

**Class name:** DDE COMMAND

**Syntax:** HCMD←CREATEOBJ 'DDE COMMAND' application topic

**Initialization data:**

Data	Description
application	Character vector - Name of a running DDE server application
topic	Character vector - Name of a topic supported by the server

**Properties:**

Property	Description
'DATA'	Specification: Character vector - Command to execute Reference: Integer scalar - Return code from last command
'EVENTS'	See the events table

**Events:**

Event	Description
'Server close'	The server has closed.

## DDE SERVER

**DDE SERVER** objects are used to manage topics in DDE server applications.

**Class name:** DDE SERVER

**Syntax:** HSERVER←CREATEOBJ 'DDE SERVER' application

**Initialization data:**

<b>Data</b>	<b>Description</b>
application	Character vector - The name by which the server should be known

**Properties:** None

**Events:** None

## DDE TOPIC

**DDE TOPIC** objects are used in DDE server applications to handle requests from DDE clients to execute commands and replace values of DDE ITEM objects.

**Class name:** DDE TOPIC

**Syntax:** HTOPIC←CREATEOBJ 'DDE TOPIC' hserver topic

**Initialization data:**

Data	Description
hserver	Integer scalar - Handle of a DDE SERVER object
topic	Character vector - Name of a topic to be supported by the server

**Properties:**

Property	Description
'DDE EXECUTE'	Reference: Character vector - Command to be executed Specification: Integer scalar - Return code from executing command
'EVENTS'	See the events table
'DDE WRITE'	Reference: 2 element array - Data sent by client  [1] Character vector - Name of DDE ITEM object to replace [2] Arbitrary array - Value with which to replace DDE ITEM object's value  Specification: 2 element integer vector - Return code  [1] Boolean: 0, Success 1, Command not executed [2] Integer 0-255: Return code if [1] is 0, ignored otherwise

**Events:**

Event	Description
'Execute value'	Client has sent command to be executed
'Write value'	Client has sent array to replace DDE ITEM object's value

## DDE ITEM

**DDE ITEM** objects are used in DDE server applications to specify named values the servers should support.

**Class name:** DDE ITEM

**Syntax:** HITEM←CREATEOBJ 'DDE ITEM' hserver htopic item

**Initialization data:**

Data	Description
hserver	Integer scalar - Handle of a DDE SERVER object
htopic	Integer scalar - Handle of a DDE TOPIC object
item	Character vector - Name of an item to be supported by the server

**Properties:**

Property	Description
'DATA'	Arbitrary array - Value of item
'EVENTS'	See the events table
'STATE LINK'	Boolean scalar - 1 indicates AP 145 should copy new values to the clipboard and set the Link clipboard format data. 0 indicates new values are not copied.

**Events:** None

## TIMER

**TIMER** objects signal events at a set interval.

**Class name:** TIMER

**Syntax:** HTIMER←CREATEOBJ 'TIMER' interval expression

**Initialization data:**

Data	Description
interval	Integer scalar - The number of seconds the timer should pause between ticks. This value must not be less than zero.
expression	Arbitrary array – Event handler

**Properties:**

Property	Description
'DATA'	Arbitrary character vector - Use the data property to store a character vector within a timer object.
'EVENTS'	2 column matrix - Event handler array [;1] Character vector - Event name [;2] Character vector - Event handler expression
'INTERVAL'	Integer scalar - The number of seconds the timer should pause between signaling events. This value must not be less than zero
'UNICODE DATA'	See Data.

[Common properties:](#)

- 'APPLICATION DATA'
- 'USER DATA'

**Events:**

Event	Description
'Timer'	Client has sent command to be executed



## ***GUITOOLS Function Reference***

The GUITOOLS workspace contains the following groups of tools.

<a href="#"><u>GPDLGPROCESS</u></a>	Dialog Processing Tools
<a href="#"><u>GPUTILITY</u></a>	Utilities
<a href="#"><u>GPPRINT</u></a>	Printing Tools and Constants

The following pages describe the tools in these groups.

## GPDLGPROCESS - Dialog Processing Tools

The dialog processing tools support processing dialogs in applications. For further information, consult the HOW\_DLGPROCESS variable in the GUITOOLS workspace.

<a href="#">ALIGN</a>	Align two or more windows
<a href="#">BITWISE</a>	Apply a function between the bit representations of integers
<a href="#">CALLAPI</a>	Call an API
<a href="#">CALLCOM</a>	Call Component Object Model interface
<a href="#">CENTER_CHILD</a>	Center a window within its parent
<a href="#">CENTER_WINDOW</a>	Center one window within another
<a href="#">CHECK_EVENTS</a>	Process a pending event
<a href="#">COLORDLG</a>	Display a Colors dialog
<a href="#">CONTEXTHELP</a>	Show contextual help
<a href="#">CREATECTL</a>	Create a control
<a href="#">CREATEDLG</a>	Create a dialog
<a href="#">CREATEMENU</a>	Create a menu
<a href="#">CREATEOBJ</a>	Create an object
<a href="#">DEFAULTPROC</a>	Default message process
<a href="#">DESTROYDLG</a>	Destroy a window
<a href="#">DESTROYOBJ</a>	Destroy an object
<a href="#">EXECUTEDLG</a>	Wait for events and execute event handlers
<a href="#">EXECUTEDLGX</a>	Wait for events from multiple processors
<a href="#">FILEDLG</a>	Display an Open or Save As file dialog
<a href="#">FOLDERDLG</a>	Display a Browse for Folder dialog
<a href="#">FONTDLG</a>	Display a Font dialog
<a href="#">FREEAPI</a>	Free an API
<a href="#">GETCHILDREN</a>	Get the handles of a window's children
<a href="#">GETPARENT</a>	Get the handle of a window's parent
<a href="#">GET_CELLSIZE</a>	Get the size of a grid control cell
<a href="#">GET_PROFILE</a>	Get and list profile values from the registry or an INI file
<a href="#">GET_PROPERTY</a>	Get window properties
<a href="#">GET_RANGE</a>	Get the value of a range property
<a href="#">GUIRETRACT</a>	Retract SV145
<a href="#">GUISHARE</a>	Share SV145
<a href="#">HANDLE_DESKTOP</a>	Return the handle of the desktop window
<a href="#">IDFROMWINDOW</a>	Get a window's identifier
<a href="#">ISWINDOW</a>	Query whether a number is a valid window handle
<a href="#">LOADAPI</a>	Load an API
<a href="#">MOVEWINDOW</a>	Move a window
<a href="#">MSGBOX</a>	Display a message box
<a href="#">POPUPMENU</a>	Display a popup a menu
<a href="#">POSTMSG</a>	Post a message
<a href="#">RESIZE</a>	Resize windows
<a href="#">SENDMSG</a>	Send a message
<a href="#">SET_CELLSIZE</a>	Set the size of a grid control cell
<a href="#">SET_PROFILE</a>	Set and delete profile values in the registry or an INI file

<a href="#"><u>SET_PROPERTY</u></a>	Set window properties
<a href="#"><u>SET_RANGE</u></a>	Set the value of a range property
<a href="#"><u>SHAREWINDOW</u></a>	Share a variable with a window property
<a href="#"><u>SHOW</u></a>	Show or hide a window
<a href="#"><u>SIZETOTEXT</u></a>	Resize a window to fit its text
<a href="#"><u>SPACE</u></a>	Space controls equally
<a href="#"><u>STARTWAIT</u></a>	Make AP 145 start saving events until processed
<a href="#"><u>UNICREATEDLG</u></a>	Create a Unicode dialog
<a href="#"><u>UNICREATEMENU</u></a>	Create a menu using Unicode data
<a href="#"><u>UNIFILEDLG</u></a>	Display a Unicode Open or Save As file dialog
<a href="#"><u>UNIFOLDERDLG</u></a>	Display a Unicode Browse for Folder dialog
<a href="#"><u>UNIMSGBOX</u></a>	Display a Unicode message box
<a href="#"><u>UNIPOPUPMENU</u></a>	Display a Unicode popup a menu
<a href="#"><u>WAIT_EVENT</u></a>	Wait for an event
<a href="#"><u>WINDOWFROMID</u></a>	Retrieve the handle of a child from a parent handle and an identifier

## **ALIGN**

Purpose:	Align two or more windows
Syntax:	boolean ALIGN handles
Arguments:	boolean 4 element Boolean vector - Edges to align: left, top, right, and bottom. 1 to align. handles Integer vector - Handles of windows to be aligned
Result:	None

## **BITWISE**

Purpose:	Apply a function between the bit representations of integers
Syntax:	result←la lo BITWISE ra
Arguments:	la Integer array ra Integer array lo Function to apply between the integer arrays
Result:	result Result of function application

## **CALLAPI**

Purpose:	Call an API
Syntax:	result←CALLAPI api parms
Arguments:	api Vector - The vector's first element identifies the API to be called. It is either: <ul style="list-style-type: none"> <li>○ A character vector containing the name of the API</li> <li>○ An integer scalar containing the address of the API</li> </ul> The vector's subsequent elements are the API's parameters.
	parms Vector of integer scalars and character vectors API parameters
Result:	result 2 element vector [1] API return code [2] Returned API name and parameters (elements may be updated)

## CALLCOM

**Purpose:** Call the APL2 Component Object Model interface  
**Syntax:** result←[control] CALLCOM 'COMMAND' [arguments]  
**Arguments:** control Boolean scalar - Default 0  
0: Check event type and return result  
1: Return event type, error message, and result

The CALLCOM right arguments are identical to the COM external function's right argument except that the WAIT command is not supported. Use WAIT\_EVENT to wait for both AP 145 and COM events. Use the 'COM OBJECT' property to retrieve the COM object handles of ActiveX controls.

**Result:** result COM result or event type, error message and COM result

## CENTER\_CHILD

**Purpose:** Center a window within its parent  
**Syntax:** [index] CENTER\_CHILD handle [id]  
**Arguments:** index Boolean scalar or vector - Centering option:  
0 Center horizontally  
1 Center vertically  
0 1 Center both horizontally and vertically (default)  
handle Integer scalar - Handle of window to center within its parent  
Id Integer scalar - Control identifier  
or Character vector - Control name  
**Result:** None

## CENTER\_WINDOW

**Purpose:** Center one window within another  
**Syntax:** [outer] CENTER\_WINDOW inner  
**Arguments:** outer Integer scalar - Handle of window to be centered  
inner Integer scalar - Handle of window within which to center inner  
**Result:** None

## CHECK\_EVENTS

**Purpose:** Process a pending event  
**Syntax:** tozero←CHECK\_EVENTS  
**Arguments:** None  
**Result:** tozero Boolean scalar - 1 if event handler branched to 0. 0 otherwise.

## COLORDLG

**Purpose:** Display a Colors dialog  
**Syntax:** COLORDLG handle [handle...]

Arguments: handle Integer scalars - Handles of window whose colors are to be set.  
Result: None

## CONTEXTHELP

Purpose: Show contextual help  
Syntax: CONTEXTHELP handle [id]  
Arguments: handle Integer scalar – Control, dialog, or menu handle  
Id Integer scalar – Control identifier or menu item identifier  
Or Character vector – Control name  
Result: None

## CREATECTL

Purpose: Create a control  
Syntax: handle←CREATECTL parent class style [id [ctldata]]  
Arguments: parent Integer scalar - Handle of controls parent  
class Character vector - Class of control to create  
style Character vector - List of control styles  
id Integer scalar - Control identifier. Defaults to 0  
or Character vector – Control name. Defaults to null.  
ctldata Integer vector - Control creation data. Use depends on class  
Result: handle Integer scalar - Handle of control

If parent is a Unicode window, CREATECTL creates a Unicode control.

See [Class Reference](#) for supported classes, styles, and control data requirements

## CREATEDLG

Purpose: Create a dialog  
Syntax: handle←[owner parent] CREATEDLG template  
Arguments: owner Integer scalar - Handle of owner window  
parent Integer scalar - Handle of parent window  
template Character vector - Contains a dialog template or a list of dialog styles.  
See [Class Reference](#) for a list of supported styles.  
Result: handle Integer scalar - Handle to the dialog window. Zero if an error occurs.

