This function displays the About Ami Pro dialog box. Choosing this function is equivalent to choosing Help/About Ami Pro.

Syntax

About()

Return Value

This function does not return a value.

Example

```
FUNCTION Example()
About()    'This brings up the About Ami Pro dialog box
END FUNCTION
```

See also:

EnhancementProducts Help HowDoIHelp KeyboardHelp MacroHelp UpgradeHelp UsingHelp

This function causes the application used in the argument to become the active application in Windows. Choosing this function is equivalent to choosing the application from the Windows 3.x Task List. The application must be started before this command is used.

Though control passes to the named application, Ami Pro regains control if the macro continues with additional statements beyond this statement.

Syntax

ActivateApp(App)

App is the name of the application that you want to find. The name you use must appear as it does in the Task List window or as it does in the application's title bar.

Return Value

1 (TRUE) if the application was activated.

0 (FALSE) if control could not be passed to the application or if the application was incorrectly named.

Example

```
FUNCTION Example()
'Notify the user what we are doing
StatusBarMsg("Activating Program Manager...")
'Check to see if the app is running
IF AppIsRunning("Program Manager")
    'Activate it if it is
    ActivateApp("Program Manager")
    'Restore it to its previous size
    AppRestore("Program Manager")
ELSE
    'Run it if it is not (it becomes the active app)
    Exec("PROGMAN.EXE", 1)
ENDIF
StatusBarMsg("")   'Clear the status bar
END FUNCTION
```

See also:

AppGetAppCount AppGetAppNames AppHide AppIsRunning AppMaximize AppMinimize AppRestore AppSize Exec

This function creates a new menu bar that can be filled with pull down menus. It must be called prior to adding any pull down menus to this menu bar.

Syntax

AddBar()

Return Value

A positive number if the new menu bar was created.

0 (FALSE) if the new menu bar was not created.

Example

```
FUNCTION Example()
BarID = AddBar() ' add a new menu bar
IF BarID < 1 ' make sure we were able to add a bar
    Message("Unable to add a menu bar.")
    EXIT FUNCTION
ENDIF
ShowBar(BarID)  ' display the new bar
AddMenu(BarID, "Letters") ' add some menus to the new bar
AddMenu(BarID, "Memos")
AddMenu(BarID, "Reports")
ONCANCEL reset
' let the user examine the new bar
UserControl("Resume to restore menus")
reset:
ShowBar(1) ' reset the default bar
END FUNCTION
```

See also:

AddMenu AddMenuItem AddMenuItemDDE ChangeMenuAction CheckMenuItem DeleteMenu DeleteMenuItem GrayMenuItem RenameMenuItem Showbar AddCascadeMenu AddCascadeMenuItem ChangeCascadeAction InsertMenu InsertMenuItem InsertCascadeMenu InsertCascadeMenuItem

This function adds a cascading menu to the end of an existing pull down menu.

Syntax

AddCascadeMenu(BarID, Menu, CascadeMenu)

BarID is the identification number of the menu bar returned from the AddBar function. To use the ami pro menu bar, use 1.

Menu is the name of the pull down menu this cascade menu should be added to. This must match exactly the name of the pull down menu you want to add to, including any ampersand (&) characters. An ampersand is placed before a character that has an underline.

CascadeMenu is the name of the Cascade menu you want to add. Placing an ampersand (&) in front of a character makes that character appear underlined and makes that character a shortcut key.

Return Value

1 (TRUE) if the cascading menu was created.

0 (FALSE) if the cascading menu was not created.

Example

```
FUNCTION Example()
' Add a new cascade to the page menu
Mac = GetRunningMacroFile$()  ' Name of running macro
Menu = "&Page"
Cascade = "New &Cascade"
DeleteMenuItem(1,Menu,Cascade)    ' We don't want multiple cascades
AddCascadeMenu(1, Menu,Cascade)
AddCascadeMenuItem(1, Menu, Cascade, "Item1","{Mac}!Item1()","Help for item1")
AddCascadeMenuItem(1, Menu, Cascade, "My File New",New,"")
AddCascadeMenuItem(1, Menu, Cascade, "My Copy",Copy,"")
End Function

Function Item1()
Message("Nice menu!")
End Function
```

See also:

AddMenu AddBar, AddMenuItem AddMenuItemDDE ChangeMenuAction AddCascadeMenuItem InsertMenu InsertMenuItem InsertCascadeMenu InsertCascadeMenuItem RenameMenuItem GrayMenuItem CheckMenuItem DeleteMenu DeleteMenuItem ShowBar

This function adds a menu item to an existing cascading menu.

Syntax

AddCascadeMenuItem(BarID, Menu, CascadeMenu, Item, MacroName[!Function[(parm1[, parm2...])]], Help)

BarID is the identification number of the menu bar returned from the AddBar function. To use the Ami Pro menu bar, use 1.

Menu is the name of the pull down menu this cascade menu should be added to. This must match exactly the name of the pull down menu you want to add to, including any ampersand (&) characters. An ampersand is placed before a character that has an underline.

CascadeMenu is the name of the cascade menu you want to add. Placing an ampersand (&) in front of a character makes that character appear underlined and makes that character a shortcut key.

Item is the name of the menu item to add. Placing an ampersand (&) in front of a character makes that character appear underlined and makes that character a shortcut key. If the menu item has a shortcut key, you must press **TAB**, type a ^, and then type the appropriate letter as part of the item name.

MacroName is the name of the macro to run if this menu item is selected. This parameter may contain the macro file name, the function within that file to call, and any parameters that function may require. At a minimum, this parameter must contain the macro file name.

Help is the one-line Help text that appears in the title bar of Ami Pro when this menu item is highlighted. It is not optional for this function.

Return Value

1 (true) if the new menu bar was created.

0 (UserCancel/FALSE) if the new menu bar was not created or if the user canceled the function.

Example

```
FUNCTION Example()
' Add a new cascade to the page menu
Mac = GetRunningMacroFile$()  ' Name of running macro
Menu = "&Page"
Cascade = "New &Cascade"
DeleteMenuItem(1,Menu,Cascade)    ' We don't want multiple cascades
AddCascadeMenu(1, Menu,Cascade)
AddCascadeMenuItem(1, Menu, Cascade, "Item1","{Mac}!Item1()","Help for item1")
AddCascadeMenuItem(1, Menu, Cascade, "My File New",New,"")
AddCascadeMenuItem(1, Menu, Cascade, "My Copy",Copy,"")
End Function

Function Item1()
Message("Nice menu!")
End Function
```

See also:

AddMenu AddBar AddMenuItem AddMenuItemDDE ChangeMenuAction AddCascadeMenu InsertMenu InsertMenuItem InsertCascadeMenu InsertCascadeMenuItem RenameMenuItem GrayMenuItem CheckMenuItem DeleteMenu DeleteMenuItem ShowBar

This function automatically creates a frame. All dimensions of the frame must be entered in twips (1 inch = 1440 twips), with the 0,0 location considered to be the top left corner of the page. Choosing this function is equivalent to choosing Frame/Create Frame.

You must be in Layout Mode to call this function.

Syntax

AddFrame(X1, Y1, X2, Y2)

X1 is the horizontal position of the top left corner of the frame.

Y1 is the vertical position of the top left corner of the frame.

X2 is the horizontal position of the bottom right corner of the frame.

Y2 is the vertical position of the bottom right corner of the frame.

The two X coordinates should either be zero or positive numbers. The two Y coordinates should either be zero or negative numbers.

To display the Add a Frame dialog box and allow the user to select the coordinates or select manual draw: **AddFrameDLG**

Return Value

1 (TRUE) if the frame was created.

-2 (GeneralFailure) if the frame was not created.

Example

```
FUNCTION Example()
Mode = GetMode()       'What mode is the screen in?
IF Mode != TRUE
    LayoutMode()       'If not Layout Mode, make it so.
ENDIF
X1 = 3000          ' horizontal - top left
Y1 = -3000         ' vertical - top right
X2 = 7000          ' horizontal - bottom right
Y2 = -7000         ' vertical - bottom right
AddFrame(X1, Y1, X2, Y2)      ' create frame
END FUNCTION
```

See also:

FrameLayout CursorPosition GraphicsScaling FrameModInit AddFrameDLG ImportPicture FrameModLines FrameModType FrameModBorders FrameModFinish GroupFrames IsFrameSelected SelectFrameByName SetFrameDefaults

This function displays the Create Frame dialog box. Choosing this function is equivalent to choosing Frame/Create Frame.

This function does not automatically create a frame. To create a frame automatically, refer to the AddFrame function.

You must be in Layout Mode to call this function.

Syntax

AddFrameDLG

Return Value

1 (TRUE) if the frame was created.

0 (UserCancel) if the user canceled the function.

Example

```
FUNCTION Example()
Mode = GetMode()   'What mode is the screen in?
IF Mode != 1
    LayoutMode()   'If not Layout Mode, make it so.
ENDIF
AddFrameDLG        'Display Add Frame dialog box.
END FUNCTION
```

See also:

FrameLayout CursorPosition GraphicsScaling FrameModInit AddFrame ImportPicture FrameModLines FrameModType FrameModBorders FrameModFinish GroupFrames IsFrameSelected SelectFrameByName SetFrameDefaults

This function creates a new menu on an existing menu bar. The new menu is added to the right of the last menu on the menu bar.

Syntax

AddMenu(BarID, Menu)

BarID is the identification number of the menu bar returned from the AddBar function. To use the Ami Pro menu bar, use 1.

Menu is the name of the pull down menu that should be added. Placing an ampersand (&) character before a character makes that character appear underlined and makes that character a shortcut key.

Return Value

1 (TRUE) if the menu was added.

0 (FALSE) if the menu was not added to the menu bar, or if an invalid BarID was used.

Example

```
FUNCTION Example()
thisfile = getrunningmacrofile$()
AddBar()       ' create the menu bar
addmenu(1,"&New Menu")    ' create a new menu
addmenuitem(1,"&New Menu","Menu Item &1","{thisfile}!message1()","Display Message 1")
addmenuitem(1,"&New Menu","Menu Item &2","{thisfile}!message2()","Display Message 2")
' create a cascade menu
addcascademenu(1,"&New Menu", "Cascade Menu &1")
addcascademenuitem(1,"&New Menu","Cascade Menu &1","Cascade Item &1", "{thisfile}!
message3()","Display Message 3")
addcascademenuitem(1,"&New Menu","Cascade Menu &1","Cascade Item &2", "{thisfile}!
message4()","Display Message 4")
end function

function message1()
message("You selected Menu Item 1.")
end function
function message2()
message("You selected Menu Item 2.")
end function
function message3()
message("You selected Cascade Item 1.")
end function
function message4()
message("You selected Cascade Item 2.")
END FUNCTION
```

See also:

AddBar AddMenuItem AddMenuItemDDE ChangeMenuAction CheckMenuItem DeleteMenu DeleteMenuItem GrayMenuItem RenameMenuItem ShowBar AddCascadeMenu ChangeCascadeAction InsertMenu InsertMenuItem InsertCascadeMenu InsertCascadeMenuItem AddCascadeMenuItem

This function adds a new menu item to the bottom of an existing pull down menu or adds an item directly to an existing menu bar.

Syntax

AddMenuItem(BarID, Menu, Item, MacroName[!Function[(parm1[, parm2...])]][, Help])

BarID is the identification number of the menu bar returned from the AddBar function. To use the Ami Pro menu bar, use 1.

Menu is the name of the pull down menu to which this item should be added. This name must match the name of the pull down menu, including any ampersand (&) characters. An ampersand is placed before a character that has an underline.

Item is the name of the menu item to add. Placing an ampersand (&) in front of a character makes that character appear underlined and makes that character a shortcut key. If the menu item has a shortcut key, you must press TAB, type a ^, and then type the appropriate letter as part of the item name.

MacroName is the name of the macro to run if this menu item is selected. This parameter may contain the macro file name, the function within that file to call, and any parameters that function may require. At a minimum, this parameter must contain the macro file name. If you want to run an Ami Pro function, use its name without quotes or parentheses.

Help is the optional one-line Help text that appears in the title bar of Ami Pro when this menu item is highlighted.

If you want the item on the menu bar: **AddMenuItem**

Return Value

This function returns:

1 (TRUE) if the new menu item was added.

0 (FALSE) if the new menu item was not added, or if an invalid BarID or MenuName was used.

Example

```
FUNCTION Example()
thisfile = getrunningmacrofile$()
AddBar()       ' create the menu bar
addmenu(1,"&New Menu")     ' create a new menu
addmenuitem(1,"&New Menu","Menu Item &1","{thisfile}!message1()","Display Message 1")
addmenuitem(1,"&New Menu","Menu Item &2","{thisfile}!message2()","Display Message 2")
' create a cascade menu
addcascademenu(1,"&New Menu", "Cascade Menu &1")
addcascademenuitem(1,"&New Menu","Cascade Menu &1","Cascade Item &1", "{thisfile}!
message3()","Display Message 3")
addcascademenuitem(1,"&New Menu","Cascade Menu &1","Cascade Item &2", "{thisfile}!
message4()","Display Message 4")
end function

function message1()
message("You selected Menu Item 1.")
end function
function message2()
message("You selected Menu Item 2.")
end function
function message3()
message("You selected Cascade Item 1.")
end function
function message4()
message("You selected Cascade Item 2.")
```

```
END FUNCTION
```

See also:

AddBar AddMenu AddMenuItemDDE ChangeMenuAction CheckMenuItem DeleteMenu DeleteMenuItem GetMacPath GrayMenuItem RenameMenuItem ShowBar AddCascadeMenuItem AddCascadeMenu ChangeCascadeAction InsertMenu InsertMenuItem InsertCascadeMenu InsertCascadeMenuItem

This function creates a new menu item in a pull down menu. This function allows an external program to be notified when the user selects this menu item. An external application must initiate a conversation with the "AmiPro!RemoteControl" Dynamic Data Exchange (DDE) channel, and request an Advise for Menus. When the user selects this menu item, the AdviseID is sent to the application as data in CF_Text format.

Syntax

AddMenuItemDDE(BarID, Menu, Item, AdviseID[, Help])

BarID is the identification number of the menu bar returned from the AddBar function. To use the Ami Pro menu bar, use 1.

Menu is the name of the pull down menu to which this menu should be added. This name must match the name of the pull down menu, including any ampersand (&) characters. An ampersand is placed before a character that has an underline.

Item is the name of the menu item to add. Placing an ampersand (&) in front of a character makes that character appear underlined and makes that character a shortcut key. If the menu item has a shortcut key, you must press TAB, type a ^, and then type the appropriate letter as part of the item name.

AdviseID is an integer that should be sent to an external DDE application if this menu item is chosen by the user.

Help is the one-line Help text that appears in the title bar of Ami Pro when this menu item is highlighted.

Return Value

1 (TRUE) if the new menu item was added.

0 (FALSE) if the new menu item was not added, or if an invalid BarID or Menu name was used.

Example

```
FUNCTION Example()
' This example launches another session of Ami Pro then, via dde, adds a menu and menu items to
the new Ami Pro session.

' Next, you should tile both windows.   Then when you click on the dde menus in the remote
application, the host application displays the result.

' In this example, the result is changing the channel on the tv.

Exec("amipro", "/L") ' Launch another session of Ami Pro
id = DDEInitiate("AmiPro", "System") ' start a dde conversation
DDEExecute(id, "[[DeleteMenu(1, ""Remote Control"")]") ' delete and add menus
DDEExecute(id, "[[AddMenu(1, ""Remote Control"")]")
DDEExecute(id, "[[AddMenuItemDDE(1, ""Remote Control"", ""Channel &1"", 1)]") ' add some menu
items
DDEExecute(id, "[[AddMenuItemDDE(1, ""Remote Control"", ""Channel &2"", 2)]")
DDEExecute(id, "[[AddMenuItemDDE(1, ""Remote Control"", ""Channel &3"", 3)]")
DDEExecute(id, "[[AddMenuItem(1, ""Remote Control"", ""-"", """")]")
DDEExecute(id, "[[AddMenuItemDDE(1, ""Remote Control"", ""Turn off TV"", 4)]")
DDETerminate(id) ' end dde session to system
Message("Tile both Ami Pro windows then try the DDE Remote Control Menus.")

RMac = GetRunningMacroFile$() ' what macro name is running?
id = DDEInitiate("AmiPro", "RemoteControl") ' start dde conversation to remote control
AllocGlobalVar("id", 1) ' create global variable for channel id
SetGlobalVar("id", id) ' store channel id in the variable
DDEAdvise(id, "Menus", "{Rmac}!Channel") ' set up to run a macro when the user clicks on our dde
menus in the other Ami Pro
END FUNCTION
```

```
Function Channel(value)
switch value  ' get the dde menu id and do the right thing
    case 1
        Type("Channel {value} is showing Wheel of Macros[enter]")
    case 2
        Type("Channel {value} is showing Gilligan's Macros[enter]")
    case 3
        Type("Channel {value} is showing The Little Macros[enter]")
    case 4
        Message("Turning off TV") ' they want to end the dde session
        id = GetGlobalVar$("id")
        DDEUnAdvise(id, "Menus") ' no longer want to be advised
        DDETerminate(id) ' terminate the dde conversation
        FreeGlobalVar("id") ' free the variable
        Message("Good Night") ' say good night Ami
endswitch
```

END FUNCTION

See also:

AddBar AddMenu AddMenuItem ChangeMenuAction CheckMenuItem DeleteMenu DeleteMenuItem DDEAdvise DDEPoke DDEExecute DDEInitiate DDEReceive$ DDETerminate DDEUnAdvise GrayMenuItem RenameMenuItem ShowBar AddCascadeMenu AddCascadeMenuItem InsertMenu InsertMenuItem InsertCascadeMenu InsertCascadeMenuItem Exec

This function allocates memory to hold a global variable (single element or array). Global variables retain their values until the user frees them or exits Ami Pro, as opposed to regular variables, which are lost once the macro is finished.

Syntax

AllocGlobalVar(Name, Count)

Name is a unique number or string naming the variable being allocated.

Count is the number of entries to allocate. A count of one allocates a single element global variable. A count of more than one allocates an array with count elements.

Return Value

The Name if the variable allocates memory.

0 (FALSE) if there is no free memory to allocate.

Example

```
FUNCTION Example()
varname = Query$("What do you wish to name the global variable?")
AllocGlobalVar("{varname}", 1)
Message("The variable {varname} has been allocated room in memory to hold one entry.")
varvalue = Query$("Enter some data to put in this global variable called {varname}.")
SetGlobalVar("{varname}", "{varvalue}")
Message("Ok, we will now go and get the value of {varname} and display it.")
Message(GetGlobalVar$("{varname}"))
END FUNCTION
```

See also:

FreeGlobalVar GetGlobalArray$ GetGlobalVar$ SetGlobalArray SetGlobalVar Global Variables GetGlobalVarCount GetGlobalVarNames

This function is used to call any Ami Pro function where the number of parameters cannot be determined ahead of time.

Syntax

AmiProIndirect(FunctionName, &Parameters, Count)

FunctionName is the function you want to call.

Parameters are the arrays that contain the parameters.

Count is the number of parameters in the array.

Return Value

1 (TRUE) if the changes were made.

0 (FALSE) if the changes could not be made.

Example

```
FUNCTION Example()
' how many parameters?
ParamCount = GetLayoutParmCnt(ModLayoutRightPage)
' dimension an array for the parameters
DIM LayoutValues(ParamCount)
' get the values
GetLayoutParameters(ModLayoutRightPage, &LayoutValues)
' increase the top margin by 1 inch
LayoutValues(2) = LayoutValues(1) + 1440
' prepare Ami Pro for a layout change
ModLayoutInit(512) ' apply layout
AmiProIndirect(ModLayoutRightPage, &LayoutValues, ParamCount)
ModLayoutFinish() ' let Ami Pro accept the changes
END FUNCTION
```

See also:

GetLayoutParmCnt GetLayoutParameters ModLayoutInit ModLayoutRightPage ModLayoutFinish

This function is used to 'reply' to an Ami Pro function that asks the user for a response. Many functions require the user to provide some response before the function can be completed. The Messages function allows the macro programmer to decide if the user has to respond to messages or if the default action is taken. The AnswerMsgBox function works in a similar manner; however, a response determined by the programmer is taken, rather than the default.

The AnswerMsgBox function should be used immediately before the function which displays the message. Unlike the Messages function, the AnswerMsgBox function provides a response only to the next Ami Pro function called, rather than all succeeding messages.

In general, within a given macro, you should not mix the Messages and AnswerMsgBox functions.

Syntax

AnswerMsgBox(Response)

Response is the desired response to a message box displayed by a function. It must be set to one of the following options:

- OK (1) - Equivalent to choosing OK
- Cancel (2) - Equivalent to choosing Cancel
- Abort (3) - Equivalent to choosing Abort
- Retry (4) - Equivalent to choosing Retry
- Yes (6) - Equivalent to choosing Yes
- No (7) - Equivalent to choosing No

Return Value

This function does not return a value.

Example

```
FUNCTION Example()
New("_default.sty", 0, 0)
Type("Lotus Ami Pro for Windows")
Message("New file created. Press OK to close without saving.")
AnswerMsgBox(7)
FileClose()
END FUNCTION
```

See also:

IgnoreKeyboard Messages SingleStep UserControl HourGlass

This function closes any Windows application.

Syntax

AppClose(App)

App is the name of the application which should be closed. The name you use must appear as it does in the Windows Task List or as it does in the application's title bar.

Return Value

1 (TRUE) if the changes were made.

0 (FALSE) if the changes could not be made.

Example

```
FUNCTION Example()
ControlPanel()
Message("Click OK to close the Control Panel.")
AppClose("Control Panel")
END FUNCTION
```

See also:

ActivateApp AppGetAppCount AppGetAppNames AppGetWindowPos AppHide AppIsRunning AppMaximize AppMinimize AppMove AppRestore AppSendMessage AppSize

This function allows you to get the number of currently running Windows applications. It is usually used to dimension an array before the AppGetAppNames function is called.

Syntax

AppGetAppCount()

Return Value

The number of running Windows applications.

Example

```
FUNCTION Example()
numapps = AppGetAppCount()    'return the number of apps
Message("Currently, there are {numapps} applications running, whether hidden or visible.")
DIM names(numapps)        'dimension a name array
AppGetAppNames(&names)    'get the name for the apps
FOR I = 1 to numapps      'show the name for each app
    app = names(I)
    Message(app)
NEXT
END FUNCTION
```

See also:

ActivateAppActivateApp AppClose AppGetAppNames AppGetWindowPos AppHide AppIsRunning AppMaximize AppMinimize AppMove AppRestore AppSendMessage AppSize

This function retrieves the names of all of the currently running Windows applications and places them in an array. The array may be dimensioned using the AppGetAppCount function.

Syntax

AppGetAppNames(&Array)

Array is the name of an array, dimensioned from the return value of the AppGetAppCount function. Note the use of indirection (&).

Return Value

The number of applications.

Example

```
FUNCTION Example()
numapps = AppGetAppCount()    'return the number of apps
Message("Currently, there are {numapps} applications running, whether hidden or visible.")
DIM names(numapps)         'dimension a name array
AppGetAppNames(&names)     'get the name for the apps
FOR I = 1 to numapps       'show the name for each app
    app = names(I)
    Message(app)
NEXT
END FUNCTION
```

See also:

ActivateApp AppClose AppGetAppCount AppGetWindowPos AppHide AppIsRunning AppMaximize AppMinimize AppMove AppRestore AppSendMessage AppSize

This function returns the location and size information about any Windows application. Note the use of indirection (&). The X, Y, Width, and Height parameters are all passed by addressing. The return values are in percentages of the screen size.

Syntax

AppGetWindowPos(App, &X, &Y, &Width, &Height)

App is the name of the application that you want to find. The name you use must appear as it does in the Windows Task List or as it does in the application's title bar.

X is the horizontal starting point of App.

Y is the vertical starting point of App.

Width is the width of App.

Height is the height of App.

Return Value

1 (TRUE) if the application was valid.

0 (FALSE) if the application was not valid.

Example

```
FUNCTION Example()
DEFSTR x, y, x2, y2;
ControlPanel()     'run the control panel
' get the position of the control panel
AppGetWindowPos("Control Panel", &x, &y, &x2, &y2)
x = Query$("How far from the left do you want Control Panel to be? (10 is about an inch)", x)
y = Query$("How far from the top do you want Control Panel to be? (10 is about an inch)", y)
x2 = Query$("In screen percentage, how wide do you want Control Panel?", x2)
y2 = Query$("In screen percentage, how tall do you want Control Panel?", y2)
AppMove("Control Panel", x, y)        'move the control panel
AppSize("Control Panel", x2, y2)      'size the control panel
END FUNCTION
```

See also:

ActivateApp AppClose AppGetAppCount AppGetAppNames AppHide AppIsRunning AppMaximize AppMinimize AppMove AppRestore AppSendMessage AppSize

This function hides any Windows application. When an application is hidden it cannot receive keyboard or mouse input; however, it responds to Dynamic Data Exchange (DDE) commands.

Syntax

AppHide(App)

App is the name of the application that should be hidden. The name you use must appear as it does in the Windows Task List or as it does in the application's title bar.