## CREATEMENU

Purpose: Create a menu bar using Multibyte data  
Syntax: menu←CREATEMENU dialog array  
Arguments: dialog Integer scalar - Handle of dialog  
array Vector of menu items  
Result: menu Integer scalar - Handle of menu

Menu items have 2, 3 or 4 elements:

- [1] Integer scalar - A unique value that is used to identify the choice in the menu's EVENTS property.
- [2] Array - Menu item data
- [3] Integer scalar - Zero or BITWISE combination of [Menu Style Flags](#). MIS\_BITMAP and MIS\_TEXT are ignored.
- [4] Integer scalar - Zero or BITWISE combination of [Menu Attribute Constants](#).

The menu item data may be a character vector or a character vector and a picture array. The character array is the menu item text. The picture array may be:

- An integer scalar that is the handle of a previously loaded bitmap or one of the following built-in picture values:

- 1 Cut
- 2 Copy
- 3 Paste
- 4 Undo
- 5 Redo
- 6 Delete
- 7 New
- 8 Open
- 9 Save
- 10 Print Preview
- 11 Properties
- 12 Help
- 13 Find
- 14 Replace
- 15 Print
- 16 Large Icons View

- A character vector containing a BMP, EMF, Exif, GIF, ICO, JPEG, PNG, TIFF, or WMF file name
- An enclosed character vector containing the contents of an image file
- A 3 element array:
  - [1] 'ICON' or 'BITMAP'
  - [2] Character vector containing name of executable
  - [3] Integer resource identifier

[GDI+](#) is required for file types other than bitmap and icon.

A menu array can also contain nested vectors of menu items. A nested vector of menu items defines a pull-down menu. The first element of a nested vector of menu items defines the pull-down menu item. The subsequent items define the pull-down menu's items.

See [Adding a Menu Bar](#) for more information about using CREATEMENU.

## CREATEOBJ

Purpose: Create an object

Syntax: object←CREATEOBJ class [classdata...]

Arguments: class Character vector - Class name  
          classdata Class specific initialization data. See [Object Reference](#).  
Result: object Integer scalar - Handle of object

## DEFAULTPROC

Purpose: Default message process  
Syntax: result←DEFAULTPROC handle msg mp1 mp2  
Arguments: handle Integer scalar - Handle of message recipient  
          msg Integer scalar - Message identifier  
          mp1 Integer scalar - Message parameter 1  
          mp2 Integer scalar - Message parameter 2  
Result: result Integer scalar - Result returned from sending message

Note: This function is only used as an operand for the EXECUTEDLG operator which supplies the arguments.

## DESTROYDLG

Purpose: Destroy a window  
Syntax: DESTROYDLG handle  
Arguments: handle Integer scalar - Handle of window to be destroyed  
Result: None

## DESTROYOBJ

Purpose: Destroy an object  
Syntax: DESTROYOBJ handle  
Arguments: handle Integer scalar - Handle of object to be destroyed  
Result: None

## EXECUTEDLG

Purpose: Wait for events and execute event handlers  
Syntax: result←[owner] (msgproc EXECUTEDLG) handle  
Arguments: owner Integer scalar - Handle of owner window to be disabled  
          msgproc Function to handle unregistered messages. DEFAULTPROC or 0.  
          handle Integer scalar - Handle of window to activate  
Result: result Arbitrary array - Last value of RESULT set by event handlers

## EXECUTEDLGX

Purpose: Wait for events from multiple processors  
Syntax: result←[owner] (svproc EXECUTEDLGX svars)handle  
Arguments: owner Integer scalar - Handle of owner window to be disabled  
          svproc Function to process shared variable events  
          svars Character matrix - List of shared variables to monitor  
          handle Integer scalar - Handle of window to activate  
Result: result Arbitrary array - Last value of RESULT set by event handlers

## FILEDLG, FILEDLGM

**Purpose:** Display an Open or Save As file dialog  
FILEDLG displays a single-selection dialog.  
FILEDLGM displays a multiple-selection dialog.

**Syntax:** filename←[typelist typeindex [title]] FILEDLG style owner filter

**Arguments:** typelist 2 column matrix - Type names and patterns  
                  [;1] - Type names i.e. 'All Files'.  
                  [;2] - Type pattern, i.e. '\*.\*'.  
typeindex Integer scalar - Index origin 0 index of type name to initially display  
style Boolean scalar - Dialog style. 0 for Open, 1 for Save As  
owner Integer scalar - Handle of owner window  
filter Character vector - File filter. For example, 'c:\temp\\*.txt'.  
title Character vector - Dialog box title

**Result:** filename Character vector - Selected filename, if single file selected.  
                  Vector of character vectors – Selected filenames, if multiple files selected.  
                  ' ' if Open or Save not pressed.

## FOLDERDLG

**Purpose:** Display a Browse for Folder dialog

**Syntax:** folder←FOLDERDLG owner path

**Arguments:** owner Integer scalar - Handle of owner window  
path Character vector – Initial path. ' ' for current directory

**Result:** folder Character vector - Selected folder. ' ' if Ok not pressed.

## FONTDLG

**Purpose:** Display a Font dialog

**Syntax:** FONTDLG handle

**Arguments:** handle Integer scalar - Handle of window whose font is to be set.

**Result:** None

## FREEAPI

**Purpose:** Free one or more APIs loaded by LOADAPI

**Syntax:** FREEAPI api

**Arguments:** api Character vector or vector of character vectors - Names of API to free

**Result:** None

## GETCHILDREN

**Purpose:** Get the handles of a window's children

**Syntax:** children←GETCHILDREN parent

**Arguments:** parent Integer scalar - Handle of parent window

**Result:** children Integer vector - Handles of parent's child windows, 10 is none.



## GETPARENT

Purpose: Get the handle of a window's parent  
Syntax: `parent←GETCHILDREN child`  
Arguments: `child` Integer scalar - Handle of child window  
Result: `parent` Integer scalar - Handle of parent window

## GET\_CELLSIZE

Purpose: Get the size of a grid control cell  
Syntax: `size←GET_CELLSIZE handle index [id]`  
Arguments: `handle` Integer scalar - Handle of grid control (or grid's parent if `id` is supplied)  
`index` 2 element integer vector - Row and column index of cell  
`id` Integer scalar - Identifier of child grid control  
or Character vector - Name of child grid control  
Result: `size` 2 element integer vector - Number of rows and columns the cell covers

## GET\_PROFILE

Purpose: Get a profile value from the registry  
Syntax: `value←GET_PROFILE com app ver fld key val`  
Arguments: `com` Character vector - Company name  
`app` Character vector - Application name  
`ver` Character vector - Version  
`fld` Character vector - Folder, ' ' if none  
`key` Character vector - Key  
`val` Character vector - Default value  
Result: `value` Profile value

Purpose: Get a profile value from an INI file  
Syntax: `value←GET_PROFILE fil app key val`  
Arguments: `com` Character vector - INI file name  
`app` Character vector - Application name  
`key` Character vector - Key  
`val` Character vector - Default value  
Result: `value` Profile value

To query the applications, versions, folders, or keys within the registry, or the applications or keys within an INI file, set the application name, version, folder, or key name to ' '.

## GET\_PROPERTY

Purpose: Get the value of a window property  
Syntax: `valuearray←[propertyarray] GET_PROPERTY handlearray [idarray]`  
Arguments: `propertyarray` Character vector or vector of character vectors - Property names  
Defaults to 'DATA'.  
`handlearray` Integer scalar or vector - Window handles  
(or parents if `idarray` is supplied)

	<code>idarray</code>	Integer scalar or vector - Identifiers of child windows Character vector or vector of character vectors - Names of child windows
Result:	<code>valuearray</code>	Property value(s)

### Conformability of window handles and child window identifier arrays

If either the window handles array or the child window identifiers array has multiple elements and the other array has a single element, the single element is combined with each of the multiple elements. The resulting pairs identify target windows.

If both the window handles array and the child window identifiers array have more than one element, then both must have the same number of elements. The corresponding elements identify target windows.

### Conformability of left and right arguments

If the left argument specifies a single property name, and the right argument identifies a single target window, then `GET_PROPERTY` returns a single property value.

For example, these are valid expressions:

```
VALUE← 'PROPERTY' GET_PROPERTY HANDLE
VALUE← 'PROPERTY' GET_PROPERTY HANDLE ID
```

If the left argument specifies a single property name, and the right argument identifies multiple target windows, then `GET_PROPERTY` returns a vector of property values.

For example, these are valid expressions:

```
(VALUE1 VALUE2)← 'PROPERTY' GET_PROPERTY HANDLE1 HANDLE2
(VALUE1 VALUE2)← 'PROPERTY' GET_PROPERTY HANDLE (ID1 ID2)
(VALUE1 VALUE2)← 'PROPERTY' GET_PROPERTY (HANDLE1 HANDLE2) (ID1 ID2)
```

If the left argument specifies multiple property names, and the right argument identifies a single target window, then `GET_PROPERTY` returns a vector of property values.

For example, this is a valid expression:

```
(VALUE1 VALUE2)← 'PROPERTY1' 'PROPERTY2' GET_PROPERTY HANDLE
```

If the left argument specifies multiple property names, and the right argument identifies multiple target windows, then `GET_PROPERTY` returns a vector containing one element for each property name. Each vector contains the property's values for each target window.

For example, these are valid expressions:

```
RESULT← 'PROPERTY1' 'PROPERTY2' GET_PROPERTY HANDLE (ID1 ID2)
(PROPERTY1_VALUES PROPERTY2_VALUES)←RESULT
```

(PROPERTY1\_VALUE1 PROPERTY1\_VALUE2) ←PROPERTY1\_VALUES  
(PROPERTY2\_VALUE1 PROPERTY2\_VALUE2) ←PROPERTY2\_VALUES

Note: See [Class Reference](#) for supported properties and values

## GET\_RANGE

Purpose: Get the value of a range property  
Syntax: value←[property] GET\_RANGE handle range [id]  
Arguments: property Character vector - Range property name. Defaults to 'CELL DATA'.  
handle Integer scalar - Window handle (or parent if id is supplied)  
range 4 element integer vector - Cell range  
[1] - Row index of first cell in range  
[2] - Column index of first cell in range  
[3] - Number of rows in range  
[4] - Number of columns in range  
id Integer scalar - Identifier of child window  
or Character vector - Name of child window  
Result: value Matrix of range property values

Note: See [Grid](#) for supported range properties and values

## GUIRETRACT

Purpose: Retract SV145  
Syntax: GUIRETRACT  
Arguments: None  
Result: None

## GUISHARE

Purpose: Share SV145  
Syntax: GUISHARE  
Arguments: None  
Result: None

## HANDLE\_DESKTOP

Purpose: Return the handle of the desktop window  
Syntax: handle←HANDLE\_DESKTOP  
Arguments: None  
Result: handle Integer scalar - Handle of the desktop

## IDFROMWINDOW

Purpose: Get a window's identifier  
Syntax: id←IDFROMWINDOW handle  
Arguments: handle Integer scalar - Window handle

Result: `id` Integer scalar - Identifier of window

## ISWINDOW

Purpose: Query whether an integer is a valid window handle  
Syntax: `bool ← ISWINDOW handle`  
Arguments: `handle` Integer scalar  
Result: `bool` Boolean scalar - 1 if window handle, 0 otherwise

## LOADAPI

Purpose: Load one or more APIs from a dynamic link library (DLL)  
Syntax: `LOADAPI library api [descrip [result]]`  
Arguments: `library` Library - Identifies the location of the API. It can be either:

- Character vector - Filename of the DLL containing the APIs
- Integer scalar - Address of the API

`api` Character vector or vector of character vectors - Names of API to load from the dynamic link library. One name per character vector. The names are case sensitive.

If the `api` argument is a character vector containing a single API name, then `descrip` and `result` arguments may be supplied.

If the `api` argument contains more than one API name or if the `descrip` argument is omitted, then AP 145 will pass all the APIs' parameters by value and as 4 byte signed integers or single byte character strings.

`descrip` Integer vector – Description of single API's parameters.

The length of the description corresponds to the number of parameters required by the API. Each item corresponds to a parameter and describes that parameter. Descriptions may not contain more than 32 items. Pass a null vector to indicate that the service has no parameters.

The absolute values of the items are interpreted as follows:

- 0 - Automatic type handling. Parameter is passed by value
- 1 - Automatic type handling. Parameter is passed by reference
- 2 - Parameter is 1 byte signed integer
- 3 - Parameter is 2 byte signed integer
- 4 - Parameter is 4 byte signed integer
- 5 - Parameter is 4 byte floating point number
- 6 - Parameter is 8 byte floating point number
- 7 - Parameter is string of single byte characters
- 8 - Parameter is string of 2 byte Unicode characters

For types 2 through 8, a positive number indicates the parameter should be passed by value and a negative number indicates it should be passed by reference.