Return Value

1 (TRUE) if the application was hidden.

0 (FALSE) if the application could not be hidden.

Example

```
FUNCTION Example()
ControlPanel()
Message("Control Panel is active. We will now hide it.")
AppHide("Control Panel")
Message("Control Panel is now hidden. Click on OK to bring it back.")
AppRestore("Control Panel")
END FUNCTION
```

See also:

ActivateApp AppClose AppGetAppCount AppGetAppNames AppGetWindowPos AppIsRunning AppMaximize AppMinimize AppMove AppRestore AppSendMessage AppSize

This function allows you to see if any Windows application is currently running.

Syntax

AppIsRunning(App)

App is the name of the application to be evaluated. The name you use must appear as it does in the Windows Task List or as it does in the application's title bar.

Return Value

1 (TRUE) if the application is running.

0 (FALSE) if the application is not running.

Example

```
FUNCTION Example()
cp = AppIsRunning("Control Panel")
IF cp = 1
    Message("Control Panel is running.")
ELSEIF cp != 1
    Message("Control Panel is not running. Click on OK to execute it.")
    ControlPanel()
ENDIF
END FUNCTION
```

See also:

ActivateApp AppClose AppGetAppCount AppGetAppNames AppGetWindowPos AppMaximize AppMinimize AppMove AppRestore AppSendMessage AppSize AppHide

This function applies text formatting and enhancements to selected text. The text formatting and enhancements must be loaded using the FastFormat function. A macro must be edited to insert this non-recordable function.

Syntax

ApplyFormat()

Return Value

This function returns 1.

Example

```
FUNCTION Example()
UserControl("Shade the area that contains the formatting you want:")
FastFormat()     ' Turn on Fast Formatting
FastFormat()     ' Turn off Fast Formatting
UserControl("Shade the area you want to apply formatting to:")
ApplyFormat()
END FUNCTION
```

See also:

FastFormat

This function maximizes any Windows application.

Syntax

AppMaximize(App)

App is the name of the application that is to be maximized. The name you use must appear as it does in the Windows Task List or as it does in the application's title bar.

Return Value

1 (TRUE) if the application was maximized.

0 (FALSE) if the application could not be maximized.

Example

```
FUNCTION Example()
Exec("winfile.exe", "", 1)
AppMaximize("File Manager")
Message("File Manager is maximized.")
AppRestore("File Manager")
Message("File Manager is windowed.")
AppMinimize("File Manager")
Message("File Manager is an icon.")
ActivateApp("File Manager")
Message("Click OK to close File Manager.")
AppClose("File Manager")
END FUNCTION
```

See also:

ActivateApp AppClose AppGetAppCount AppGetAppNames AppGetWindowPos AppIsRunning AppMinimize AppMove AppRestore AppSendMessage AppSize AppHide

This function minimizes any Windows application.

Syntax

AppMinimize(App)

App is the name of the application that is to be minimized. The name you use must appear as it does in the Windows Task List or as it does in the application's title bar.

Return Value

1 (TRUE) if the application was minimized.

0 (FALSE) if the application could not be minimized.

Example

```
FUNCTION Example()
Exec("winfile.exe", "", 1)
AppMaximize("File Manager")
Message("File Manager is maximized.")
AppRestore("File Manager")
Message("File Manager is windowed.")
AppMinimize("File Manager")
Message("File Manager is an icon.")
ActivateApp("File Manager")
Message("Click OK to close File Manager.")
AppClose("File Manager")
END FUNCTION
```

See also:

ActivateApp AppClose AppGetAppCount AppGetAppNames AppGetWindowPos AppIsRunning AppMaximize AppMove AppRestore AppSendMessage AppSize AppHide

This function moves a Windows application. It does not size the window, but moves it to a new location on the screen.

Syntax

AppMove(App, X, Y)

App is the name of the application that is to be moved. The name you use must appear as it does in the Windows Task List or as it does in the application's title bar. To identify Ami Pro as the application to move, use the null string ("").

X is the horizontal location on the screen, in percent.

Y is the vertical location on the screen, in percent.

(0,0) is the upper left position on the screen.

Return Value

1 (TRUE) if the application was moved.

0 (FALSE) if the application could not be moved.

Example

```
FUNCTION Example()
DEFSTR x, y, x2, y2;
ControlPanel()     'run the control panel
' get the position of the control panel
AppGetWindowPos("Control Panel", &x, &y, &x2, &y2)
x = Query$("How far from the left do you want Control Panel to be? (10 is about an inch)", x)
y = Query$("How far from the top do you want Control Panel to be? (10 is about an inch)", y)
x2 = Query$("In screen percentage, how wide do you want Control Panel?", x2)
y2 = Query$("In screen percentage, how tall do you want Control Panel?", y2)
AppMove("Control Panel", x, y)       'move the control panel
AppSize("Control Panel", x2, y2)     'size the control panel
END FUNCTION
```

See also:

"ActivateApp AppClose AppGetAppCount AppGetAppNames AppGetWindowPos AppIsRunning AppMaximize AppMinimize AppRestore AppSendMessage AppSize AppHide

This function restores any Windows application to its former non-maximized or non-minimized size.

Syntax

AppRestore(App)

App is the name of the application that is to be restored. The name you use must appear as it does in the Windows Task List or as it does in the application's title bar.

Return Value

1 (TRUE) if the application was restored.

0 (FALSE) if the application could not be restored.

Example

```
FUNCTION Example()
Exec("winfile.exe", "", 1)
AppMaximize("File Manager")
Message("File Manager is maximized.")
AppRestore("File Manager")
Message("File Manager is windowed.")
AppMinimize("File Manager")
Message("File Manager is an icon.")
ActivateApp("File Manager")
Message("Click OK to close File Manager.")
AppClose("File Manager")
END FUNCTION
```

See also:

ActivateApp AppClose AppGetAppCount AppGetAppNames AppGetWindowPos AppIsRunning AppMaximize AppMinimize AppMove AppSendMessage AppSize AppHide

This function allows you to send any Windows message to any Windows application. You should be familiar with the Message, Wparam, and Lparam parameters. This function first locates the application's handle and then calls the SendMessage function from Windows with the specified parameters.

Syntax

AppSendMessage(App, Message, Wparam, Lparam)

App is the name of the application to send the message to. It can also be a window handle.

Message is the Windows message to send to the application.

Wparam is the wparam parameter for the Windows message.

Lparam is the lparam parameter for the Windows message.

Return Value

The return value from the Windows function SendMessage.

Example

```
DEFINE Em_SetPassWord  1052
DEFINE Asterisk        42

FUNCTION Example()
' get the name of this macro file
Mac = GetRunningMacrofile$()
' this is called before the box is displayed
SetDlgCallBack(0,"{Mac}!InitDlg")
Box = DialogBox(".","Pass")
pword = GetDialogField$(8000)
Message("Password was {pword}")
END FUNCTION

FUNCTION InitDlg(Hdlg, id, value)
' Get the handle to the edit box 8000
hEditBox = GetDlgItem(Hdlg, 8000)
' Password protect edit box
AppSendMessage(hEditbox,Em_SetPassWord,Asterisk,0)
END FUNCTION

DIALOG Pass
-2134376448 2 80 51 160 52 "" "" "Password Example"
FONT 8 "Helv"
6 22 100 12 8000 1350631552 "edit" "" 0
116 4 40 14 1 1342242817 "button" "OK" 0
END DIALOG
```

See also:

SetDlgCallback ActivateApp AppClose AppGetAppCount AppGetAppNames AppGetWindowPos AppIsRunning AppMaximize AppMinimize AppMove AppRestore AppSize AppHide

This function changes the size of any Window application.

Syntax

AppSize(App, Width, Height)

App is the name of the application that is to be sized. The name you use must appear as it does in the Windows Task List or as it does in the application's title bar. Using null ("") will cause AppSize to affect the current Ami Pro window.

Width is the new width, as a percentage of screen size, of the App.

Height is the new height, as a percentage of screen size, of the App.

Return Value

1 (TRUE) if the application was successfully sized.

0 (FALSE) if the application could not be sized.

Example

```
FUNCTION Example()
DEFSTR x, y, x2, y2;
ControlPanel()     'run the control panel
' get the position of the control panel
AppGetWindowPos("Control Panel", &x, &y, &x2, &y2)
x = Query$("How far from the left do you want Control Panel to be? (10 is about an inch)", x)
y = Query$("How far from the top do you want Control Panel to be? (10 is about an inch)", y)
x2 = Query$("In screen percentage, how wide do you want Control Panel?", x2)
y2 = Query$("In screen percentage, how tall do you want Control Panel?", y2)
AppMove("Control Panel", x, y)       'move the control panel
AppSize("Control Panel", x2, y2)     'size the control panel
END FUNCTION
```

See also:

ActivateApp AppClose AppGetAppCount AppHide AppGetAppNames AppGetWindowPos AppIsRunning AppMaximize AppMinimize AppMove AppRestore AppSendMessage

This function deletes the specified record from an existing array.
This function does not re-dimension the array.

Syntax

ArrayDelete(&Array, Index)

Array is the name of the array from which to delete a record. Note the use of indirection (&).

Index is the location in the array to delete. This parameter must be a valid array index.

Return Value

1 (TRUE) if the record could be deleted.

0 (FALSE) if the record could not be deleted.

Example

```
FUNCTION Example()
as = Query$("How many records do you want this array to hold?")
DIM Names(as)      ' dimension the name array
FOR I = 1 to as
    ThisName = Query$("What is this person's Name?")
    IF ThisName != ""  'if not blank, insert the name
        ArrayInsert(&Names, 1, ThisName)
    ENDIF
NEXT
Size = ArraySize(&Names)  'return the number of names
Message("Number of records is {Size}.")
ArraySort(&Names)      'sort the names
Again:
' Find the location of the name
Ndex = ArraySearch(&Names, Query$("What name to find?"))
IF Ndex != 0
    IF Decide("Do you want to delete record {Ndex}?")
        ' delete the record at that index
        ArrayDelete(&Names, Ndex)
        ' shrink the size of the array
        Size = Size - 1
    ENDIF
ELSE
    Message("Could not find record.")
    GoTo again
ENDIF
FOR I = 1 to Size
    ThisName = Names(I)
    Message("Record Number {I} of {Size} is {ThisName}.")
NEXT
END FUNCTION
```

See also:

DIM ArrayInsert ArrayInsertByKey ArraySearch ArraySize ArraySort

This function inserts the specified record into the array at the specified index. It "moves" existing records to make room for the new record. If the array is full, the function expands the array.

Syntax

ArrayInsert(&Array, Index, Record)

Array is the name of the array in which to insert the specified record. Note the use of indirection (&).

Index is the position in the array to insert the specified record. Index must be a valid array index or one greater than the full array. If the array index is greater than the current size of the array by one, the record appends to the end of the array.

Record is the data to insert into the array.

The name of the array uses indirection (&).

Return Value

1 (TRUE) if the record was inserted.

0 (FALSE) if the record could not be inserted.

Example

```
FUNCTION Example()
as = Query$("How many records do you want this array to hold?")
DIM Names(as)      ' dimension the name array
FOR I = 1 to as
    ThisName = Query$("What is this person's Name?")
    IF ThisName != ""  'if not blank, insert the name
        ArrayInsert(&Names, 1, ThisName)
    ENDIF
NEXT
Size = ArraySize(&Names)  'return the number of names
Message("Number of records is {Size}.")
ArraySort(&Names)      'sort the names
Again:
' Find the location of the name
Ndex = ArraySearch(&Names, Query$("What name to find?"))
IF Ndex != 0
    IF Decide("Do you want to delete record {Ndex}?")
        ' delete the record at that index
        ArrayDelete(&Names, Ndex)
        ' shrink the size of the array
        Size = Size - 1
    ENDIF
ELSE
    Message("Could not find record.")
    GoTo again
ENDIF
FOR I = 1 to Size
    ThisName = Names(I)
    Message("Record Number {I} of {Size} is {ThisName}.")
NEXT
END FUNCTION
```

See also:

DIM ArrayDelete ArrayInsertByKey ArraySearch ArraySize ArraySort

This function inserts the record based on the assumption that the array is sorted. The array can be a single element array or field delimited records. If the array is full, the function expands the array.

Syntax

ArrayInsertByKey(&Array, Record, DuplicatesOk[, FieldNumber, Delimiter])

Array is the name of the array in which to insert the specified record. Note the use of indirection (&).

Record is the data to insert into the array.

DuplicatesOk determines if this record should be inserted if its key is already in the array. If the DuplicatesOk parameter is TRUE and this record has a duplicate key, the record is inserted after all other records with the same key.

FieldNumber is the optional field number when using field delimited records. This defines the Key field.

Delimiter is the optional delimiter in field delimited records.

Return Value

A positive number which is the index of the inserted record.

0 (FALSE) if the record was not inserted.

Example

```
FUNCTION Example()
as = Query$("How many records do you want this array to hold?")
DIM Names(as)      'dimension the name array
FOR I = 1 to as
    ThisName = Query$("What is this one's name?")
    ArrayInsertByKey(&Names, ThisName, TRUE)
NEXT
Size = ArraySize(&Names)
FOR I = 1 to Size
    ThisName = Names(I)
    Message("Record #{I} of {Size} is {ThisName}.")
NEXT
END FUNCTION
```

See also:

DIM ArrayInsert ArrayDelete ArraySearch ArraySize ArraySort

This function searches the specified array looking for a key. If a Fieldnumber and a delimiter are specified, the function assumes the records are field delimited and matches only on the specified field. If duplicate matches occur, this function finds the first matching record. If an index is specified, the function begins searching at that position and returns the next matching index of the record.

Syntax

ArraySearch(&Array, Key[, FieldNumber, Delimiter][, Index])

Array is the name of the array to search. Note the use of indirection (&).

Key is the data to search for.

FieldNumber is the optional field number when using field delimited records. This defines the Key field.

Delimiter is the optional delimiter in field delimited records.

Index is the position in the array to begin the search.

Return Value

A positive number which is the index of the matching record.

0 (FALSE) if the record was not matched.

Example

```
FUNCTION Example()
as = Query$("How many records do you want this array to hold?")
DIM Names(as)      ' dimension the name array
FOR I = 1 to as
    ThisName = Query$("What is this person's Name?")
    IF ThisName != ""  'if not blank, insert the name
        ArrayInsert(&Names, 1, ThisName)
    ENDIF
NEXT
Size = ArraySize(&Names)  'return the number of names
Message("Number of records is {Size}.")
ArraySort(&Names)      'sort the names
Again:
' Find the location of the name
Ndex = ArraySearch(&Names, Query$("What name to find?"))
IF Ndex != 0
    IF Decide("Do you want to delete record {Ndex}?")
        ' delete the record at that index
        ArrayDelete(&Names, Ndex)
        ' shrink the size of the array
        Size = Size - 1
    ENDIF
ELSE
    Message("Could not find record.")
    GoTo again
ENDIF
FOR I = 1 to Size
    ThisName = Names(I)
    Message("Record Number {I} of {Size} is {ThisName}.")
NEXT
END FUNCTION
```

See also:

DIM ArrayInsert ArrayDelete ArrayInsertByKey ArraySize ArraySort

This function returns the number of elements in an existing array.

Syntax

ArraySize(&Array)

Array is the name of the array to be examined. Note the use of indirection (&).

Return Value

The number of dimensioned elements in the array.

Example

```
FUNCTION Example()
as = Query$("How many records do you want this array to hold?")
DIM Names(as)      ' dimension the name array
FOR I = 1 to as
    ThisName = Query$("What is this person's Name?")
    IF ThisName != ""  'if not blank, insert the name
        ArrayInsert(&Names, 1, ThisName)
    ENDIF
NEXT
Size = ArraySize(&Names)  'return the number of names
Message("Number of records is {Size}.")
ArraySort(&Names)      'sort the names
Again:
' Find the location of the name
Ndex = ArraySearch(&Names, Query$("What name to find?"))
IF Ndex != 0
    IF Decide("Do you want to delete record {Ndex}?")
        ' delete the record at that index
        ArrayDelete(&Names, Ndex)
        ' shrink the size of the array
        Size = Size - 1
    ENDIF
ELSE
    Message("Could not find record.")
    GoTo again
ENDIF
FOR I = 1 to Size
    ThisName = Names(I)
    Message("Record Number {I} of {Size} is {ThisName}.")
NEXT
END FUNCTION
```

See also:

DIM ArrayInsert ArrayDelete ArrayInsertByKey ArraySearch ArraySort

This function sorts an existing array in ascending order. If a FieldNumber and D elimiter are specified, the function assumes the records are field delimited and the sort is performed on the specified field. If duplicate keys exist, their order is undefined.

Numbers sort before alphabetical characters.

Syntax

ArraySort(&Array[, FieldNumber, Delimiter])

Array is the name of the array to be sorted. Note the use of indirection (&).

FieldNumber is the optional field number when using field delimited records. This defines the Key field.

Delimiter is the optional delimiter in field delimited records.

Return Value

1 (TRUE) if the array was sorted.

0 (FALSE) if the array could not be sorted.

Example

```
FUNCTION Example()
as = Query$("How many records do you want this array to hold?")
DIM Names(as)      ' dimension the name array
FOR I = 1 to as
    ThisName = Query$("What is this person's Name?")
    IF ThisName != ""  'if not blank, insert the name
        ArrayInsert(&Names, 1, ThisName)
    ENDIF
NEXT
Size = ArraySize(&Names)  'return the number of names
Message("Number of records is {Size}.")
ArraySort(&Names)      'sort the names
Again:
' Find the location of the name
Ndex = ArraySearch(&Names, Query$("What name to find?"))
IF Ndex != 0
    IF Decide("Do you want to delete record {Ndex}?")
        ' delete the record at that index
        ArrayDelete(&Names, Ndex)
        ' shrink the size of the array
        Size = Size - 1
    ENDIF
ELSE
    Message("Could not find record.")
    GoTo again
ENDIF
FOR I = 1 to Size
    ThisName = Names(I)
    Message("Record Number {I} of {Size} is {ThisName}.")
NEXT
END FUNCTION
```

See also:

DIM ArrayInsert ArrayDelete ArrayInsertByKey ArraySearch ArraySize

This function determines the numeric ANSI code of the first letter in the text. If the text contains more than one letter, the remaining letters are ignored.

Syntax

ASC(Text)

Text is a string of one or more letters.

Return Value

A number that is the ANSI character code of the first letter in the text.

Example

```
FUNCTION Example()
String = Query$("What character to return the ASCII Value of?")
Number = ASC(String)
Message("The ASCII equivalent of {String} is {Number}.")
END FUNCTION
```

See also:

CHR$ LCASE$ MID$ strchr strfield$ UCASE$ Right$ Left$ Instr LEN strcat$

This function sets the ASCII options for any opened file that uses the ASCII filter. Choosing this function is equivalent to choosing File/Open, specifying the ASCII file type, and choosing ASCII Options.

Syntax

ASCIIOptions(Options)

Options is the options available to open an ASCII file. It is one or more of the following values:

CRLF (2) - Expect carriage return/line feed characters at end of lines; if not set, expect CR/LFs at end of paragraphs

ASCII (4) - 7 bit ASCII file

PCASCII (8) - 8 bit PC ASCII file

ANSI (16) - 8 bit ANSI file

KeepNames (32) - Keep style names

Return Value

This function returns 1.

Example

```
FUNCTION Example()
ASCIIOptions(PCASCII)  'set the ascii option
WinDir = GetWindowsDirectory$()
FileOpen("{WinDir}PRINTERS.TXT", 16, "ASCII")
END FUNCTION
```

See also:

FileOpen ImportExport SaveAs

This function assigns the result of an expression to a variable and returns the same result. The expression can be anything that you can pass to another function. This function is useful when using IF/THEN and WHILE/WEND statements.

Syntax

Assign(&Var, Express)

Var is the name of the variable to assign the return value from Express to. Note the use of indirection (&).

Express is the expression to evaluate.

Return Value

The return value of Express.

Example

```
FUNCTION Example()
DEFSTR id, Line;
'if the open file can be assigned to id
IF 0 != Assign(&id, fopen("c:\autoexec.bat", "r"))
    New("_DEFAULT.STY", 0, 0) 'new file
    WHILE -1 != Assign(&Line, fgets$(id))
        TYPE("{Line}[Enter]") 'type the autoexec lines
    WEND
    fclose(id)
ELSE
    Message("Could not open C:\AUTOEXEC.BAT")
ENDIF
END FUNCTION
```

See also:

DEFSTR fopen fgets$ fclose

This function assigns an open and close macro to the current file. Choosing this function is equivalent to choosing Tools/Macro/Edit/Assign.

Syntax

AssignMacroToFile(OpenMacro, CloseMacro, Flag)

OpenMacro is the macro name that is specified to run when the current file opens. Use the null string ("") to specify no opening macro.

CloseMacro is the macro name that is specified to run when the current file closes. Use the null string ("") to specify no closing macro.

Flag indicates whether to run the specified open or close macro. Add the values together to run both open and close macros.

- 2 - Run the opening macro
- 4 - Run the closing macro

Return Value

This function returns 0.

Example

```
FUNCTION Example()
' assign the openit macro to run when you open the doc
' assign the closeit macro to run when you close the doc
AssignMacroToFile("Openit.smm", "Closeit.smm", 6)
END FUNCTION
```

See also:

MacroEdit MacroOptions MacroPlay

This function determines if the insertion point is at the end of the document.

Syntax

AtEOF()

Return Value

1 (TRUE) if the insertion point is at the end of the document.

0 (FALSE) if the insertion point is not at the end of the document.

Example

```
FUNCTION Example()
New("_default.sty", 0, 0)
Type("This is the time for all good men to come to the aid of their country.")
Sendkeys("[ctrlhome]")    'go to the beginning of the doc
WHILE not AtEOF()
    Sendkeys("[right]")    'move the cursor right one character
WEND
Message("The insertion point is now at the end of this file.")
END FUNCTION
```

See also:

CurChar$ CurWord$ GetPageNo Type
TopOfFile EndOfFile

This function displays the Ami Pro's online Help. Choosing this function is equivalent to choosing Help/Basics from the menu. This function does not select a help topic automatically. The user can use the Help system in the regular way to get help on the topic. Because Help displays in a window other than the regular Ami Pro window, further macro functions which cause a repainting of the Ami Pro window cause the Help window to be replaced by the Ami Pro window.

If this function is used, it should be the last function used in the macro.

Syntax

BasicsHelp()

Return Value

1 (TRUE) if the Help window displays.

-2 (GeneralFailure) if the Help window could not display for some other reason.

-6 (NoMemory) if the function failed because of insufficient memory.

Example

See also:

About EnhancementProducts Help HowDoIHelp KeyboardHelp MacroHelp UpgradeHelp UsingHelp

This function tells Windows to sound (beep) the current audio device if the sound is turned on in the Control Panel.

Syntax

Beep()

Return Value

This function does not return a value.

Example

```
FUNCTION Example()
numbeeps = Query$("How many beeps do you want to hear?", 50)
FOR I = 1 to numbeeps
    Beep()
NEXT
END FUNCTION
```

See also:

Message StatusBarMsg HourGlass

This function takes binary information and converts it to the Ami Pro brackets convention. This function is useful after using the fread function to see exactly what was read.

Syntax

BinToBrackets(Data)

Data is the binary information you want to convert to brackets.

Return Value

The bracket convention to which the binary information equates.

Example

```
FUNCTION Example()
DEFSTR string;
id = fopen("test.txt", "w")       'open test.txt for write
IF id != 0     ' get the name and address
    Name = Query$("What is your name?")
    fwrite(id, Name)
    ' convert to binary
    fwrite(id, BracketsToBin([Enter]))
    fputs(id, Query$("What is your address?"))
    fclose(id)
ENDIF
id2 = fopen("test.txt", "r")      'open test.txt for read
IF id2 != 0
    NameLength = len(Name)    'get the length of the name
    fseek(id2, 0, 0)
    Name = fread(id2, NameLength)
    ' convert to brackets from binary
    EnterKey = BinToBrackets(fread(id2, 2))
    AddressBegins = ftell(id2)
    Address = fgets$(id2)
    Message("Your name is {Name}.")
    Message("Your address is {Address}.")
    Message("EnterKey code is = {EnterKey}")
    Message("Your address begins at the number {AddressBegins} byte in TEST.TXT.")
    fclose(id2)
ENDIF
END FUNCTION
```

See also:

BracketsToBin fopen fclose fread fwrite fgets$ fputs ASC CHR$

This function sets the bold attribute for selected text or all following text if no text is selected. It acts as a toggle, turning off the attribute if it is currently on and turning on the attribute if it is currently off. Choosing this function is equivalent to choosing Text/Bold. this function toggles the bold attribute on or off, depending on its current state.

Syntax

Bold()

Return Value

0 if the bold attribute is toggled on and there are no attributes assigned to the text.

4 if the bold attribute is toggled off.

8 if the bold attribute is toggled on and the italic attribute is assigned.

16 if the bold attribute is toggled on and the underline attribute is already assigned.

32 if the bold attribute is toggled on and the word underline attribute is already assigned.

-2 (GeneralFailure) if the text was not changed.

The return values may be added together to identify the attributes that were previously assigned.