`result` Integer scalar – Description of single API's result.

The value of `result` is interpreted as follows:

- 2 - Result is 1 byte signed integer
- 3 - Result is 2 byte signed integer
- 4 - Result is 4 byte signed integer
- 5 - Result is 4 byte floating point number
- 6 - Result is 8 byte floating point number
- 7 - Result is string of single byte characters
- 8 - Result is string of 2 byte Unicode characters

Result: None

## MOVEWINDOW

Purpose: Move a window

Syntax: `handle MOVEWINDOW x y`

Arguments: `handle` Integer scalar - Handle of window to move  
`x` Integer scalar - Distance in pixels from the left edge of the parent  
`y` Integer scalar - Distance in pixels from the bottom edge of the parent

Result: None

## MSGBOX

Purpose: Display a message in a box

Syntax: `result←[owner] MSGBOX text`  
`result←[owner] MSGBOX title text [style]`

Arguments: `owner` Integer scalar - Handle of message box owner  
`title` Character vector - Message box title. Default: 'APL2 Message'  
`text` Character vector - Line feed delimited message text  
`style` Integer scalar - BITWISE combination of [Message Box Style Flags](#)  
Default: MB\_OKCANCEL MB\_ICONEXCLAMATION MB\_MOVEABLE

Result: `result` Integer scalar - See [Message Box Result Codes](#) for the valid codes.

## POPUPMENU

Purpose: Prompt the user with a popup menu

Syntax: `index←POPUPMENU handle x y items`

Arguments: `handle` Integer scalar - Handle of the window relative to which the menu is positioned  
`x` Integer scalar - Horizontal distance from window origin  
`y` Integer scalar - Vertical distance from window origin  
`items` Array of popup menu items. Each menu item is a character vector or a vector of character vectors. A character vector defines one menu item. A vector of character vectors defines a submenu. The first character vector defines the submenu title. Subsequent character vectors define submenu items. Use a null character vector, ' ', to display a separator.

Result: Index origin zero index of the user's selection. If the user dismisses the menu without making a selection, -1 is returned. Separators and submenu titles do not affect selection indices.

## POSTMSG

**Purpose:** Post a message  
**Syntax:** POSTMSG handle msg mp1 mp2  
**Arguments:** handle Integer scalar - Handle of message recipient  
msg Integer scalar - Message identifier  
mp1 Integer scalar - Message parameter 1  
mp2 Integer scalar - Message parameter 2  
**Result:** None

## RESIZE

**Purpose:** Resize windows  
**Syntax:** direction RESIZE handles  
**Arguments:** direction Boolean scalar - 0 resize horizontally, 1 resize vertically  
handles Integer vector - Handles of windows to resize  
**Result:** None

## SENDMSG

**Purpose:** Send a message  
**Syntax:** result←SENDMSG handle msg mp1 mp2  
**Arguments:** handle Integer scalar - Handle of message recipient  
msg Integer scalar - Message identifier  
mp1 Integer scalar - Message parameter 1  
mp2 Integer scalar - Message parameter 2  
**Result:** result 5 element integer array  
[1] - Message result  
[2] - handle  
[3] - msg  
[4] - mp1  
[5] - mp2

## SET\_CELL\_SIZE

**Purpose:** Set the size of a grid control cell  
**Syntax:** SET\_CELL\_SIZE handle index size [id]  
**Arguments:** handle Integer scalar - Handle of grid control (or grid's parent if id is supplied)  
index 2 element integer vector - Row and column index of cell  
size 2 element integer vector - Number of rows and columns the cell covers  
id Integer scalar - Identifier of child grid control  
or Character vector - Name of child control  
**Result:** None

## SET\_PROFILE

**Purpose:** Set a profile value in the registry

Syntax: SET\_PROFILE com app ver fld key val  
Arguments: com Character vector - Company name  
app Character vector - Application name  
ver Character vector - Version  
fld Character vector - Folder, ' ' if none  
key Character vector - Key  
val Character vector - Profile value  
Result: None

Purpose: Set a profile value in an INI file  
Syntax: SET\_PROFILE fil app key val  
Arguments: fil Character vector - INI file name  
app Character vector - Application name  
key Character vector - Key  
val Character vector - Profile value  
Result: None

To delete companies, applications, versions, folders, or keys within the registry, or applications or keys within an INI file, set the contained application, version, folder, key name, or value to 10. Arguments after a null application name, version, folder, or key name are ignored. AP 145 prompts the user for delete confirmation.

## SET\_PROPERTY

Purpose: Set the value of a window property  
Syntax: [propertyarray] SET\_PROPERTY handlearray valuearray [idarray]  
Arguments: propertyarray Character vector or vector of character vectors - Property names.  
Defaults to 'DATA'.  
handlearray Integer scalar or vector - Window handles  
(or parents if idarray is supplied)  
valuearray Property value(s)  
idarray Integer scalar or vector - Identifiers of child windows  
Character vector or vector of character vectors - Names of child windows  
Result: None

### Conformability of window handles and child window identifier arrays

If either the window handles array or the child window identifiers array has multiple elements and the other array has a single element, the single element is combined with each of the multiple elements. The resulting pairs identify target windows.

If both the window handles array and the child window identifiers array have more than one element, then both must have the same number of elements. The corresponding elements identify target windows.

### Conformability of left and right arguments

If the left argument specifies a single property name, and the right argument identifies a single target window, then the property is set with the values array.

For example, these are valid expressions:

```
'PROPERTY' SET_PROPERTY HANDLE VALUE
'PROPERTY' SET_PROPERTY HANDLE VALUE ID
```

If the left argument specifies a single property name, and the right argument identifies multiple target windows, then the values array must contain either one item or the same number as elements as target windows.

For example, these are valid expressions:

```
'PROPERTY' SET_PROPERTY (HANDLE1 HANDLE2) (VALUE1 VALUE2)
'PROPERTY' SET_PROPERTY HANDLE (VALUE1 VALUE2) (ID1 ID2)
'PROPERTY' SET_PROPERTY (HANDLE1 HANDLE2) (VALUE1 VALUE2) (ID1 ID2)
```

If the left argument specifies multiple property names, and the right argument identifies a single target window, then the values array must contain either one item or the same number as elements as the number of property names.

For example, this is a valid expression:

```
'PROPERTY1' 'PROPERTY2' GET_PROPERTY HANDLE (VALUE1 VALUE2)
```

If the left argument specifies multiple property names, and the right argument identifies multiple target windows, then the values array must contain a vector with the same number of elements as the number of property names. Each element of the vector must contain a vector of values with the same number of elements as the number of target windows.

For example, this is a valid use of SET\_PROPERTY:

```
PROPERTY1_VALUES←PROPERTY1_VALUE1 PROPERTY1_VALUE2
PROPERTY2_VALUES←PROPERTY2_VALUE1 PROPERTY2_VALUE2
VALUES←PROPERTY1_VALUES PROPERTY2_VALUES
'PROPERTY1' 'PROPERTY2' SET_PROPERTY HANDLE VALUES (ID1 ID2)
```

Note: See [Class Reference](#) for supported properties and values

## SET\_RANGE

**Purpose:** Set the value of a range property

**Syntax:** [property] SET\_RANGE handle range value [id]

**Arguments:**

property	Character vector - Range property name. Defaults to 'CELL DATA'.
handle	Integer scalar - Window handle (or parent if id is supplied)
range	4 element integer vector - Cell range
	[1] - Row index of first cell in range
	[2] - Column index of first cell in range
	[3] - Number of rows in range
	[4] - Number of columns in range
value	Matrix of range property values
id	Integer scalar - Identifier of child window



or Character vector - Name of child window

Result: None

Note: See [Grid](#) for supported range properties and values

## SHAREWINDOW

Purpose: Share a variable with a window property

Syntax: [property] SHAREWINDOW handle varname [id]

Arguments: property Character vector - Property name. Defaults to 'DATA'.  
handle Integer scalar - Window handle (or parent if id is supplied)  
varname Name of variable to share with property  
id Integer scalar - Identifier of child window  
or Character vector - Name of child window

Result: None

## SHOW

Purpose: Show or hide a window

Syntax: [bool] SHOW handle

Arguments: bool Boolean scalar - 1 to show, 0 to hide  
handle Integer scalar - Window handle

Result: None

## SIZETOTEXT

Purpose: Resize a window to fit its text

Syntax: SIZETOTEXT handle [id]

Arguments: handle Integer scalar - Window handle (or parent if id is supplied)  
id Integer scalar - Identifier of child window  
or Character vector - Name of child window

Result: None

Note: SIZETOTEXT does not support listview, slider, tab, and treeview controls.

## SPACE

Purpose: Space controls equally

Syntax: direction pixels SPACE handles

Arguments: direction Boolean scalar - 0 resize horizontally, 1 resize vertically  
pixels Integer scalar - Number of pixels between controls  
handles Integer vector - Handles of windows to resize

Result: None

## STARTWAIT

Purpose: Make AP 145 start saving events until processed

Syntax: STARTWAIT

Arguments: None  
Result: None

## UNICREATEDLG

Purpose: Create a Unicode dialog  
Syntax: `handle←[owner parent] UNICREATEDLG template`  
Arguments: `owner` Integer scalar - Handle of owner window  
`parent` Integer scalar - Handle of parent window  
`template` Character vector - Contains a dialog template or a list of dialog styles. See [Class Reference](#) for a list of supported styles.  
Result: `handle` Integer scalar - Handle to the dialog window. Zero if an error occurs.

## UNICREATEMENU

Purpose: Create a menu bar using Unicode data  
Syntax: `menu←UNICREATEMENU dialog array`  
Arguments: `dialog` Integer scalar - Handle of dialog  
`array` Vector of menu items  
Result: `menu` Integer scalar - Handle of menu

Menu items have 2, 3 or 4 elements:

- [5] Integer scalar - A unique value that is used to identify the choice in the menu's EVENTS property.
- [6] Array - Menu item data
- [7] Integer scalar - Zero or BITWISE combination of [Menu Style Flags](#). MIS\_BITMAP and MIS\_TEXT are ignored.
- [8] Integer scalar - Zero or BITWISE combination of [Menu Attribute Constants](#).

The menu item data may be a character vector or a character vector and a picture array. The character array is the menu item text. The picture array may be:

- An integer scalar that is the handle of a previously loaded bitmap
- A character vector containing a BMP, EMF, Exif, GIF, ICO, JPEG, PNG, TIFF, or WMF file name
- An enclosed character vector containing the contents of an image file
- A 3 element array:
  - [1] 'ICON' or 'BITMAP'
  - [2] Character vector containing name of executable
  - [3] Integer resource identifier

[GDI+](#) is required for file types other than bitmap and icon.

A menu array can also contain nested vectors of menu items. A nested vector of menu items defines a pull-down menu. The first element of a nested vector of menu items defines the pull-down menu item. The subsequent items define the pull-down menu's items.

See [Adding a Menu Bar](#) for more information about using CREATEMENU.

## UNIFILEDLG, UNIFILEDLGM

**Purpose:** Display a Unicode Open or Save As file dialog  
UNIFILEDLG displays a single-selection dialog.  
UNIFILEDLGM displays a multiple-selection dialog.

**Syntax:** filename←[typelist index [title]] UNIFILEDLG style owner filter

**Arguments:** typelist 2 column matrix - Type names and patterns  
                  [;1] - Type names i.e. 'All Files'.  
                  [;2] - Type pattern, i.e. '\*.\*'.  
index Integer scalar - Index origin 0 index of type name to initially display  
style Boolean scalar - Dialog style. 0 for Open, 1 for Save As  
owner Integer scalar - Handle of owner window  
filter Character vector - File filter. For example, 'c:\temp\\*.txt'.  
title Character vector - Dialog box title

**Result:** filename Character vector - Selected filename, if single file selected.  
                  Vector of character vectors - Selected filenames, if multiple files selected.  
                  ' ' if Open or Save not pressed.