Example

```
FUNCTION Example()
text = Query$("Enter some text:")
New("_default.sty", 0, 0)     'open a new file
TYPE("{text}")
TYPE("[enter]")
TYPE("[ctrlhome][ctrlshiftend]")
Copy()  'copy text to clipboard
FOR I = 1 to 10
    Paste()     'paste text 10 times
NEXT
Message("The text will be shaded, bolded, italicized, underlined, and centered.")
TYPE("[ctrlhome][ctrlshiftend]")
Bold()
Italic()
Underline()
Center()
TYPE("[ctrlhome]")
END FUNCTION
```

See also:

Italic NormalText Underline WordUnderline

This function converts Ami Pro bracketed values to their binary equivalents. It is useful when you are using the fwrite function.

Syntax

BracketsToBin(Data)

Data is the bracketed information to convert to binary.

Return Value

The binary equivalent to the bracketed information.

Example

```
FUNCTION Example()
DEFSTR string;
id = fopen("test.txt", "w")       'open test.txt for write
IF id != 0     ' get the name and address
    Name = Query$("What is your name?")
    fwrite(id, Name)
    ' convert to binary
    fwrite(id, BracketsToBin([Enter]))
    fputs(id, Query$("What is your address?"))
    fclose(id)
ENDIF
id2 = fopen("test.txt", "r")      'open test.txt for read
IF id2 != 0
    NameLength = len(Name)    'get the length of the name
    fseek(id2, 0, 0)
    Name = fread(id2, NameLength)
    ' convert to brackets from binary
    EnterKey = BinToBrackets(fread(id2, 2))
    AddressBegins = ftell(id2)
    Address = fgets$(id2)
    Message("Your name is {Name}.")
    Message("Your address is {Address}.")
    Message("EnterKey code is = {EnterKey}")
    Message("Your address begins at the number {AddressBegins} byte in TEST.TXT.")
    fclose(id2)
ENDIF
END FUNCTION
```

See also:

BinToBrackets fopen fclose fread fwrite fgets$ fputs ASC CHR$

This function brings the selected frame to the front of a stack of frames for editing. Choosing this function is equivalent to choosing Frame/Bring to Front.

Syntax

BringFrameToFront()

Return Value

1 (TRUE) if the frame was brought to the front.

0 (NoAction) if no action was taken because the frame was already brought to the front.

Example

```
FUNCTION Example()
'get the cursor position
x = strfield$(CursorPosition$(), 1, ",")
y = strfield$(CursorPosition$(), 2, ",")
' add the frames and bookmark them
AddFrame(x, y, (x + 1440), (y - 1440))
MarkBookMark("Frame1", AddBookMark)
AddFrame((x + 360), (y - 360), (x + 1800), (y - 1800))
MarkBookMark("Frame2", AddBookMark)
SendFrameToBack()  ' send frame to back
MarkBookMark("Frame1", FindBookMark)
Message("The First frame is in front.")
MarkBookMark("Frame2", FindBookMark)
BringFrameToFront()    'bring frame to front
Message("The Second frame is in front.")
END FUNCTION
```

See also:

GoToCmd SendFrameToBack SelectFrameByName AddFrame AddFrameDLG

This function cascades and overlaps multiple open windows with the currently active window on top. Choosing this function is equivalent to choosing Window/Cascade.

Syntax

CascadeWindow()

Return Value

This function does not return a value.

Example

```
FUNCTION Example()
Text = UCASE$(Left$(Query$("What action to take (Tile, Cascade, or New) on MDI Windows?"), 1))
SWITCH Text
CASE "T"
TileWindow()
CASE "C"
CascadeWindow()
CASE "N"
NewWindow()
default
Message("Only the proposed options are available.")
ENDSWITCH
END FUNCTION
```

See also:

TileWindow NextWindow SelectWindow NewWindow

This function turns centering on or off. This function toggles the current state of the centering attribute. Choosing this function is equivalent to choosing Text/Alignment/Center.

Syntax

Center()

Return Value

1 (TRUE) if the text was centered, or if centering was removed.

-2 (GeneralFailure) if the alignment was not changed.

Example

```
FUNCTION Example()
text = Query$("Enter some text:")
New("_default.sty", 0, 0)     'open a new file
TYPE("{text}")
TYPE("[enter]")
TYPE("[ctrlhome][ctrlshiftend]")
Copy()  'copy text to clipboard
FOR I = 1 to 10
    Paste()    'paste text 10 times
NEXT
Message("The text will be shaded, bolded, italicized, underlined, and centered.")
TYPE("[ctrlhome][ctrlshiftend]")
Bold()
Italic()
Underline()
Center()
TYPE("[ctrlhome]")
END FUNCTION
```

See also:

Justify LeftAlign NormalText RightAlign

This function changes the action of the selected cascade menu item.

Syntax

ChangeCascadeAction(BarID, Menu, CascadeMenu, Item, NewAction, Help)

BarID is the identification number of the menu bar returned from the AddBar function. To use the Ami Pro menu bar, use 1.

Menu is the name of the pull down menu this cascade menu rests on. This must match exactly the name of the pull down menu you are referencing, including any ampersand (&) characters in the name of the menu. An ampersand is placed before a character that has an underline.

CascadeMenu is the name of the Cascade menu that the item to change is on. This must match exactly the name of the cascade menu that the item to change is on, including any ampersands (&).

Item is the name of the cascade menu item to be changed. If the menu item has a shortcut key, you must press **TAB**, type a ^, and then type the appropriate letter as part of the item name.

NewAction can either be an Ami Pro function or a macro.

Help is the one-line Help text that appears in the title bar of Ami Pro when this menu item is highlighted. This parameter is not optional for this function.

Return Value

1 (TRUE) if the item's action was changed.

0 (FALSE) if the change could not be made.

Example

```
FUNCTION Example()
MacFile = GetRunningMacroFile$() 'get the name of this macro
'change the Edit/Insert/Note text to say Hello World
ChangeCascadeAction(1, "&Edit", "&Insert", "&Note", "{MacFile}!Test()", "")
END FUNCTION

FUNCTION Test()
Message("Hello World...")
END FUNCTION
```

See also:

AddMenu AddBar AddMenuItem ChangeMenuAction ShowBar AddCascadeMenu AddCascadeMenuItem InsertMenu InsertMenuItem InsertCascadeMenu InsertCascadeMenuItem RenameMenuItem GrayMenuItem CheckMenuItem

This function changes the current icon set to the icon set specified in the Name parameter. Choosing this function is equivalent to choosing Tools/SmartIcons or the SmartIcons button on the status bar and specifying the desired icon set. A macro must be edited to insert this non-recordable function.

In Ami Pro release 2.0, the icon palettes are stored in .INI files. In Ami Pro release 3.0, the icon palettes are stored in .SMI files.

Modified in 3.0 In Ami Pro release 2.0, the Name parameter must be the file name of the .INI file.

Syntax

ChangeIcons(Name)

Name is the name of the icon set. This is the full icon set name, not the file name.

A null string ("") changes the current set to the default icon set.

Return Value

This function returns a 1.

Example

```
FUNCTION Example()
ChangeIcons("Working Together")
END FUNCTION
```

See also:

GetIconPalette IconBottom IconCustomize IconFloating IconLeft IconRight IconTop SetIconSize

This function changes the default language path and sets the language used for the current document and new documents. Choosing this function is equivalent to choosing Tools/Spell Check/Language Options.

Modified in 3.0 In Ami Pro release 2.0, dictionary 14 is Brazilian and dictionary 15 is Australian. The dictionaries 16 - 23 are not available is Ami Pro release 2.0.

Syntax

ChangeLanguage(Path, NewLang, CurrentLang)

Path is the full path where the dictionary file is located.

NewLang is the number of the language dictionary to use.

CurrentLang is the number of the language dictionary for this document.

Language numbers are:

1- American English
2 - British English
3 - French
4 - French Canadian
5 - Italian
6 - Spanish
7 - German
8 - Dutch
9 - Norwegian
10 - Swedish
11 - Danish
12 - Portuguese
13 - Finnish
14 - Medical
15 - Swiss German
16 - Greek
17 - Brazilian
18 - Australian
19 - Polish
20 - Russian
21 - Catalan
22 - Nynorsk
23 - Voorkeur

Return Value

1 (TRUE) if the language was changed.

0 (UserCancel/FALSE) if the language dictionary could not be changed or if the user canceled the function.

Example

```
FUNCTION Example()
' get the path for the dictionary from the AMIPRO.INI
path = GetProfileString$("AmiPro", "dictionary", "AMIPRO.INI")
newlang = 7    'German
currlang = 1   'American English
ChangeLanguage(path, newlang, currlang)
```

```
END FUNCTION
```

See also:

SpellCheck Thesaurus

This function changes the function of a menu item on a pull down menu.

Syntax

ChangeMenuAction(BarID, Menu, Item, MacroName[!Function[(parm1[, parm2...])]][, Help])

BarID is the identification number of the menu bar returned from the AddBar function. To use the Ami Pro menu bar, use 1.

Menu is the name of the pull down menu on which the item is located. This name must match the name of the pull down menu, including any ampersand (&) characters. An ampersand is placed before a character that has an underline.

Item is the name of the existing menu item to change. This name must match the name of the existing menu item, including any ampersand (&) characters. An ampersand is placed before a character that has an underline.

MacroName is the name of the Ami Pro function or macro to run if this menu item is selected. This parameter may contain the macro file name, the function within that file to call, and any parameters that function may require. At a minimum, this parameter must contain the macro file name.

Help is the one-line Help text that appears in the title bar of Ami Pro when this menu item is highlighted. If an Ami Pro function is used, its one-line Help appears.

Return Value

A positive number if the menu item was changed.

0 (FALSE) if the menu item could not be changed, or if an invalid BarID or MenuName was used.

Example

```
FUNCTION Example()
RMac = GetRunningMacroFile$() ' get the name of the running macro
ChangeMenuAction(1, "Too&ls", "Smart&Icons...", IconLeft, "Places the SmartIcon bar on the left")
' make the menu toggle the icon bar
OnCancel RestoreIt ' set a goto address in case the user cancels
UserControl("Try out the SmartIcon menu item then select Resume") ' try it out
RestoreIt:
' change the menu
ChangeMenuAction(1, "Too&ls", "Smart&Icons...", IconCustomize, "Customize SmartIcons by
specifying the desired position, size and set of icons")
END FUNCTION
```

See also:

AddBar AddMenu AddMenuItem AddMenuItemDDE ChangeMenuAction CheckMenuItem DeleteMenu DeleteMenuItem GetMacPath$ GrayMenuItem RenameMenuItem ShowBar AddCascadeMenu AddCascadeMenuItem ChangeCascadeAction InsertMenu InsertMenuItem InsertCascadeMenu InsertCascadeMenuItem

This function assigns a shortcut key combination to run a macro. It overrides an existing macro key definition. It does not check to make sure that the MacroName exists. If an invalid MacroName is used, the keystroke does nothing when pressed.

Syntax

ChangeShortcutKey(MacroName, Key, State)

MacroName is the name of the macro, including the file name that should be assigned to a keystroke.

Key is the alphanumeric function key used as the shortcut key combination.

State is the shift state of the key. It can be "C" (Control key), "S" (Shift key), "A" (Alternate key), or a combination of the three.

Return Value

1 (TRUE) if the macro was successfully assigned to the function key.

-2 (GeneralFailure) if the key could not be assigned because an invalid keyname was used.

Example

```
FUNCTION Example()
FileName = Query$("What macro do you want to change the shortcut key for?")
Key = Query$("What number function key do you want to assign?")
IF not IsNumeric(Key)
    Message("Not a valid function key; try again.")
ENDIF
IF Decide("Use Shift with F{key}?")
    State = "S"
ELSE
    State = ""
ENDIF
IF Decide("Use Ctrl with F{key}?")
    State = strcat$(State, "C")
ENDIF
IF Decide("Use Alt with F{key}?")
    State = strcat$(State, "A")
ENDIF
RealKey=strcat$("F", Key)
IF -2 = ChangeShortcutKey(Filename, RealKey, State)
    Message("Could not change shortcut key for {Filename} to {State}F{Key}")
ENDIF
END FUNCTION
```

See also:

AddCascadeMenu AddCascadeMenuItem AddMenu AddMenuItem AddBar ShowBar ChangeMenuAction InsertMenu InsertMenuItem OnKey InsertCascadeMenu InsertCascadeMenuItem

This function scrolls the document left one character without moving the insertion point. Choosing this function is equivalent to clicking once on the right arrow at the right of the horizontal scroll bar.

Syntax

CharLeft()

Return Value

This function returns 0.

Example

```
FUNCTION Example()
Again:
UserControl("Click Resume to shift to the left, or Cancel to Quit.")
CharLeft()
GoTo Again
END FUNCTION
```

See also:

CharRight EndOfFile LeftEdge LineDown LineUp RightEdge ScreenDown ScreenLeft ScreenRight ScreenUp TopOfFile

This function scrolls the document right one character without moving the insertion point. Choosing this function is equivalent to clicking once on the left arrow at the left of the horizontal scroll bar.

Syntax

CharRight()

Return Value

This function returns 0.

Example

```
FUNCTION Example()
Again:
UserControl("Click Resume to shift to the right, or Cancel to Quit.")
CharRight()
GoTo Again
END FUNCTION
```

See also:

CharLeft EndOfFile LeftEdge LineDown LineUp RightEdge ScreenDown ScreenLeft ScreenRight ScreenUp TopOfFile

This function displays the Charting dialog box. Choosing this function is equivalent to choosing Tools/Charting (charting must be installed). This function does not draw a chart automatically. The user must select the charting options manually.

Syntax

ChartingMode()

Return Value

1 (TRUE) if the charting mode was initiated.

-2 (GeneralFailure) if the charting mode was not initiated.

Example

```
FUNCTION Example()
New("_BASIC.STY", 0, 0)
TYPE("12 33 55 87 99[Enter]45 67 32 88 94[Enter][Enter]")
TYPE("[CtrlHome][CtrlShiftEND]")
Copy()
ChartingMode()
END FUNCTION
```

See also:

AddFrame ModifyLayout DrawingMode

This function places or removes a check mark next to a menu item. Use this function to indicate that the menu function is currently active.

Syntax

CheckMenuItem(BarID, Menu[, CascadeMenu], Item, State)

BarID is the identification number of the menu bar returned from the AddBar function. To use the Ami Pro menu bar, use 1.

Menu is the name of the pull down menu that the item to be checked or unchecked rests on. This must match exactly the name of the pull down menu you want to modify, including any ampersand (&) characters. An ampersand is placed before a character that has an underline.

CascadeMenu is the optional name of the cascade menu that the item to be checked or unchecked rests on. This must match exactly the name of the cascade menu you want to modify, including any ampersand (&) characters.

Item is the name of the menu item you want to check or uncheck. If the menu item has a shortcut key, you must press **TAB**, type a ^, and then type the appropriate letter as part of the item name.

State indicates whether the item is checked or unchecked. State can be either a 1 (On) or a 0 (Off).

Return Value

1 (TRUE) if the item was successfully checked or unchecked.

0 (FALSE) if the item could not be checked/unchecked, or if an invalid BarID, MenuName, or ItemName was used.

Example

```
FUNCTION Example()
MacFile = GetRunningMacroFile$()  'get the name of this macro
State = 0
DeleteMenuItem(1, "&Text", "B&old+Italic+Underline")
InsertMenuItem(1, "&Text", 10, "B&old+Italic+Underline", "{MacFile}!Example2({State})", "Bold and
Italicize and Underline Text.")
END FUNCTION


FUNCTION Example2(State)
MacFile = GetRunningMacroFile$()
' This is the most important line in the macro
' it handles the toggle feature passed as the parameter State
' by making zero into one, one into zero.
State = Right$((State - 1), 1)
Bold(State)
Underline(State)
Italic(State)
ChangeMenuAction(1, "&Text", "B&old+Italic+Underline", "{MacFile}!Example2({State})", "Bold and
Italicize and Underline Text.")
CheckMenuItem(1, "&Text", "B&old+Italic+Underline", State)
END FUNCTION
```

See also:

AddBar AddMenu AddMenuItem AddMenuItemDDE ChangeMenuAction ChangeMenuItem DeleteMenu DeleteMenuItem GrayMenuItem RenameMenuItem ShowBar AddCascadeMenu AddCascadeMenuItem ChangeCascadeAction InsertMenu InsertMenuItem InsertCascadeMenu InsertCascadeMenuItem

This function determines the ANSI character of the number provided.

Syntax

CHR$(Value)

Value is a number between 0 and 255.

Return Value

A single character string that is the ANSI character derived from the function's argument.

Example

```
FUNCTION Example()
TYPE("NumberCharacter[Enter]")
FOR I = 1 to 255
    Char = CHR$(I)
    TYPE("{I}={Char}[Enter]")
NEXT
END FUNCTION
```

See also:

ASC LCASE$ MID$ strchr strfield$ UCASE$ Right$ Left$ strcat$ Instr

This functions determines which elements display in Clean Screen mode. Choosing this function is equivalent to choosing View/View Preferences and choosing Clean Screen Options.

Syntax

CleanScreenOptions(Options)

Options is a flag that determines the screen elements to display. The Options parameter should be set to one or more of the following options:

None (0) - None of the elements display.
TitleBar (1) - Display the title bar. This value must be combined with MenuBar (2).
MenuBar (2) - Display the menu bar.
Icons (4) - Display the SmartIcons.
StatusBar (8) - Display the status bar.
VertScroll (16) - Display the vertical scroll bar.
HorzScroll (32) - Display the horizontal scroll bar.
ReturnIcon (64) - Display the icon to exit Clean Screen mode.

Return Value

This function returns 1.

Example

```
FUNCTION Example()
CleanScreenOptions(112)   ' Display the scroll bars and return icon
ToggleCleanScreen()
END FUNCTION
```

See also:

ToggleCleanScreen ViewPreferences

This function retrieves information from the Clipboard.

Syntax

ClipboardRead(Format)

Format is how the information is stored in the Clipboard. It passes a 0 for information in a text format.

Return Value

1 (TRUE) if the information was read.

0 (FALSE) if the information was not read.

Example

```
FUNCTION Example()
Stuff = Query$("Enter what you want put on the clipboard:")
ClipboardWrite(Stuff, 1)
CALL Example2()
END FUNCTION

FUNCTION Example2()
Paste()
TYPE("[Enter]Should look exactly like the line below:[Enter]")
TYPE(ClipboardRead(1))
END FUNCTION
```

See also:

ClipboardWrite Cut Copy Paste DDELinks

This function places information of a specified format on the Clipboard.

Syntax

ClipboardWrite(Data, Format)

Data is the string to be placed on the Clipboard.

Format is how the text is stored in the Clipboard. It passes 0 for information in a text format.

Return Value

1 (TRUE) if the information was written.

0 (FALSE) if the information was not written.

Example

```
FUNCTION Example()
Stuff = Query$("Enter what you want put on the clipboard:")
ClipboardWrite(Stuff, 1)
CALL Example2()
END FUNCTION


FUNCTION Example2()
Paste()
TYPE("[Enter]Should look exactly like the line below:[Enter]")
TYPE(ClipboardRead(1))
END FUNCTION
```

See also:

ClipboardRead Cut Copy Paste DDELinks

This function connects or disconnects the selected table cells. Choosing this function is equivalent to choosing Table/Connect Cells. The ConnectCells function is a toggle. If the selected cells are not connected, they become connected. If the selected cells are already connected, they become disconnected.

Syntax

ConnectCells()

Return Value

1 (TRUE) if the cells were successfully connected or disconnected.

-2 (GeneralFailure) if the cells could not be connected or disconnected.

-7 (CouldNotFind) if the cells to connect or disconnect could not be located.

Example

```
FUNCTION Example()
DEFSTR StartRow, StartCol, EndRow, EndCol;
Tables(1, TRUE, 4, 10)
WHILE StartRow != 2
    TYPE("[Right]")
    TableGetRange(&StartRow, &StartCol, &EndRow, &EndCol)
WEND
TYPE("[ShiftRight][ShiftRight]")
ConnectCells()
TableLines(AllSides, 0, 0, OnePoint, 0)
END FUNCTION
```

See also:

ProtectCells TableLayout Tables SizeColumnRow TableLines

This function runs the Microsoft Windows Control Panel. Choosing this function is equivalent to choosing System/Control Panel. Control Panel settings cannot be set directly with this macro function. This function opens the Control Panel window and then immediately returns control to the macro. The user can adjust the Control Panel settings, close the window, and return to word processing.

If later functions in the macro involve screen display, these functions cause the Ami Pro window to obscure the Control Panel window as the macro continues to run in the background. To prevent this from occurring either minimize the Ami Pro window before using this function or allow the function to be the last function in the macro. You could also place a Pause or UserControl box to halt operation of the macro temporarily.

Syntax

ControlPanel

Return Value

This function returns 1.

Example

```
FUNCTION Example()
UserControl("Click Resume to bring up Control Panel...")
ControlPanel
UserControl("Click Resume to shut down Control Panel...")
AppClose("Control Panel")
END FUNCTION
```

See also:

PrintSetup PrintOptions

This function copies the selected text, selected frame, or table to the clipboard. Choosing this function is equivalent to choosing Edit/Copy.

Syntax

Copy()

Return Value

1 (TRUE) if the text was copied to the clipboard.

-2 (GeneralFailure) if the text could not be copied or if there was no text to copy.

Example

```
FUNCTION Example()
TYPE("Hello World")
TYPE("[CtrlHome][CtrlShiftEND]")
Copy()
Paste()
Paste()
END FUNCTION
```

See also:

Cut Paste CurChar$ CurWord$ CurShade$ DDELinks ClipboardRead ClipboardWrite

This function displays the Create A New dialog box and allows you to create a NewWave object. Choosing this function is equivalent to choosing Objects/Create A New. This function does not automatically create a new object.

If a frame is selected, it must be empty to be able to insert the new object.

Syntax

CreateANew()

Return Value

1 (TRUE) if the new object was successfully created.

-2 (GeneralFailure) if the new object was not created.

Example

```
FUNCTION Example()
CreateANew
END FUNCTION
```

See also:

ImportText IsNewWave ListObjects NWGetContainerCount NWGetContainerNames NWGetCurrentContainer NWGetCurrentObject$ NWGetObjectCount NWGetObjectNames NWGetParent NWReferenceToFile$ ObjectAttributes OpenObject SaveAsMaster SaveAsObject Share ShowLinks

This function creates a new data file. It does not run the automated data file record entry function.

Syntax

CreateDataFile()

Return Value

This function returns 1.

Example

```
FUNCTION Example()
CreateDataFile()
END FUNCTION
```

See also:

CreateDescriptionFile Merge MergeToFile OpenDataFile OpenMergeFile

This function creates an Ami Pro merge description file. A description file contains a list of the fields in an external data file. Choosing this function is equivalent to File/Merge, choosing Option 1, selecting a file type and file name, and entering the field names in the Merge Data File Fields dialog box.

Syntax

CreateDescriptionFile(FileName, Type[, Fields...])

FileName is the name of the description file, in Ami Pro format, to be created.

Type is the format of the data file and is one of the following:

- Ami Pro
- 1-2-3
- 1-2-3 rel 3
- Comma delimited
- dBase
- DIF
- Excel
- Excel 3.0
- Excel 4.0
- Fixed length ASCII
- Paradox
- SuperCalc

Fields are the names of the fields to insert into the description file.

Return Value

This function returns 1.

Example

```
FUNCTION Example()
' this a description file with the name 123desc.sam with the following lines
CreateDescriptionFile("123desc.sam", "1-2-3 rel 3", "Name", "Address", "City", "State", "Zip")
END FUNCTION
```

See also:

CreateDataFile MergeToFile OpenDataFile OpenMergeFile

This function creates a new paragraph style. Choosing this function is equivalent to choosing Style/Create Style.

The paragraph style name cannot exceed thirteen characters.

Syntax

CreateStyle(NewName, BaseName, Options)

NewName is the name of the new paragraph style.

BaseName is the name of the existing paragraph style on which to base the new paragraph style.

Options is a flag which determines whether to create or modify a style. It also determines whether to create or modify a style based on the style at the insertion point position or in the BaseName parameter.

CreateSty (0) - Create style

ModSty (1) - Modify style

SelectSty (2) - Create or modify style based on the style at the insertion point position. If this parameter is used, the BaseName parameter is ignored.

You must still enter an empty string ("") for the BaseName.

The Options parameter must use either CreateSty or ModSty. SelectSty may be added to one of the other parameters.

To display the Create Styles dialog box and allow the user to select the new paragraph style name and the paragraph style it is based on: **CreateStyle**

Return Value

1 (TRUE) if the new paragraph style was created.

0 (UserCancel) if the user canceled the function.

-2 (GeneralFailure) if the paragraph style could not be created.

-6 (NoMemory) if the function failed because of insufficient memory.