## UNIFOLDERDLG

**Purpose:** Display a Unicode Browse for Folder dialog

**Syntax:** folder←UNIFOLDERDLG owner path

**Arguments:** owner Integer scalar - Handle of owner window  
                  path Character vector - Initial path. ' ' for current directory

**Result:** folder Character vector - Selected folder. ' ' if Ok not pressed.

## UNIMSGBOX

**Purpose:** Display a message in a Unicode box

**Syntax:** result←[owner] MSGBOX text  
          result←[owner] MSGBOX title text [style]

**Arguments:** owner Integer scalar - Handle of message box owner  
                  title Character vector - Message box title. Default: 'APL2 Message'  
                  text Character vector - Line feed delimited message text  
                  style Integer scalar - BITWISE combination of [Message Box Style Flags](#)  
                          Default: MB\_OKCANCEL MB\_ICONEXCLAMATION MB\_MOVEABLE

**Result:** result Integer scalar - See [Message Box Result Codes](#) for the valid codes.

## UNIPOPUPMENU

**Purpose:** Prompt the user with a Unicode popup menu

**Syntax:** index←UNIPOPUPMENU handle x y items

**Arguments:** handle Integer scalar - Handle of the window relative to which the menu is positioned  
                  x Integer scalar - Horizontal distance from window origin  
                  y Integer scalar - Vertical distance from window origin  
                  items Array of popup menu items. Each menu item is a character vector or a vector of character vectors. A character vector defines one menu item. A vector of character vectors defines a submenu. The first character vector defines the

submenu title. Subsequent character vectors define submenu items. Use a null character vector, ' ', to display a separator.

**Result:** Index origin zero index of the user's selection. If the user dismisses the menu without making a selection, -1 is returned. Separators and submenu titles do not affect selection indices.

## WAIT\_EVENT

**Purpose:** Wait for an event

**Syntax:** event←WAIT\_EVENT timeout

**Arguments:** timeout Real numeric scalar - Number of seconds to wait  
0 to check if an event has happened

**Result:** event ' ' if no event occurred

If an AP 145 event, a 5 element vector:

[1] - HANDLE - Integer handle of window or object

[2] - MSG - Integer message identifier

[3] - MP1 - Integer message parameter 1

[4] - MP2 - Integer message parameter 2

[5] - EXP - Arbitrary array, Event handler

If a COM event, a 6 or 7 element vector:

[1] - OBJECT - Integer, Handle of the object that signaled the event

[2] - NAME - Character vector, Event name

[3] - EXP - Arbitrary array, Event handler

[4] - CURSOR - 2 Integers, Position of the mouse pointer

[5] - TIME - Integer, Time the event occurred

[6] - NAMED - 2 column array, Named parameters. Columns are:

[;1] Character vector - Parameter name

[;2] Array - Parameter value

[7] - POSITIONAL - Vector of arrays, Positional arguments

**Notes:**

You can distinguish between AP 145 and COM events by examining the second element of the result. In AP 145 events, the second element is an integer scalar message identifier; in COM events it is a character vector event name.

Some messages occur during application initialization before the application is ready to process messages. To cause APL2 to queue these messages for later processing when the application is ready, call START\_WAIT which will return immediately but start to queue messages.

For COM events, the APL2 application must release the handle of the object that signaled the event and any handles passed in named or positional parameters.

## WINDOWFROMID

**Purpose:** Retrieve the handle of a child from a parent handle and an identifier

**Syntax:** child←WINDOWFROMID parent id

**Arguments:** parent Integer scalar - Handle of parent window

id Integer scalar - Identifier of child window

Result:        `child`        or Character vector – Name of child window  
Integer scalar - Handle of child window (0 if none.)

## GPUTILITY - Utilities

The utility functions may be used to perform common operations with AP 145.

The following functions are tools for performing common high-level user interface operations. For detailed information, consult the HOW\_UTILITY variable in the GUITOOLS workspace.

<a href="#">APLEEDIT</a>	Edit a function using the APL2 file editor
<a href="#">APLEXECPGM</a>	Execute an operating system command
<a href="#">EDIT</a>	Edit an array
<a href="#">HELP145</a>	Display the AP 145 online help
<a href="#">PROMPT</a>	Display a message and prompt for input
<a href="#">QUERYSYSCOLOR</a>	Query the RGB color of a system component
<a href="#">SELECT_1</a>	Display a character matrix and prompt the user to select 1 row
<a href="#">SELECT_SOME</a>	Display a character matrix and prompt the user to some rows
<a href="#">UNIEDIT</a>	Edit an array containing Unicode characters
<a href="#">UNIFILE</a>	Access a Unicode file with a Multibyte name
<a href="#">UNIAFM</a>	Access a Unicode file with a Multibyte name as a character matrix
<a href="#">UNIAFV</a>	Access a Unicode file with a Multibyte name as a vector of character vectors

### APLEEDIT

Purpose: Edit a function using the APL2 file editor  
Syntax: `result←APLEEDIT name`  
Arguments: `name` Character vector - Function or operator name  
Result: `result` Character vector -  $\square$ FX result

### APLEXECPGM

Purpose: Execute an operating system command  
Syntax: `result←APLEXECPGM command`  
Arguments: `command` Character vector - Command to execute  
Result: `result` Integer scalar - Command return code

### EDIT

Purpose: Edit an array  
Syntax: `result←[title] EDIT array`  
Arguments: `title` Character vector - Title of edit window  
`array` Array - Data to edit. Can be character scalar, vector, or matrix, or vector of character vectors, or mixed matrix.  
Result: `result` Edited data - If input is a matrix, then the output is a matrix.  
Otherwise, the output is a vector of character vectors.

### HELP145

Purpose: Display the AP 145 online help  
Syntax: HELP145  
Arguments: None  
Result: None

## PROMPT

Purpose: Display a message and prompt for input  
Syntax: response←PROMPT text  
Arguments: text Character vector - Prompt message  
Result: response 2 element vector  
[1] Boolean scalar - 0 Okay [2] is input, 1 Cancel [2] is null  
[2] Character vector - Input text or null

## QUERYSYSCOLOR

Purpose: Query the RGB color of a system component  
Syntax: rgb←QUERYSYSCOLOR index  
Arguments: index Integer scalar - Color index. See function comments for valid indices  
Result: rgb 3 Integers 0-255 - Red, green, and blue components of system color

## SELECT\_1

Purpose: Display a character matrix and prompt the user to select 1 row  
Syntax: index←SELECT\_1 matrix  
Arguments: matrix Character matrix - Items for user to select from  
Result: index Integer scalar - Index of selected item or  $\square$ IO-1 if Cancel selected.

## SELECT\_SOME

Purpose: Display a character matrix and prompt the user to some rows  
Syntax: indices←SELECT\_SOME matrix  
Arguments: matrix Character matrix - Items for user to select from  
Result: indices Integer vector - Index of selected items or  $\square$ IO-1 if Cancel selected.

## UNIEDIT

Purpose: Edit a Unicode character array  
Syntax: result←[title] UNIEDIT array  
Arguments: title Character vector - Title of edit window  
array Array - Data to edit. Can be character scalar, vector, or matrix, or vector of character vectors, or mixed matrix. Can contain Unicode characters.  
Result: result Edited data - If input is a matrix, then the output is a matrix.  
Otherwise, the output is a vector of character vectors.

## UNIFILE

Purpose: Read and write files.

The monadic forms read either Multibyte or Unicode data.  
The dyadic forms write Unicode data.

**Syntax:** `rdata`  $\leftarrow$  UNIFILE filename  
`rc`  $\leftarrow$  wdata UNIFILE filename

**Arguments:** `filename` Character vector - File name. May include a path.  
`wdata` Character array or zero length numeric vector  
If `wdata` is a character array, it contains the data to write to the file. The data is raveled, converted to Unicode, and prefixed with a byte order mark which identifies the file contents as Unicode, and written to the file. If the file already exists, the old contents are replaced by the new data. If the character array is empty, a 0-length file will be written.  
If `wdata` is a numeric null, the file will be erased.

**Result:** `rdata` Character vector - The contents of the file.  
Any line or record separators are retained as they existed in the file.  
Any error conditions encountered while reading a file are reported by returning a numeric result in place of the data.  
`rc` Integer scalar - The return code from writing a file; zero for success. When an error occurs, the return code of the failing API is returned. The CHECK\_ERROR function can be used to obtain more information.

## UNIΔFM

**Purpose:** Read and write a file as a matrix  
The monadic forms read either Multibyte or Unicode data.  
The dyadic forms write Unicode data.

**Syntax:** `rdata`  $\leftarrow$  UNIΔFM filename  
`rc`  $\leftarrow$  wdata UNIΔFM filename

**Arguments:** `filename` Character vector - File name. May include a path.  
`wdata` Character matrix. The data to write to the file.  
Trailing blanks are removed, the data is converted to Unicode, prefixed with a byte order mark which identifies the file contents as Unicode, and written to the file. If the file already exists, the old contents are replaced by the new data.  
If `wdata` is empty, the file will be erased.

**Result:** `rdata` Character matrix - The contents of the file.  
Records are delimited by line feeds or carriage returns and line feeds. Trailing blanks and end-of-file characters are removed. Any error conditions encountered while reading a file are reported by returning a numeric result in place of the data.  
`rc` Integer scalar - The return code from writing a file; zero for success. When an error occurs, the return code of the failing API is returned. The CHECK\_ERROR function can be used to obtain more information.

## UNIΔFV

**Purpose:** Read and write a file as a vector of character vectors  
The monadic forms read either Multibyte or Unicode data.  
The dyadic forms write Unicode data.

**Syntax:** `rdata`  $\leftarrow$  UNIΔFV filename  
`rc`  $\leftarrow$  wdata UNIΔFV filename



Arguments:	filename	Character vector - File name. May include a path.
	wdata	<p>Vector of character vectors. The data to write to the file.</p> <p>Trailing blanks are removed, the data is converted to Unicode, prefixed with a byte order mark which identifies the file contents as Unicode, and written to the file. If the file already exists, the old contents are replaced by the new data.</p> <p>If <code>wdata</code> is empty, the file will be erased.</p>
Result:	rdata	<p>Vector of character vectors - The contents of the file.</p> <p>Records are delimited by line feeds or carriage returns and line feeds. Trailing blanks and end-of-file characters are removed. Any error conditions encountered while reading a file are reported by returning a numeric result in place of the data.</p>
	rc	<p>Integer scalar - The return code from writing a file; zero for success. When an error occurs, the return code of the failing API is returned. The <code>CHECK_ERROR</code> function can be used to obtain more information.</p>

## GPPRINT - Printing Tools and Constants

The tools in GPPRINT allow applications to produce printed documents.

To create a typical document, the tools are used as follows:

1. Create a logical printer with `CREATE_PRINTER`.
2. Select a printer by using `PRINT_PROPERTY` to set the `PRINTER` property to the name of the desired printer. Use the `PRINTERS` property to retrieve the names of the available printers. Alternatively, display a printer selection dialog with `SELECT_PRINTER`.
3. Set other printer properties such as orientation and margins with `PRINT_PROPERTY`. Alternatively, display a page customization dialog with `PAGE_SETUP`.
4. Start the document with `OPEN_DOCUMENT`.
5. Set desired attributes for the document with `PRINT_PROPERTY` and the `SET_` functions.
6. Add text to the document with `PRINT_SENTENCE`.
7. Begin new lines and pages with `NEWLINE` and `NEWPAGE`.
8. Repeat steps 5, 6, and 7 as needed to create the entire document.
9. Close and send the document to the printer with `CLOSE_DOCUMENT`.
10. Destroy the logical printer with `DESTROY_PRINTER`.

For detailed information, consult the `HOW_PRINT` variable in the `GUITOOLS` workspace.

<code>CANCEL_DOCUMENT</code>	Aborts a document
<code>CLOSE_DOCUMENT</code>	Ends a document
<code>CREATE_PRINTER</code>	Creates a printer object
<code>CREATE_UNICODE_PRINTER</code>	Creates a Unicode printer object
<code>DESTROY_PRINTER</code>	Destroys a printer object
<code>NEWLINE</code>	Starts a new line in a document
<code>NEWPAGE</code>	Starts a new page in a document
<code>OPEN_DOCUMENT</code>	Starts a print document
<code>PAGE_SETUP</code>	Displays the page customization dialog
<code>PRINTER_PROPERTIES</code>	Displays the current printer's Properties dialog
<code>PRINT_PROPERTY</code>	Sets a print document property
<code>PRINT_SENTENCE</code>	Adds a sentence to a print document
<code>QUERY_LENGTH</code>	Queries the length of a sentence
<code>QUERY_PAGENUMBER</code>	Queries the current page number
<code>SELECT_PRINTER</code>	Displays a printer selection dialog
<code>SET_COLOR</code>	Sets the color used in the body of a document
<code>SET_FONT</code>	Sets the fonts used in a document
<code>SET_FOOTING</code>	Sets the running footing text
<code>SET_HEADING</code>	Sets the running heading text
<code>SET_INDENT</code>	Sets the indentation
<code>SET_LINESPACE</code>	Sets the line spacing
<code>SET_MARGIN</code>	Sets the inside margin when duplex printing
<code>SET_PAGENUMBERS</code>	Enables or disables page numbers
<code>SET_WORDBREAK</code>	Enables or disables word break

## ***GUIVARS Constants Reference***

The following groups of constants can be found in the GUIVARS workspace.