Example

```
FUNCTION Example()
NewName = Query$("What do you want to name the new style?", "TestStyle")
' Get the current style name and create the new style from it
BaseName = GetStyleName$()
CreateStyle(NewName, BaseName, 0)
' select the new style name
ModifySelect(NewName)
' modify the paragraph style
ModifyAlignment(AlignLeft, 0, 0, 0, 0, 0)
ModifyBreaks(4, 0, 0)
ModifyEffects("<·10>", SpaceIndent, ".20", 0, 0, 0, 0)
ModifySpacing(2, 0, 0, 0, 0, 0, 100)
ModifyTable(3, 2, ".", ",", "$", (8 + 16 + 128 + 256))
ModifyLines(1, 2, 180, 0, 0, 0, 0, 65535)
ModifyFont("TimesNewRomanPS", (20 * 20), 255, 1)
SetStyle(NewName)
' Reread the style information.
ModifyReflow()
TYPE("This[Enter]is what the new[Enter]style looks[Enter]Like...[Enter]")
END FUNCTION
```

See also:

ModifyAlignment ModifyBreaks ModifyEffects ModifyFont ModifyLines ModifyStyle ModifyReflow ModifySelect ModifySpacing ModifyTable SaveAsNewStyle UseAnotherStyle

StyleManageAction StyleManageFinish StyleManageInit StyleManagement

This function is used to determine the character to the right of the insertion point.

Syntax

CurChar$()

Return Value

The character to the right of the insertion point.

The null string ("") if there is a mark at the insertion point. Marks returning the null string include a bookmark, a note, a date mark, a merge variable, the end of a paragraph, a footnote, etc.

Example

```
FUNCTION Example()
UserControl("Place your cursor in the document, then click Resume...")
Message(CurChar$())
UserControl("Shade some text, then click Resume...")
Message(CurShade$())
UserControl("Double-click a word in your document, then click Resume...")
Message(CurWord$())
END FUNCTION
```

See also:

CurWord$ CurShade$ GoToCmd GetMarkText$ GetTextBeforeCursor$ GoToShade

This function is used to obtain the value of the selected text.

Syntax

CurShade$()

Return Value

The null string ("") if there is no text selected.

Only printable characters are returned. Marks in the text are ignored.

Example

```
FUNCTION Example()
UserControl("Place your cursor in the document, then click Resume...")
Message(CurChar$())
UserControl("Shade some text, then click Resume...")
Message(CurShade$())
UserControl("Click a word in your document, then click Resume...")
Message(CurWord$())
END FUNCTION
```

See also:

CurChar$ CurWord$ GoToCmd GetMarkText$ GetTextBeforeCursor$ GoToShade

This function is used to determine the location of the insertion point on the page. X represents the horizontal position on the page in twips (1 inch = 1440 twips). Y represents the vertical position on the page in twips. The upper left corner of the page is location 0,0. The X value gets larger as the position moves right. The Y value is negative, and becomes a larger negative number as the position moves down.

Syntax

CursorPosition$()

Return Value

The null string ("") if the insertion point is not displayed, is not on the current page, or if Ami Pro is in Draft Mode.

A string with the position of the insertion point in the format "X, Y".

Example

```
FUNCTION Example()
LayoutMode()
' Get current cursor position, in twips
Position = CursorPosition$()
' Parse out the X and Y coordinate
X = strfield$(Position, 1, ",")
Y = strfield$(Position, 2, ",")
WidthTwips=Query$("In inches, how wide do you want the frame?", "1")
WidthInches=WidthTwips * 1440
' Compute x-endpoint of box
X2 = X + WidthInches
HeightTwips=Query$("In inches, how tall do you want the frame?", "1")
HeightInches=HeightTwips * 1440
' Compute y-endpoint of box
Y2 = Y + HeightInches
' Add the frame, using current defaults
AddFrame(X, Y, X2, Y2)
END FUNCTION
```

See also:

AddFrame GetTextBeforeCursor$ AddFrameDLG TYPE strfield$ GetMode LayoutMode DraftMode OutlineMode

This function is used to obtain the value of the word at the insertion point.

Syntax

CurWord$()

Return Value

Only printable characters. Marks in the text are ignored.

Example

```
FUNCTION Example()
UserControl("Place your cursor in the document, then click Resume...")
Message(CurChar$())
UserControl("Shade some text, then click Resume...")
Message(CurShade$())
UserControl("Click a word in your document, then click Resume...")
Message(CurWord$())
END FUNCTION
```

See also:

CurChar$ CurShade$ GoToCmd GetMarkText$ GetTextBeforeCursor$ GoToShade

This function changes the current view level to Custom View. Choosing this function is equivalent to choosing View/Custom.

Syntax

CustomView()

Return Value

This function returns 1.

Example

```
FUNCTION Example()
CustomView()
END FUNCTION
```

See also:

EnlargedView FacingView FullPageView GetViewLevel LayoutMode StandardView

This function is used to cut the selected text, selected frame, or table from the document and save it to the clipboard. Choosing this function is equivalent to choosing Edit/Cut.

Syntax

Cut()

Return Value

This function does not return a value. If no text was shaded, the macro terminates with an error message.

Example

```
FUNCTION Example()
UserControl("Shade the text to move to the end of the doc, then click Resume...")
Cut()      ' cut the selected text
TYPE("[CtrlEND]")  ' move to the end of the file
Paste()    ' paste the text at the end of the file
END FUNCTION
```

See also:

Copy Paste CurChar$ CurWord$ CurShade$ DDELinks ClipboardRead ClipboardWrite

This function takes two dates (Date1, Date2) and calculates the number of days elapsed by subtracting Date2 from Date1.

Syntax

DateDiff(Date1, Date2)

Date1 is the first date to examine. It can be stated as the number of seconds since January 1, 1970 or the current Windows date format.

Date2 is the second date to examine. It can be stated as the number of seconds since January 1, 1970 or the current Windows date format.

Return Value

The number of days the two dates differ.

Example

```
FUNCTION Example()
Born = Query$("What is your Birthday (MM/DD/YYYY)?")
Date = FormatDate$(Now(), "h")    'current date
Time = FormatTime$(Now(), 6)  'current time
Days = DateDiff(Born, Date)   ' how many days old
TextDate = FormatDate$(Now(), "d")    'format the date
Message("It is now {Time} on {TextDate}. You are {Days} days old.")
END FUNCTION
```

See also:

FormatDate$ FormatTime$ Now GetTime

This function is used to have another application notify Ami Pro when data has changed in that application. Before using the DDEAdvise function, a conversation with the other application must be started using the DDEInitiate function.

Following the DDEAdvise function, Ami Pro continues operation normally. When data is received from the channel and item, the macro specified in the DDEAdvise function is started. The data received from the other application (in CF_Text format) is passed to the macro specified in the DDEAdvise function as the first argument to the macro. It is up to the macro to parse the data from the other application as needed.

Syntax

DDEAdvise(ChannelID, Item, MacroName[!Function])

ChannelID is the channel ID of the Dynamic Data Exchange (DDE) conversation.

Item describes the information requested from the linked application.

MacroName is the name of an Ami Pro macro to execute when data changes in the linked application.

Return Value

1 (TRUE) if the other application responded to the Ami Pro request.

0 (FALSE) if the other application could not be contacted or if an invalid ChannelID was used.

Example

```
FUNCTION Example()
' This example starts a conversation with 123w untitled worksheet and
' sets up a dde advise back to Ami Pro with the cells A1 and A2.

' If the user types data into A1, Ami Pro is notified and displays the value.
' If the user types data into A2, Ami Pro breaks the Advise with DDEUnAdvise
' and terminates the conversation.

id = DDEInitiate("123w", "Untitled") ' start a dde conversation with 123w
IF id = 0
    Exec("c:\123w\123w.exe", "") ' launch app if unable to connect
    id = DDEInitiate("123w", "Untitled") ' contact one more time
    IF id = 0
        Message("Unable to contact server application.") ' failed to contact
        EXIT FUNCTION
    ENDIF
ENDIF
AllocGlobalVar("ChannelID", 1) ' create a global variable
SetGlobalVar("ChannelID", id) ' store the channel id in a global variable
RMac = GetRunningMacroFile$() ' what macro name are we running?
DDEAdvise(id, "A1", "{RMac}!GetData") ' set up advise for cell A1
DDEAdvise(id, "A2", "{RMac}!EndDDE") ' set up advise for cell A2
END FUNCTION

Function GetData(value)
Message("The value just typed in A1 = {value}") ' display the value
END FUNCTION

Function EndDDE()
id = GetGlobalvar$("ChannelID") ' get the channel id
DDEUnAdvise(id, "A1") ' unadvise on both cells
DDEUnAdvise(id, "A2")
DDETerminate(id) ' terminate the dde conversation
FreeGlobalVar("ChannelID") ' free the global variable
Message("DDE Session Terminated!") ' let the user know it worked
```

```
END FUNCTION
```

See also:

AddMenuItemDDE DDEExecute DDEInitiate DDEPoke DDEReceive$ DDETerminate DDEUnAdvise Exec GetMacPath$ GetRunningMacroFile$

This function gives a command to an application using the Windows Dynamic Data Exchange (DDE). Before using this function, a conversation with the other application must be started using the DDEInitiate function. The command given to the linked application is defined by that application.

Syntax

DDEExecute(ChannelID, Command)

ChannelID is the channel ID of the DDE conversation.

Command is the command, in square brackets, to give to the linked application.

Return Value

0 if the command was successfully sent to the other application.

-1 if the application refused the command or if an invalid ChannelID was used.

Example

```
FUNCTION Example()
' This example commands 123w to open the file EMPLOYEE.WK3

' start a dde conversation with 123w
id = DDEInitiate("123w", "Untitled")
IF id = 0
    ' launch app if unable to connect
    Exec("c:\123w\123w.exe", "")
    ' contact one more time
    id = DDEInitiate("123w", "Untitled")
    IF id = 0
        ' failed to contact
        Message("Unable to contact server application.")
        EXIT FUNCTION
    ENDIF
ENDIF
onerror endit ' set up an error address
filename = "Employee.wk3" ' assign a file to a variable
' assign a command to a variable
command = "[[RUN(""/FR{filename}~"")]"
DDEExecute(id, command) ' command 123w to execute the command
endit:
DDETerminate(id) ' terminate the conversation
END FUNCTION
```

See also:

ActivateApp AddMenuItemDDE DDEAdvise DDEInitiate DDEPoke DDEReceive$ DDETerminate DDEUnAdvise Exec

This function starts a conversation with another application using the Windows Dynamic Data Exchange (DDE). The application must be capable of communicating using the DDE protocol. This function must be used prior to any other DDE activity with an application.

Syntax

DDEInitiate(App, Data)

App is the application that is the target of the DDE conversation.

Data is defined by the other application, but usually is the name of the application's data file you want to access.

Return Value

A positive number if the ChannelID of the conversation was successfully established.

0 (FALSE) if the conversation could not be started. This was because the application was not running, did not have the Topic open, or does not support DDE.

Example

```
FUNCTION Example()
Mac = GetRunningMacroFile$()
id = DDEInitiate("123W", "Untitled")
IF id = 0
    Exec("123W.EXE","") ' No id, try to start 123W
    id = DDEInitiate("123W", "Untitled")
        IF id = 0
            Message("Could not initiate a conversation with 1-2-3!")
            Exit Function
        ENDIF
ENDIF
' Poke data into spreadsheet
Address = "A2" ' This could be a named range
Value = 30
DDEPoke(id,Address,Value)
DDEPoke(id, "A1", "200")
DDETerminate(Id)   ' Terminate the conversation
END FUNCTION
```

See also:

DDEAdvise DDEExecute DDEPoke DDEReceive$ DDETerminate DDEUnAdvise Exec

This function displays the Link Options dialog box. Choosing this function is equivalent to choosing Edit/Link Options. Links cannot be adjusted automatically using a macro. When the dialog box displays, the user can view and adjust them manually and then choose OK or Cancel to return to the macro.

Syntax

DDELinks(Flag, Count[, LinkSpecPairs][, App, Data, Item])

Flag is the action taken.

1 - Unlink
2 - Update
3 - Stop
4 - Change
5 - Create

Count is the number of LinkSpecPairs. Each pair consists of the internal ID and the Type (text or frame).

Create is probably the most commonly used value.

Return Value

1 (TRUE) if the links dialog box was successfully displayed.

0 (UserCancel) if the user canceled the function.

Example

```
FUNCTION Example()
' this command was initiated with a DDE link from cell A2 in a 1-2-3
' for Windows file called FILE0001.WK3 into an Ami Pro document
' called TESTFILE.SAM.  This document was opened and initially DDE links
' were not updated.

flag=2  'update link
numlinks=1  'number of link to change with this statement
whichlink=1'the number of the link to change
linktype="Text"    'the kind of link to change
DDELinks(flag, numlinks, whichlink, linktype)
END FUNCTION
```

See also:

Paste Cut Copy ClipboardRead ClipboardWrite

This function is used to send data from Ami Pro to another application or to a bookmark. Before using this function, a conversation with the other application must be started using the DDEInitiate function.