- [Accelerator Flags](#)
- [Accelerator Virtual Key Codes](#)
- [Menu Style Flags](#)
- [Menu Attribute constants](#)
- [Menu Box Style Flags](#)
- [Manu Box Result Codes](#)

## Accelerator Flags

Accelerator flags indicate whether accelerator's key entry is a:

AF_CHAR	The accelerator entry is a character
AF_SCANCODE	The accelerator entry is a keyboard scan code
AF_VIRTUALKEY	The accelerator entry is a virtual key code
AF_SHIFT	The keystroke combination includes the Shift key
AF_CONTROL	The keystroke combination includes the Ctrl key
AF_ALT	The keystroke combination includes the Alt key
AF_LONEKEY	The keystroke combination does not include any other keys

The following accelerator flags are defined in GUIVARS but are obsolete and do not work on Windows:

AF_HELP	AF_SYSCOMMAND
---------	---------------

## Accelerator Virtual Key Codes

Virtual key codes are used to define accelerators for non-alphanumeric keys:

VK_ALT	VK_F1
VK_ALTGRAF	VK_F2
VK_BACKSPACE	VK_F3
VK_BACKTAB	VK_F4
VK_BREAK	VK_F5
VK_BUTTON1	VK_F6
VK_BUTTON2	VK_F7
VK_CAPSLOCK	VK_F8
VK_CTRL	VK_F9
VK_DELETE	VK_F10
VK_DOWN	VK_F11
VK_END	VK_F12
VK_ENTER	VK_F13
VK_ESC	VK_F14
VK_HOME	VK_F15
VK_INSERT	VK_F16
VK_LEFT	VK_F17
VK_NEWLINE	VK_F18
VK_NUMLOCK	VK_F19
VK_PAGEDOWN	VK_F20
VK_PAGEUP	VK_F21
VK_PAUSE	VK_F22
VK_PRINTSCRN	VK_F23
VK_RIGHT	VK_F24
VK_SCROLLLOCK	
VK_SHIFT	
VK_SPACE	
VK_TAB	
VK_UP	

Other virtual key codes defined in GUIVARS are obsolete and do not work on Windows.

## Menu Style Flags

Use menu style flags to specify how menu items should be drawn.

MIS_BITMAP	Displays a picture
MIS_BREAK	Starts a new column
MIS_BREAKSEPARATOR	Starts a new column and draws a vertical line
MIS_SEPARATOR	Draws a horizontal line
MIS_STATIC	Menu item is not selectable
MIS_TEXT	Displays text (default)

The following menu style flags are defined in GUIVARS but are not used on Windows:

MIS\_BUTTONSEPARATOR  
MIS\_GROUP  
MIS\_HELP  
MIS\_MULTMENU  
MIS\_OWNERDRAW  
MIS\_SINGLE  
MIS\_SUBMENU  
MIS\_SYSCOMMAND

## Menu Attribute Constants

Use menu style flags to specify the initial state of menu items.

MIA_CHECKED	Draws a check mark to the left of the menu item
MIA_DISABLED	Disable and grey menu item

The STATE CHECKED and STATE ENABLE properties can be used to change the state of menu items:

```
'STATE CHECKED' SET_PROPERTY menu boolean id  
'STATE ENABLE' SET_PROPERTY menu boolean id
```

The following menu attribute constants are defined in GUIVARS but are not used on Windows:

```
MIA_FRAMED  
MIA_HILITED  
MIA_NODISMISS
```

## Message Box Style Flags

One or more of the following flags can be used in the WinMessageBox service's Style parameter.

MB_OK	Contains an Ok button
MB_OKCANCEL	Contains Ok and Cancel buttons
MB_RETRYCANCEL	Contains Retry and Cancel buttons
MB_ABORTRETRYIGNORE	Contains Abort, Retry, and Ignore buttons
MB_YESNO	Contains Yes and No buttons
MB_YESNOCANCEL	Contains Yes, No, and Cancel buttons
MB_ENTER	Contains an Ok button
MB_ENTERCANCEL	Contains Ok and Cancel buttons
MB_APPLMODAL	Disables the rest of the application
MB_CANCEL	Contains a Cancel button
MB_CUACRITICAL	Contains a Stop sign icon
MB_CUANOTIFICATION	Contains an Ok button
MB_CUAWARNING	Contains an exclamation point icon
MB_DEFBUTTON1	The first button is the default
MB_DEFBUTTON2	The second button is the default
MB_DEFBUTTON3	The third button is the default
MB_HELP	Contains a Help button
MB_ICONASTERISK	Contains an information symbol icon
MB_ICONEXCLAMATION	Contains an exclamation point icon
MB_ICONHAND	Contains a Stop sign icon
MB_ICONQUESTION	Contains a question mark icon
MB_SYSTEMMODAL	Disables all other windows

Other message box style flags defined in GUIVARS are obsolete and do not work on Windows.



## Message Box Result Codes

The WinMessageBox service returns one of the following codes indicating which button the user pressed to close the message box:

```
MBID_ABORT  
MBID_CANCEL  
MBID_ENTER  
MBID_ERROR  
MBID_IGNORE  
MBID_NO  
MBID_OK  
MBID_RETRY  
MBID_YES
```

The following message box return code is defined in GUIVARS but is never returned on Windows.

```
MBID_HELP
```

## DEMO145 Function Reference

GUI demonstration programs are found in the DEMO145 workspace in public library 2. The following programs are supplied:

DEMO	Interactive tool to invoke other demonstration programs
DEMO_ACCEL	Keyboard accelerators
DEMO_ACTIVEX_IE	ActiveX interface to Internet Explorer
DEMO_ACTIVEX_IMAGECOMBO	ActiveX ImageCombo controls
DEMO_ACTIVEX_MONTH	ActiveX Month controls
DEMO_AP207	Using AP 207 graphics are in AP 145 dialog
DEMO_APLHELP	Contextual help
DEMO_CHECKS	STATE CHECK property
DEMO_COLOR	Color properties
DEMO_COMBOBOX	Combo box controls
DEMO_CREATECTL	Dynamic control creation
DEMO_CUSTOM	Programming a custom control
DEMO_DATETIME	Date and Time controls
DEMO_DRAGDROP	Handling file drag and drop
DEMO_EVENTS	Event handling
DEMO_EXECUTEDLGX	Handling both AP 145 and another shared variable events
DEMO_EXPLORE	Using dialogs in a Split style dialog
DEMO_GDI	Using Windows Graphics Device Interface (GDI) APIs
DEMO_GRAPHPAK	Using GRAPHPAK in an AP 145 dialog
DEMO_GRID	Grid controls
DEMO_GRID_SCROLL	Scrolling multiple grid controls together
DEMO_GROUP	Position controls within group boxes
DEMO_GUISPACE	Using the GUITOOLS namespace
DEMO_HOVER	Using the Hover event
DEMO_LIMITS	Limiting the amount of data that can be entered
DEMO_LISTVIEW	Listview controls
DEMO_MENU	Menu bars
DEMO_MDI	Using the NoIcon and NoIgnore styles to build MDI style applications
DEMO_MODE	Message boxes and modal and modeless dialogs
DEMO_MONTH	Month controls
DEMO_MOVIE	Multimedia services
DEMO_OPENGL_GLRECT	OpenGL Superbible's GLRECT example
DEMO_OPENGL_GLSAMPLE	Blaine Hodge's GISample
DEMO_PATH	Using the PATH, PATH SELECTION, and PATH STYLE properties
DEMO_PICTURE	Picture and cursor properties
DEMO_POLLING	Polling for AP 145 events during long running operations
DEMO_PRINT	Printing
DEMO_PRINT_POSITION	Positioning output in print jobs
DEMO_PROGRESS	Progress bars
DEMO_PROPERTIES	Using a Tab control in a Properties window
DEMO_RESIZE	Dialogs that can be resized
DEMO_SCROLLBARS	Scroll bars

DEMO_SHARE	Sharing variables with window properties
DEMO_SHEET	Using entry field and grid control as spread sheet style input device
DEMO_SLIDER	Slider controls
DEMO_SPINBUTTONS	Spin button controls
DEMO_SPLIT	Dialogs with multiple panes
DEMO_STATES	State properties
DEMO_STATUS	Dialog status bars
DEMO_TAB	Tab controls
DEMO_TIMER	Timer objects
DEMO_TOOLBAR	Tool bars
DEMO_TREEVIEW	Treeview controls
PRINT_DEMO	Using the GUITOOLS print functions

## Appendix A: AP 145 Services

The following sections describe the services provided by AP 145. Cover functions are provided for most services. Where available, cover functions should be used.

The following APL services are provided:

- [AplCallCOM](#)
- [AplColorDlg](#)
- [AplCopyMem](#)
- [AplCreateControl](#)
- [AplCreateMenu](#)
- [AplCreateObject](#)
- [AplEnableAplInput](#)
- [AplExecPgm](#)
- [AplFileDialog](#)
- [AplFolderDlg](#)
- [AplFontDlg](#)
- [AplFreeService](#)
- [AplGetCellSize](#)
- [AplGetChildren](#)
- [AplGetParent](#)
- [AplGetProfile](#)
- [AplGetProperty](#)
- [AplGetRange](#)
- [AplHelp](#)
- [AplLoadPicture](#)
- [AplLoadService](#)
- [AplPopupMenu](#)
- [AplPrinterProperties](#)
- [AplQueryDdeServers](#)
- [AplReturn](#)
- [AplSetCellSize](#)
- [AplSetFont](#)
- [AplSetProfile](#)
- [AplSetProperty](#)
- [AplSetRange](#)
- [AplShareWindow](#)
- [AplShowContextHelp](#)
- [AplSizeToText](#)
- [AplStrLen](#)
- [AplVersion](#)
- [AplWaitMsg](#)

AP 145 emulates some OS/2 DOS services. These services are only provided for use by cover functions in the GUITOOLS workspace. These services do not provide all the functionality of the OS/2 services and should not be used except by the cover functions. The following OS/2 DOS services are supported:

- DosFreeModule
- DosLoadModule
- DosQueryProcAddr

AP 145 emulates some OS/2 Presentation Manager services. These services are only provided for use by cover functions in the GUITOOLS workspace. These services do not provide all the functionality of the OS/2 services and should not be used except by the cover functions. The following OS/2 Presentation Manager Window services are supported:

- WinCreateDlg
- WinCreateHelpInstance
- WinCreateMenu
- WinCreateSecondaryWindow
- WinCreateWindow
- WinDestroyWindow
- WinEnableWindow
- WinIsWindow
- WinMessageBox
- WinPostMsg
- WinQueryCp
- WinQueryDesktopWindow
- WinQuerySysColor
- WinQueryWindowULong
- WinQueryWindowUShort
- WinRegisterClass
- WinSendMsg
- WinSetWindowPos
- WinWindowFromID

AP 145 provides a set of Unicode services. These services provide the same function as the corresponding Apl and Win services with the enhanced behavior of supporting Unicode. The following Unicode services are supported:

- UniCreateDlg
- UniCreateMenu
- UniFileDialog
- UniFolderDlg
- UniMessageBox
- UniPopupMenu

## Apl Services

### AplCallCOM

**Purpose:** Call the APL2 Component Object Model interface  
**Syntax:** SV145←'AplCallCOM' 'COMMAND' [arguments] comresult  
**Parameters:** The AplCallCOM service's parameters are identical to the COM external function's right argument with the addition of the comresult parameter.

Note: The AP 145 COM service does not support the WAIT command.

comresult    Input: Scalar zero  
                  Output: Event type, error message, and COM result

**Result:**        0                    Success

### AplColorDlg

**Purpose:** Display a color dialog with which the user can set a window's colors  
**Syntax:** SV145←'AplColorDlg' handle [handle...]  
**Parameters:** handle        Integer scalars - Handles of windows whose colors can be changed.  
                                  The first window handle supplied is used as the owner of the colors dialog.  
**Results:**        0                    Success

### AplCopyMem

**Purpose:** Copy data from one storage location to another  
**Syntax:** SV145←'AplCopyMem' target source length  
**Parameters:** target        Pointer - Storage into which the data will be copied.  
                  source        Pointer - Storage from which data will be copied  
                  length        Integer scalar - Number of bytes to copy  
**Result:**        0                    Success  
                  1                    Access violation

### AplCreateControl

**Purpose:** Create a control window.  
**Syntax:** SV145←'AplCreateControl' parent class styles [id [ctldata]]  
**Parameters:** parent        Integer scalar - Handle of parent window  
                  class        Character vector - Control class to create  
                  styles        Character vector - List of control styles  
                  id            Integer scalar - Control identifier  
                  ctldata        Control class specific creation data  
**Result:**        Handle of control window. Zero if error.

### AplCreateMenu

Purpose: Create a menu bar using Multibyte data  
 Syntax: `SV145←AplCreateMenu dialog array`  
 Parameters: `dialog` Integer scalar - Handle of dialog  
               `array` Vector of menu items  
 Result: `menu` Integer scalar - Handle of menu

Menu items have 2, 3 or 4 elements:

- [1] Integer scalar - A unique value that is used to identify the choice in the menu's EVENTS property.
- [2] Array – Menu item data
- [3] Integer scalar - Zero or BITWISE combination of [Menu Style Flags](#). MIS\_BITMAP and MIS\_TEXT are ignored.
- [4] Integer scalar - Zero or BITWISE combination of [Menu Attribute Constants](#).

The menu item data may be a character vector or a character vector and a picture array. The character array is the menu item text. The picture array may be:

- An integer scalar that is the handle of a previously loaded bitmap or one of the following built-in picture values:
  - 1 Cut
  - 2 Copy
  - 3 Paste
  - 4 Undo
  - 5 Redo
  - 6 Delete
  - 7 New
  - 8 Open
  - 9 Save
  - 10 Print Preview
  - 11 Properties
  - 12 Help
  - 13 Find
  - 14 Replace
  - 15 Print
  - 16 Large Icons View
- A character vector containing a BMP, EMF, Exif, GIF, ICO, JPEG, PNG, TIFF, or WMF file name
- An enclosed character vector containing the contents of an image file
- A 3 element array:
  - [1] 'ICON' or 'BITMAP'
  - [2] Character vector containing name of executable
  - [3] Integer resource identifier

[GDI+](#) is required for file types other than bitmap and icon.

A menu array can also contain nested vectors of menu items. A nested vector of menu items defines a pull-down menu. The first element of a nested vector of menu items defines the pull-down menu item. The subsequent items define the pull-down menu's items.

See [Adding a Menu Bar](#) for more information about using CREATEMENU.

### **AplCreateObject**

**Purpose:** Create an object  
**Syntax:** SV145←'AplCreateObject' class [classdata]  
**Parameters:** class Character vector - Object class to create  
classdata Class specific creation data  
**Result:** Handle of object. Zero if error.

See [Objects: DDE and Timers](#) for information about object classes.

### **AplEnableAplInput**

**Purpose:** Enable or disable APL keyboard input for a window  
**Syntax:** SV145←'AplEnableAplInput' handle state  
**Parameters:** handle Integer scalar - Window handle  
state Boolean scalar - 1 APL input enabled, 0 disabled  
**Results:** 0 Success  
1 APL input already enabled or disabled  
2 Windows service error

To display the keyboard modification dialog, either enable or disable APL input and post an APLKEY\_MSG\_MODIFY message to the window

### **AplExecPgm**

**Purpose:** Execute an operating system command  
**Syntax:** SV145←'AplExecPgm' command cmdrc visible leave  
**Parameters:** command Character vector - Command to execute. Must be a name of an executable file.  
rc Input: Integer scalar - value ignored  
Output: Integer scalar - Command's return code  
visible Boolean scalar - 0 command is executed in an invisible window, 1 a visible one.  
leave Boolean scalar - 0 window is closed when command completes, 1 window is left open.  
**Results:** 0 Success  
1 Command cancelled  
2 Windows service error  
3 Invalid command

**Note:** Leave is ignored. Command windows are always closed automatically.

### **AplFileDlg**



**Purpose:** Display a modal file dialog

**Syntax:** SV145←'AplFileDlg' owner style name id rc [types index [title]]

**Parameters:**

owner	Integer scalar - Handle of the dialog's owner
style	Integer scalar FDS_OPEN_DIALOG for an Open dialog; FDS_SAVEAS_DIALOG for a Save As dialog.
name	Input – Character vector filename filter, for example, c:\temp\*.txt Output – Filename(s) Character vector if single file selected Vector of character vectors if multiple files selected Null if the user did not press Ok
id	Integer scalar - 0 for single-selection dialog, 1 for multiple-selection dialog.
rc	Integer scalar - Updated with dialogs return code. 0 if the user pressed Open or Save, 1 otherwise
types	Matrix of character vectors [;1] Type name, i.e. 'All Files' [;2] Type pattern, i.e. '*.*'
index	Integer scalar - Zero origin index of row of types Input - Index of file type to display Output - Index of user's selected type
title	Character vector - Title of dialog box

**Results:**

0	Success
1	Windows service error or invalid argument

### **AplFolderDlg**

**Purpose:** Display a browse for folder dialog

**Syntax:** SV145←'AplFolderDlg' owner path

**Parameters:**

owner	Integer scalar - Handle of the dialog's owner
path	Character vector Input - Initial path. ' ' for current directory Output – Selected folder if the user presses Ok, null otherwise.

**Results:**

0	Success
1	Windows service error or user pressed Cancel

### **AplFontDlg**

**Purpose:** Display a font dialog with which the user can set a window's font

**Syntax:** SV145←'AplFontDlg' handle id

**Parameters:**

handle	Integer scalar - Handle of window whose font can be changed
id	Integer scalar - Identifier for the dialog, unused

**Results:**

0	Success
1	Windows service error

### **AplFreeService**

**Purpose:** Unload one or more services previously loaded with the AplLoadService service.

**Syntax:** SV145←'AplFreeService' service

Parameters: `service` Character vector or vector of character vectors - Names of services to free  
 Results: 0 Success  
 1 Service not loaded  
 2 Unload failed

### **AplGetCellSize**

Purpose: Get the size of a grid control cell  
 Syntax: `SV145←'AplGetCellSize' handle cell size [id]`  
 Parameters: `handle` Integer scalar – Grid control handle  
           `cell` 2 element integer vector - Row and column index of the cell  
           `size` Input: Integer scalar, value ignored  
                   Output: Updated with 2 element integer vector - Number of rows and columns the cell spans.  
           `id` Integer scalar - Identifier of child grid window or Character vector – Name of child grid window  
 Results: 0 Success  
 1 Invalid window handle or id  
 2 Invalid range property name for class of window  
 3 System error (out of storage, threads, or SVP error)

### **AplGetChildren**

Purpose: Get the handles of a parent's child windows  
 Syntax: `SV145←'AplGetChildren' parent children`  
 Parameters: `parent` Integer scalar - Handle of a parent window.  
           `children` Input: Integer scalar, value ignored  
                   Output: Integer vector, child window handles, null if none  
 Results: 0 Success

### **AplGetParent**

Purpose: Get the handle of a child window's parent  
 Syntax: `SV145←'AplGetParent' handle`  
 Parameters: `handle` Integer scalar - Window handle  
 Results: 0 Window does not have a parent  
 Other Handle of parent window

### **AplGetProfile**

Purpose: Get a profile value from the registry  
 Syntax: `SV145←'AplGetProfile' com app ver fld key val`  
 Parameters: `com` Character vector - Company name  
           `app` Character vector - Application name  
           `ver` Character vector - Version  
           `fld` Character vector - Folder, ' ' if none  
           `key` Character vector - Key  
           `val` Character vector - Default value (updated with profile value)

Results: 0 Success

Purpose: Get a profile value from an INI file

Syntax: SV145←'AplGetProfile' fil app key val

Parameters: fil Character vector - INI file name  
app Character vector - Application name  
key Character vector - Key  
val Character vector - Default value (updated with profile value)

Results: 0 Success

To query the applications, versions, folders, or keys within the registry, or the applications or keys within an INI file, set the application name, version, folder, or key name to ' '.

### AplGetProperty

Purpose: Get the value of a window property.

Syntax: SV145←'AplGetProperty' handles properties values [ids]

Parameters: handles Integer scalar or vector - Window handles  
properties Character vector or vector of character vectors - Property names  
values Input: Integer scalar, value ignored  
Output: Values of properties  
ids Integer scalar or vector - Child window identifiers  
Character vector or vector of character vectors - Names of child windows  
If ids are supplied, the child windows' properties are retrieved.

Results: 0 Success  
1 Invalid window handle or id  
2 Invalid property name for class of window  
3 System error (out of storage, threads, or SVP error)

See [GET\\_PROPERTY](#) for more information.

### AplGetRange

Purpose: Get the value of a window range property.

Syntax: SV145←'AplGetRange' handle property range value [id]

Parameters: handle Integer scalar - Window handle  
property Character vector - Range property name  
range 4 element integer vector - Range  
[1 2] - Row and column index of first cell in range  
[3 4] - Number of rows and columns in range  
value Input: Integer scalar, value ignored  
Output: Value of range property  
id Integer scalar: Identifier of window that is a child of handle  
or Character vector - Name of child window  
If id is supplied, the child window's range property is retrieved.

Results: 0 Success  
1 Invalid window handle or id  
2 Invalid range property name for class of window

**AplHelp**

**Purpose:** Display the AP 145 online help.  
**Syntax:** SV145←'AplHelp' topic  
**Parameters:** topic Character vector - Topic name. No longer supported. Use ' '.  
**Results:** 0 Success  
 1 Invalid topic

**AplLoadPicture**

**Purpose:** Load a picture for use in ActiveX objects.  
**Syntax:** SV145←'AplLoadPicture' picture  
**Parameters:** picture Character vector - Name and path of BMP, JPG, GIF, and PNG files  
 Enclosed character vector - contents of BMP, ICO, or WMF file.  
**Results:** result Integer scalar - Handle of picture object, 0 if error  
 The handle is an IDispatch interface and is useable with the ListImages class's Add method.

**AplLoadService**

**Purpose:** Load one or more APIs from a dynamic link library (DLL)  
**Syntax:** SV145←'AplLoadService' service library [descrip [result]]  
**Parameters:** service Character vector or vector of character vectors - Names of API to load from the dynamic link library. One name per character vector. The names are case sensitive.

If the `service` argument is a character vector containing a single API name, then `descrip` and `result` arguments may be supplied.

If the `service` argument contains more than one API name or if the `descrip` argument is omitted, then AP 145 will pass all the APIs' parameters by value and as 4 byte signed integers or single byte character strings.

`library` Identifies the location of the API. It can be either:
 

- Character vector - Filename of the DLL containing the APIs
- Integer scalar - Address of the API

`descrip` Integer vector – Description of single API's parameters.

The length of the description corresponds to the number of parameters required by the API. Each item corresponds to a parameter and describes that parameter. Descriptions may not contain more than 32 items. Pass a null vector to indicate that the service has no parameters.

The absolute values of the items are interpreted as follows:

- 0 - Automatic type handling. Parameter is passed by value
- 1 - Automatic type handling. Parameter is passed by reference

- 2 - Parameter is 1 byte signed integer
- 3 - Parameter is 2 byte signed integer
- 4 - Parameter is 4 byte signed integer
- 5 - Parameter is 4 byte floating point number
- 6 - Parameter is 8 byte floating point number
- 7 - Parameter is string of single byte characters
- 8 - Parameter is string of 2 byte Unicode characters

For types 2 through 8, a positive number indicates the parameter should be passed by value and a negative number indicates it should be passed by reference.

`result` Integer scalar – Description of single API's result.

The value of `result` is interpreted as follows:

- 2 - Result is 1 byte signed integer
- 3 - Result is 2 byte signed integer
- 4 - Result is 4 byte signed integer
- 5 - Result is 4 byte floating point number
- 6 - Result is 8 byte floating point number
- 7 - Result is string of single byte characters
- 8 - Result is string of 2 byte Unicode characters

Results:	0	Success
	1	Service name already in use.
	2	Load failed
	-1	Insufficient space

## **AplPopupMenu**

**Purpose:** Prompt the user with a popup menu

**Syntax:** `SV145←'AplPopupMenu' owner x y items`

**Parameters:** `owner` Integer scalar - Handle of the window relative to which the menu should be positioned  
`x and y` Coordinates of the popup menu position relative to the owner's origin  
`items` Array of popup menu items. Each menu item is a character vector or a vector of character vectors. A character vector defines one menu item. A vector of character vectors defines a submenu. The first character vector defines the submenu title. Subsequent character vectors define submenu items. Use a null character vector, ' ', to display a separator.

**Results:** Index origin zero index of the user's selection. If the user dismisses the menu without making a selection, -1 is returned. Separators and submenu titles do not affect selection indices.

## **AplPrinterProperties**

**Purpose:** Display the current printer's Properties dialog

**Syntax:** `SV145←'AplPrinterProperties' printer`

**Parameters:** `printer` Integer scalar - Handle of a printer created by `CREATE_PRINTER` or `CREATE_UNICODE_PRINTER`

**Results:** Scalar integer - 0 if the user presses Ok. 1 if the user presses Cancel.

## **AplQueryDdeServers**

**Purpose:** Retrieve the names of the active DDE servers and the topics they support  
**Syntax:** SV145←'AplQueryDdeServers' list  
**Parameters:** list           Input: Integer scalar - value ignored  
                                  Output: 2 column matrix of character vectors  
                                  [;1] Server name  
                                  [;2] Topic name  
**Results:**     0            Success  
                  1            Unable to create query object  
                  2            Out of memory

Note: Only names of servers that support topics are returned.

## **AplReturn**

**Purpose:** Return a message result  
**Syntax:** SV145←'AplReturn' mresult  
**Parameters:** mresult    Integer scalar - Message result  
**Results:**     0            Success  
                  1            No message is waiting for a result from this variable.

## **AplSetCellSize**

**Purpose:** Enlarge a cell so that it covers adjacent cells  
**Syntax:** SV145←'AplSetCellSize' handle cell size [id]  
**Parameters:** handle    Integer scalar - Window handle  
                  cell     2 element integer vector - Row and column index of the cell  
                  size     2 element integer vector - Number of rows and columns the cell should span  
                  id       Integer scalar - Identifier of a child window whose parent is handle  
                                  or Character vector - Name of child window  
**Results:**     0            Success  
                  1            Invalid window handle or id  
                  2            Invalid cell index or size or system error

## **AplSetFont**

**Purpose:** Set a window's font  
**Syntax:** SV145←'AplSetFont' handle font  
**Parameters:** handle    Handle of the window whose font is to be set  
                  font     Character vector - Point size and face name. i.e. '10.Times New Roman'  
**Results:**     0            Success  
                  1            Windows Service Error  
                  2            Face name not available  
                  3            Point size not available  
                  4            Invalid font name

## AplSetProfile

**Purpose:** Set a profile value in the registry  
**Syntax:** SV145←'AplSetProfile' com app ver fld key val  
**Parameters:** com Character vector - Company name  
app Character vector - Application name  
ver Character vector - Version  
fld Character vector - Folder, ' ' if none  
key Character vector - Key  
val Character vector - Profile value  
**Results:** 0 Success

**Purpose:** Set a profile value in an INI file  
**Syntax:** SV145←'AplSetProfile' fil app key val  
**Parameters:** fil Character vector - INI file name  
app Character vector - Application name  
key Character vector - Key  
val Character vector - Profile value  
**Results:** 0 Success

To delete companies, applications, versions, folders, or keys within the registry, or applications or keys within an INI file, set the contained application, version, folder, key name, or value to 10. Arguments after a null application name, version, folder, or key name are ignored. AP 145 prompts the user for delete confirmation.

## AplSetProperty

**Purpose:** Set a window property  
**Syntax:** SV145←'AplSetProperty' handles properties values [ids]  
**Parameters:** handles Integer scalar or vector - Window handles  
properties Character vector - Property names  
values Values of properties  
ids Integer scalar or vector - Child window identifiers  
Character vector or vector of character vectors - Names of child windows  
If ids are supplied, the child windows' properties are set.  
**Results:** 0 Success  
1 Invalid window handle or id  
2 Invalid property name for class of window  
3 System error (out of storage, threads, or SVP error)  
4 Invalid property value

See [SET PROPERTY](#) for more information.

## AplSetRange

**Purpose:** Set a range property  
**Syntax:** SV145←'AplSetRange' handle property range value [id]  
**Parameters:** handle Integer scalar - Window handle  
property Character vector - Range property name

	range	4 element integer vector - Range [1 2] - Row and column index of first cell in range [3 4] - Number of rows and columns in range
	value	Value of range property
	id	Integer scalar - Identifier of window that is a child of handle. Or Character vector - Name of child window If id is supplied, the child window's property is set.
Results:	0	Success
	1	Invalid window handle or id
	2	Invalid range property name for class of window
	3	System error (out of storage, threads, or SVP error)
	4	Invalid range property value

## AplShareWindow

Purpose:	Share a variable with a window property	
Syntax:	SV145←'AplShareWindow' handle svname property [id]	
Parameters:	handle	Integer scalar - Window handle
	svname	Character vector - Variable name to share
	property	Character vector - Property name
	id	Integer scalar - Identifier of window that is a child of handle. Or character vector -Name of child window If id is supplied, the variable is shared with the child window's property.
Results:	0	Success - Variable offered by AP 145
	1	Invalid window handle or id
	2	Invalid property name for class of window
	3	System error (out of storage, threads, or SVP error)

### Important Usage Notes:

The AplShareWindow service does not share a variable with a window. Rather, AplShareWindow causes AP 145 to offer a variable to the processor that called the service. It is the responsibility of the application to match the share offer before making use of the variable. The function SHAREWINDOW calls AplShareWindow and matches AP 145's subsequent offer. Use of SHAREWINDOW rather than calling AplShareWindow directly is strongly advised. SHAREWINDOW does not support variable names that begin with the letters Δ.

AP 145 does not retract a variable shared with a window property when the window is destroyed or when the variable that created the window is retracted. AP 145 only retracts a variable after its offer has been accepted and the variable has been retracted. Therefore, you should always accept variables offered by AP 145 when using AplShareWindow. It is strongly suggested that variables shared with window properties be localized so they are automatically retracted upon application termination.

AP 145 does not specify variables shared with window properties every time the window properties change. AP 145 detects when the variable is referenced and immediately specifies the variable with the current value of the property. Therefore, you cannot use □SVE to wait for a property to change. If you need to respond to a window event, set the EVENTS property for the appropriate event name.



## **AplShowContextText**

**Purpose:** Show context help  
**Syntax:** SV145←'AplShowContextHelp' handle [id]  
**Parameters:** handle Integer scalar - Control, dialog, or menu handle  
id Integer scalar - Control identifier or menu item identifier  
or Character vector - Control name  
**Results:** 0 Success

## **AplSizeToText**

**Purpose:** Set the size of a control so that it just fits the control's text  
**Syntax:** SV145←'AplSizeToText' handle [id]  
**Parameters:** handle Integer scalar - Window handle  
id Integer scalar - Child window identifier  
or Character vector - Name of child window  
**Results:** 0 Success

Note: AplSizeToText does not support listview, slider, tab, and treeview controls.

## **AplStrLen**

**Purpose:** Get the length of a null terminated string  
**Syntax:** SV145←'AplStrLen' address  
**Parameters:** address Integer scalar - Address of string  
**Result:** Integer scalar - Number of characters in the string excluding the null

## **AplVersion**

**Purpose:** Query the current service level  
**Syntax:** SV145←,c'AplVersion'  
**Parameters:** None  
**Result:** Integer scalar - Service level

## **AplWaitMsg**

**Purpose:** Wait for messages and optionally provide a message result  
**Syntax:** SV145←'AplWaitMsg' handle msg mp1 mp2 [handler] [timeout]  
**Parameters:** handle Input: Integer scalar - Message result (optional)  
Output: Integer scalar - Window handle  
msg Input: Integer scalar, value ignored  
Output: Integer scalar - Message identifier  
mp1 Input: Integer scalar, value ignored  
Output: Integer scalar - Message parameter 1  
mp2 Input: Integer scalar, value ignored  
Output: Integer scalar - Message parameter 2  
handler Input: Boolean scalar - 0 Wait for messages, 1 Start queuing messages  
Output: Arbitrary array - Event handler

timeout      Input: Real numeric scalar – Number of seconds to wait for an event.  
0 to check if an event has happened.  
If a timeout is not provided, AplWaitMsg waits forever.  
Output: Reserved

Result:      0                      Success  
              1                      No event occurred in timeout seconds

Notes:

Messages which do not have event handler expressions receive default processing.

While AplWaitMsg is not waiting, all messages are dispatched to the appropriate window procedure. After a message result is returned with AplReturn and until AplWaitMsg is called to wait again, all messages are dispatched normally. To avoid messages with event handler expressions being dispatched, do not use AplReturn. Use AplWaitMsg to supply message results and continue to wait for messages.

Some messages occur during application initialization before the application is ready to wait for messages. To cause AP145 to queue these messages for later processing when the application is ready, call AplWaitMsg with the fifth parameter set to 1. AplWaitMsg will return immediately but start to queue messages.

## **Appendix B: Where to find Tools, Samples, and Other Information**

To make effective use of APL2's GUI application programming tools, it is important that you know where to find APL2's GUI application development tools and information about how to use them.

The document you are reading, **APL2 Programming: Developing GUI Applications**, is the primary source of information. The GUI Applications choice on the session manager's Help menu opens this book. The HELP145 function in the GUITOOLS and DEMO145 workspaces also opens it.

### **GUITOOLS**

The GUI programming tools are found in the GUITOOLS workspace in public library 2.

The GUITOOLS workspace contains four groups:

GPDESC	Documentation variables
GPDLGPROCESS	Tools for creating and processing dialogs
GPPRINT	Tools for printing
GPUTILITY	Miscellaneous utility tools

The GUITOOLS workspace has several variables that contain reference information:

ABSTRACT	Workspace overview
DESCRIBE	Description of workspace's groups
HOW	Instructions for using workspace's groups

The DESCRIBE\_ variables contain brief descriptions of the available tools:

DESCRIBE_DLGPROCESS	Tools for creating and processing dialogs
DESCRIBE_PRINT	Tools for printing
DESCRIBE_UTILITY	Tools that provide common interface features

The HOW\_ variables contain detailed syntax, argument, and usage information:

HOW_CONTROL	Class and style information for controls and dialogs
HOW_DLGPROCESS	Tools for creating and processing dialogs
HOW_EVENTS	Events supported by controls and dialogs
HOW_EVENTS_CODE	Sample code for setting event handlers
HOW_PRINT	Print utilities
HOW_PROPERTY	Properties supported by controls and dialogs
HOW_PROPERTY_CODE	Sample code for setting properties
HOW_RANGE	Range properties support by grid controls
HOW_RANGE_CODE	Sample code for setting range properties
HOW_UTILITY	Tools that provide common interface features

In addition, each function contains comments describing its usage, syntax, arguments, and results.

## DEMO145

GUI demonstration programs are found in the DEMO145 workspace in public library 2. There are functions to demonstrate how to use most kinds of controls and properties. For example, the DEMO\_GRID function demonstrates how to use grid controls. The top level DEMO function lists the available demonstration functions.

## GUIVARS

GUI constants are found in the GUIVARS workspace in public library 2. These constants are used with several of the tools in GUITOOLS.

## DDESHARE

The DDESHARE workspace in public library 2 contains tools for communicating with other applications using Dynamic Data Exchange.

The DDESHARE workspace contains two groups:

GPDEMO	Demonstration DDE server and clients
GPUTILITY	Functions for creating and using DDE connections

The DDESHARE workspace has three variables that contain reference information:

ABSTRACT	Workspace overview
DESCRIBE	Brief description of workspace's groups
HOW	Detailed instructions for using workspace's tools

## Unicode

Documentation about APL2's Unicode support is provided in the Double Byte Character Set Support appendix of the **APL2 User's Guide**.

## Dialog Editor

The dialog editor is a graphical window design tool that can be used to create dialog templates. A dialog template is an array that defines a dialog and its controls.

The dialog editor is part of the APL2 Session Manager. Invoke the dialog editor from either the Open Object dialog on the Edit pull down menu or by double clicking on an existing dialog template name.

## Appendix C: Using APL2, DDE and Microsoft Excel

This appendix demonstrates how to use DDE to send commands to Microsoft Excel.

First, you need to make sure Excel is running. Try to create a DDE COMMAND object. If it succeeds, Excel is running. If it fails, Excel is not running. Here's how to do it:

```
HCMD←CREATEOBJ 'DDE COMMAND' 'Excel' 'System'  
HCMD ⍎ HCMD is zero if Excel is not running  
0
```

To start Excel, you could use AP 100:

```
)COPY 2 WINDOWS HOST  
SAVED 1999-02-03 10.53.26 (GMT-7)  
HOST 'START EXCEL /e'  
0
```

Or, use the Windows ShellExecute API:

```
SV145←'AplLoadService' 'ShellExecuteA' 'SHELL32'(0 0 0 0 0 0)  
(APRC OSRC CMD)←SV145  
⍀ES(APRC≠0)/'AP 145 failed'  
⍀ES(OSRC≠0)/'AplLoadService failed'  
  
SHOWCMD←10 ⍎ SW_SHOWDEFAULT  
SV145←'ShellExecuteA' 1 'open' 'excel.exe' 0 '' SHOWCMD  
(APRC OSRC CMD)←SV145  
⍀ES(APRC≠0)/'AP 145 failed'
```

Once Excel is running (and you have successfully created a DDE COMMAND object), you can send commands to Excel:

```
SET_PROPERTY HCMD command
```

Or, you can share a variable with the object for easier programming:

```
SHAREWINDOW HCMD 'EXCEL'
```

Note: Word uses WordBasic rather than Visual Basic for commands sent through DDE connections. So, Visual Basic commands like this:

```
Workbooks.Open FileName:="C:\TEMP\test.xls"
```

Will not work through DDE. You must use WordBasic commands like these:

```
EXCEL←'[Open("d:\text.xls")]' ⍎ To open a workbook  
EXCEL←'[Close("d:\text.xls")]' ⍎ To close a workbook  
EXCEL←'[Quit()]' ⍎ To shutdown Excel
```

## Appendix D: Using APL2, DDE and Microsoft Access

This appendix demonstrates how to use DDE to work with Microsoft Access from APL2.

Like Excel, you must first make sure Access is running. You can try to create a DDE COMMAND object. If it fails, Access needs to be started.

```
APPNAME←'MSAccess'  
HCMD←CREATEOBJ 'DDE COMMAND' APPNAME 'System'  
HCMD
```

0

Start Access like this:

```
SHOWCMD←10 ⌘ SW_SHOWDEFAULT  
SV145←'ShellExecuteA' 1 'open' 'msaccess.exe' 0 '' SHOWCMD  
(APRC OSRC CMD)←SV145  
⊞ES(APRC≠0)/'AP 145 failed'
```

Once Access is running (and you have successfully created a DDE COMMAND object), you can send commands to Access. For example, you can open a database:

```
DB←'D:\PROGRAM FILES\MICROSOFT OFFICE\OFFICE\SAMPLES\NORTHWIND.MDB'  
SET_PROPERTY HCMD '[OpenDatabase ',DB,']'  
GET_PROPERTY HCMD
```

0

Here is a function that is useful for retrieving lists of things:

```
VR←GET_ACCESS_LIST ARG;APPNAME;TOPIC;TYPE;HDATA;DATA  
[1] (APPNAME TOPIC TYPE)←ARG  
[2] HDATA←CREATEOBJ 'DDE DATA' APPNAME TOPIC TYPE  
[3] DATA←'XLTABLE DATA' GET_PROPERTY HDATA  
[4] R←⊞IO⇒DATA  
[5] R←(R≠⊞AF 9)⊂R  
[6] ∇
```

Retrieve the list of available tables:

```
⇒GET_ACCESS_LIST APPNAME DB 'TableList'  
Categories  
Customers  
Employees  
Order Details  
Orders  
Products  
Shippers  
Suppliers
```

Retrieve the list of available queries:

```
>GET_ACCESS_LIST APPNAME DB 'QueryList'  
Category Sales for 1995  
Current Product List  
Customers and Suppliers by City  
Employee Sales by Country  
Invoices  
Invoices Filter  
Order Details Extended  
Order Subtotals  
Product Sales for 1995  
Products Above Average Price  
Quarterly Orders  
Quarterly Orders by Product  
Sales by Category  
Sales by Year  
Ten Most Expensive Products
```

Retrieve the list of available forms:

```
>GET_ACCESS_LIST APPNAME DB 'FormList'  
Categories  
Customer Labels Dialog  
Customer Orders  
Customer Orders Subform1  
Customer Orders Subform2  
Customer Phone List  
Customers  
Employees  
Employees (page break)  
Main Switchboard  
Orders  
Orders Subform  
Product List  
Products  
Quarterly Orders  
Quarterly Orders Subform  
Sales Analysis  
Sales by Year Dialog  
Sales Reports Dialog  
Startup  
Suppliers
```

Retrieve the list of available reports:

```
>GET_ACCESS_LIST APPNAME DB 'ReportList'  
Alphabetical List of Products  
Catalog  
Customer Labels  
Employee Sales by Country  
Invoice  
Products by Category  
Sales by Category  
Sales by Category Subreport  
Sales by Year  
Sales by Year Subreport  
Sales Totals by Amount  
Summary of Sales by Quarter  
Summary of Sales by Year
```

Retrieve the list of available macros:

```
>GET_ACCESS_LIST APPNAME DB 'MacroList'  
Customer Labels Dialog  
Customer Phone List  
Customers  
Employees (page break)  
Sales Totals by Amount  
Sample Autokeys  
Suppliers
```

Retrieve the list of available modules:

```
>GET_ACCESS_LIST APPNAME DB 'ModuleList'  
Startup  
Utility Functions
```

Open a form (forms are dialogs in APL2 terms):

```
SET_PROPERTY HCMD '[OpenForm Employees,0,,1,0]'  
GET_PROPERTY HDMC  
0
```

Get all the field names and data from a table:

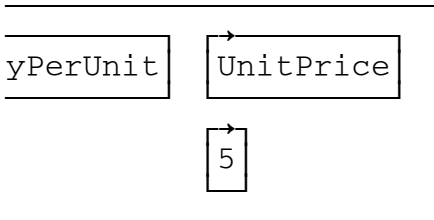
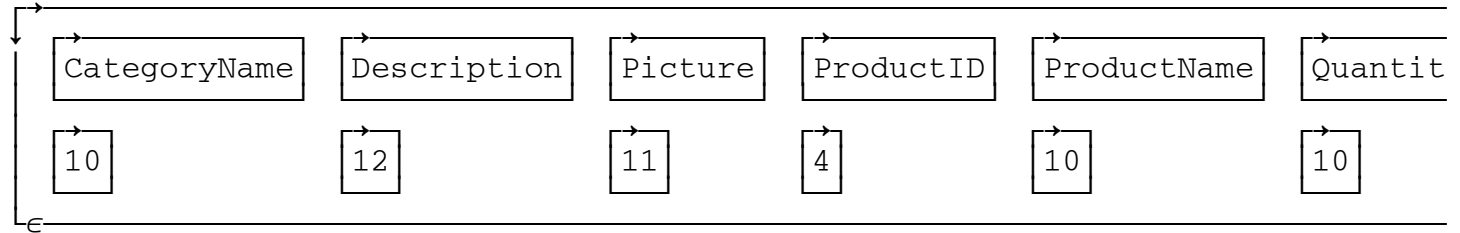
```
TABLE←';TABLE Shippers'  
HDATA←CREATEOBJ 'DDE DATA' APPNAME (DB, TABLE) 'All'  
'XLTABLE DATA' GET_PROPERTY HDATA  
ShipperID CompanyName Phone  
1 Speedy Express (503) 555-9831  
2 United Package (503) 555-3199  
3 Federal Shipping (503) 555-9931  
DESTROYOBJ HDATA
```

Get the field names and types returned by a predefined query:

```
QUERY←';QUERY Catalog'
```



```
HDATA←CREATEOBJ 'DDE DATA' APPNAME (DB,QUERY) 'FieldNames;T'
DISPLAY 'XLTABLE DATA' GET_PROPERTY HDATA
```



```
DESTROYOBJ HDATA
```

Use a predefined query to get some data:

```
QUERY←';QUERY Ten Most Expensive Products'
HDATA←CREATEOBJ 'DDE DATA' APPNAME (DB,QUERY) 'All'
'XLTABLE DATA' GET_PROPERTY HDATA
TenMostExpensiveProducts UnitPrice
C#te de Blaye 263.5
Th#ringer Rostbratwurst 123.79
Mishi Kobe Niku 97
Sir Rodney's Marmalade 81
Carnarvon Tigers 62.5
Raclette Courdavault 55
Manjimup Dried Apples 53
Tarte au sucre 49.3
Ipoh Coffee 46
R#ssle Sauerkraut 45.6
DESTROYOBJ HDATA
```

Retrieve the SQL statement representing a table:

```
TABLERNAME←';TABLE Shippers'
HDATA←CREATEOBJ 'DDE DATA' APPNAME (DB,TABLERNAME) 'SQLText'
GET_PROPERTY HDATA
SELECT * FROM Shippers WITH OWNERACCESS OPTION;
DESTROYOBJ HDATA
```

Perform a SQL statement:

```
SQL←';SQL SELECT * FROM Orders WHERE OrderID > 10050;'
HDATA←CREATEOBJ 'DDE DATA' APPNAME (DB,SQL) 'FieldNames;T'
DISPLAY 'XLTABLE DATA' GET_PROPERTY HDATA
```

OrderID	CustomerID	EmployeeID	OrderDate	RequiredDate	ShippedDa
4	10	4	8	8	8

---

te	ShipVia	Freight	ShipName	ShipAddress	ShipCity	ShipRegion	S
	4	5	10	10	10	10	1

---

ShipPostalCode	ShipCountry
0	10

DESTROYOBJ HDATA

Perform another SQL statement:

```
SQL←';SQL SELECT OrderID, CustomerID FROM Orders'
SQL←SQL, ' WHERE OrderID > 11050;'
HDATA←CREATEOBJ 'DDE DATA' APPNAME (DB,SQL) 'Data'
'XLTABLE DATA' GET_PROPERTY HDATA
```

```
11051 LAMAI
11052 HANAR
11053 PICCO
11054 CACTU
11055 HILAA
11056 EASTC
11057 NORTS
11058 BLAUS
11059 RICAR
11060 FRANS
11061 GREAL
11062 REGGC
11063 HUNGO
11064 SAVEA
11065 LILAS
11066 WHITC
11067 DRACD
11068 QUEEN
11069 TORTU
```

```
11070 LEHMS
11071 LILAS
11072 ERNSH
11073 PERIC
11074 SIMOB
11075 RICSU
11076 BONAP
11077 RATTC
      DESTROYOBJ HDATA
```

And finally, to shut down Access:

```
      SET_PROPERTY HCMD '[Quit] '
      DESTROYOBJ HCMD
```

For further information about using Access through DDE, consult the Microsoft Access online help.

1. Select Contents and Index from the Help pull down menu.
2. In the Help Topics window, select the Index tab.
3. Type the topic: Servers, DDE
4. Hit Enter to display "Use Microsoft Access as a DDE Server"

## ***Appendix E: Microsoft Windows Common Control Library***

The common controls are a set of window classes that are implemented by updates to the common control library, which is a dynamic-link library (DLL) included with the Microsoft Windows operating system.

The following common control classes are supported by APL2:

- Date
- Listview
- Month
- Progress bar
- Tab
- Time
- Up-Down (used in Spin buttons)
- Treeview

The common control library has been updated several times since it was first introduced. Updates for the common control library in older Windows operating systems are distributed with Internet Explorer. Updates for newer operating systems are distributed with service packs. If you are unable to create a common control on a particular system, the common control library may need to be updated; install the latest version of Internet Explorer or operating system service pack.

## ***Appendix F: GDI+ Graphics Device Interface Plus***

GDI+ is a library of graphics routines that is included in all current versions of Windows. Although GDI+ is not included in older versions of Windows, it is available for free from Microsoft.

Several AP 145 properties use GDI+ to display images. If you are unable to set an image property for a control, you may need to install GDI+. To install GDI+, go to [www.microsoft.com](http://www.microsoft.com), search for GDI+, and follow the download and installation instructions.