If you name an existing text bookmark in a DDEPoke message, Ami Pro replaces the contents of the bookmark with the poked data.

Syntax

DDEPoke(ChannelID, Location, Data)

ChannelID is the channel ID of the Dynamic Data Exchange (DDE) conversation.

Location is the location in the other application to poke the data or the name of the bookmark.

Data is the data to send to the other application.

Return Value

1 (TRUE) if the other application accepted the data sent by Ami Pro.

0 (FALSE) if the other application could not be contacted or if an invalid ChannelID was used.

Example

```
FUNCTION Example()
Mac = GetRunningMacroFile$()
id = DDEInitiate("123W", "Untitled")
IF id = 0
    ' No id, try to start 123W
    Exec("123W.EXE","")
    id = DDEInitiate("123W", "Untitled")
        IF id = 0
            Message("Could not initiate a conversation with 1-2-3!")
            Exit Function
        ENDIF
ENDIF

' Poke data into spreadsheet
Address = "A2" ' This could be a named range
Value = 30
DDEPoke(id,Address,Value)
DDEPoke(id, "A1", "200")
DDETerminate(Id)   ' Terminate the conversation
END FUNCTION
```

See also:

AddMenuItemDDE DDEAdvise DDEExecute DDEInitiate DDEReceive$ DDETerminate DDEUnAdvise Exec

This function receives up to 250 characters from an application using the Windows Dynamic Data Exchange (DDE). Before using this function, a conversation with the other application must be started using the DDEInitiate function. When this function is used, Ami Pro waits until the other application processes the request and then receives the data.

Syntax

DDEReceive$(ChannelID, Item)

ChannelID is the channel ID of the DDE conversation.

Item describes the information requested from the linked application.

Return Value

A null string ("") if the application could not send the data.

The data requested from the other application.

Example

```
FUNCTION Example()
id = DDEInitiate("123W", "Untitled")
IF id = 0
    Exec("C:\123W\123W.EXE", 1)
    FOR I = 1 to 10
        id = DDEInitiate("123W", "Untitled")
        IF id > 0
            OK = TRUE
            BREAK
        ENDIF
    NEXT
    IF not OK
        Message("Could not initiate a conversation with 1-2-3!")
        EXIT FUNCTION
    ENDIF
ENDIF
Principal = Query$("What is the principal of the loan?")
Interest = Query$("What is the annual interest rate (12% = .12)?") / 12
Term = Query$("What is the term in months?")
IF Interest > 1
    Interest = Interest / 100
ELSEIF Right$(Interest, 1) = "%"
    Interest = Left$(Interest, (Len(Interest) - 1)) / 100
ENDIF
DDEExecute(id, "[[RUN(""+@PMT({Principal},{Interest},{Term})~"")]")
Value = DDEReceive$(id, "A1")
DDETerminate(id)
Message(Value)
END FUNCTION
```

See also:

AddMenuItemDDE DDEAdvise DDEExecute DDEInitiate DDEPoke DDETerminate DDEUnAdvise Exec

This function ends a conversation with an application using the Windows Dynamic Data Exchange (DDE). When this function is used, Ami Pro closes the link with the other application. If more information is needed after this function is used, a new conversation must be started.

Syntax

DDETerminate(ChannelID)

ChannelID is the channel ID of the DDE conversation.

Return Value

1 (TRUE) if the conversation was successfully concluded.

0 (FALSE) if the conversation was not concluded or if an invalid ChannelID was used.

Example

```
FUNCTION Example()
Mac = GetRunningMacroFile$()
id = DDEInitiate("123W", "Untitled")
IF id = 0
    ' No id, try to start 123W
    Exec("123W.EXE","")
    id = DDEInitiate("123W", "Untitled")
        IF id = 0
            Message("Could not initiate a conversation with 1-2-3!")
            Exit Function
        ENDIF
ENDIF

' Poke data into spreadsheet
Address = "A2" ' This could be a named range
Value = 30
DDEPoke(id,Address,Value)
DDEPoke(id, "A1", "200")
DDETerminate(Id)   ' Terminate the conversation
END FUNCTION
```

See also:

AddMenuItemDDE DDEAdvise DDEExecute DDEInitiate DDEPoke DDEReceive$ DDEUnAdvise Exec

This function is used to reset the notification status set up using the DDEAdvise function. When this function is used, the other application on the current Dynamic Data Exchange (DDE) channel no longer notifies Ami Pro when data changes have occurred.

Syntax

DDEUnAdvise(ChannelID, Item)

ChannelID is the channel ID of the DDE conversation.

Item describes the information requested from the linked application.

Return Value

1 (TRUE) if the other application responded to the Ami Pro request.

0 (FALSE) if the other application could not be contacted or if an invalid ChannelID was used.

Example

```
FUNCTION Example()
MacFile = GetRunningMacroFile$()
DECLARE GetNumber()
id = DDEInitiate("123W", "Untitled")
IF id = 0
    Exec("123W.EXE", 1)
    FOR I = 1 to 10
        id = DDEInitiate("123W", "Untitled")
        IF id > 0
            OK = TRUE
            BREAK
        ENDIF
    NEXT
    IF not OK
        Message("Could not initiate a conversation with 1-2-3!")
        EXIT FUNCTION
    ENDIF
ENDIF
AllocGlobalVar("ChannelNumber", 1)
SetGlobalVar("ChannelNumber", id)
DDEPoke(id, "A1", "12345")
DDEAdvise(id, "A1", "{MacFile}!GetNumber")
GoTo endit
toast:
DDEUnAdvise(id, "A1")
FreeGlobalVar("ChannelNumber")
DDETerminate(id)
endit:
END FUNCTION


FUNCTION GetNumber()
id = GetGlobalVar$("ChannelNumber")
FreeGlobalVar("ChannelNumber")
Value = DDEReceive$(id, "A1")
DDEUnAdvise(id, "A1")
DDETerminate(id)
Message("The contents of cell A1 is {Value}")
END FUNCTION
```

See also:

AddMenuItemDDE DDEAdvise DDEExecute DDEInitiate DDEPoke DDEReceive$ DDETerminate Exec

This function displays a standard Windows message box with a question mark icon, the specified prompt, and Yes and No push buttons. It waits for the user to select a push button and then returns the ID of the button pressed.

DecideYes (1) - Yes

DecideNo (0) - No

Modified in 3.0 The Title parameter has been added to Ami Pro release 3.0.

Syntax

Decide(Prompt[, Title])

prompt is a string used as a prompt to the user. It can be a maximum of 80 characters.

Title is the title for the message box. The default is "Ami Pro Macro."

Return Value

A number representing the number pressed.

Example

```
FUNCTION Example()
again:
result = Decide("See this message again?")
IF result = 1
    GoTo again
ENDIF
END FUNCTION
```

See also:

DialogBox Message MultiDecide Query$ UserControl

This function pre-defines a function to the macro compiler. This allows you to call a non-defined function without having to use a CALL statement.

Syntax

DECLARE [FileName!]MacroName([parm1[, parm2...]])[, ALIAS ShortName]

FileName is the optional file name of the function to be pre-defined. If not specified, the macro is assumed to be in the current file.

MacroName is the name of the macro to be pre-defined.

ALIAS is a reserved word that takes the parameter **ShortName** if used. This optional specification allows you to assign the macro function that has been pre-defined on this line to a shorter name.

Any parameters must be specified so the compiler knows how many to expect.

Return Value

This function does not return a value.

Example

```
DECLARE GetAverage(Num1,Num2)

FUNCTION Example()
Value1 = 40
Value2 = 30
Stat = GetAverage(Value1,Value2)
Message("The average of {Value1} and {Value2} is {Stat}")
END FUNCTION

FUNCTION GetAverage(Num1,Num2)
Average = (Num1+Num2)/2
Return Average
END FUNCTION
```

See also:

CALL,CALLI

This function takes the attributes of the selected text and applies it to the currently selected style. Choosing this function is equivalent to choosing Style/Define Style.

Syntax

DefineStyle()

Return Value

1 (TRUE) if the style was changed.

0 (UserCancel) if the user canceled the function.

Example

```
FUNCTION Example()
' Type some text and apply an attribute
Type ("This is some text")
Type ("[ShiftCtrlLeft]")  ' Shade one word
Bold()
Italic()
DefineStyle()      ' Apply the attributes to the current style
END FUNCTION
```

See also:

CreateStyle ModifyStyle ModifyLines ModifyAlignment ModifyReflow ModifyBreaks ModifySelect ModifyEffects ModifySpacing ModifyFont ModifyTable UseAnotherStyle SaveAsNewStyle StyleManageAction StyleManageFinish StyleManageInit StyleManagement

This function deletes a column or row from a table. Choosing this function is equivalent to choosing Table/Delete Column/Row.

Syntax

DeleteColumnRow(Which)

Which determines whether to delete the current column or the current row. This parameter should be set to one of the following values:

Column (1) - Delete the current column

Row (0) - Delete the current row

To display a dialog box that allows the user to decide whether to delete a column or row:
DeleteColumnRow

Return Value

1 (TRUE) if the column or row was deleted.

0 (UserCancel/FALSE) if the user canceled the function or if no action was taken.

Example

```
FUNCTION Example()
DeleteColumnRow(Column)
END FUNCTION
```

See also:

InsertColumnRow SizeColumnRow

This function deletes the selected table. Choosing this function is equivalent to choosing Table/Delete Entire Table.

Syntax

DeleteEntireTable()

Return Value

0 (NoAction) if no action is taken. The table may not exist or may not be selected.

1 (True) if the table is deleted.

-2 (GeneralFailure) if the function failed.

Example

```
FUNCTION Example()
DeleteEntireTable()
END FUNCTION
```

See also:

DeleteColumnRow InsertColumnRow SizeColumnRow TableLayout TableLines Tables

This function deletes a menu from a menu bar.

Syntax

DeleteMenu(BarID, Menu)

BarID is the identification number of the menu bar returned from the AddBar function. To use the Ami Pro menu bar, use 1.

Menu is the name of the pull down menu to be deleted. This name must match the name of the pull down menu you want to delete, including any ampersand (&) characters. An ampersand is placed before a character that has an underline.

Return Value

1 (TRUE) if the menu was successfully deleted.

0 (FALSE) if the menu could not be deleted or if an invalid BarID or Menu name was used.

Example

```
FUNCTION Example()
' get the name of this macro
Mac = GetRunningMacroFile$()
Menu = "&Custom"
DeleteMenu(1,Menu)
AddMenu(1,Menu)    'Add a menu called Custom
' add menu items
AddMenuItem(1,Menu,"New...",New,"")
AddMenuItem(1,Menu,"My Copy",Copy,"")
AddMenuItem(1,Menu,"-","","")
AddMenuItem(1,Menu,"Remove {Menu}","{Mac}!Remove()","Remove {Menu} menu.")
End Function

Function Remove()
DeleteMenu(1,"&Custom")
END FUNCTION
```

See also:

AddBar AddMenu AddMenuItem AddMenuItemDDE ChangeMenuAction CheckMenuItem DeleteMenuItem GrayMenuItem RenameMenuItem ShowBar

This function deletes a menu item from a menu or a cascade menu.

Syntax

DeleteMenuItem(BarID, Menu[, CascadeMenu], Item)

BarID is the identification number of the menu bar returned from the AddBar function. To use the Ami Pro menu bar, use 1.

Menu is the name of the pull down where the item to be deleted is located. This name must match the name of the pull down menu, including any ampersand (&) characters. An ampersand is placed before a character that has an underline.

CascadeMenu is the optional name of the cascade menu where the item to be deleted is located. This name must match the name of the cascade menu, including any ampersand (&) characters. An ampersand is placed before a character that has an underline.

ItemName is the name of the menu item to be deleted. This name must match. If the menu item has a shortcut key, you must press **TAB**, type a ^, and then type the appropriate letter as part of the item name.

Return Value

1 (TRUE) if the item was successfully deleted.

0 (FALSE) if the item could not be deleted or if an invalid BarID, MenuName, or ItemName was used.

Example

```
FUNCTION Example()
RMac = GetRunningMacroFile$() ' get the name of the running macro
DeleteMenuItem(1, "&File", "&Print...^P")
OnCancel RestoreIt ' set a goto address in case the user cancels
UserControl("The Print menu item is gone. Select Resume to restore.")
RestoreIt:
' put the the menu back
InsertMenuItem(1, "&File", 17, "&Print...^P", fileprint, "Print the current document.")
END FUNCTION
```

See also:

AddBar AddMenu AddMenuItem AddMenuItemDDE ChangeMenuAction CheckMenuItem DeleteMenu GrayMenuItem RenameMenuItem ShowBar

This function displays a modal dialog box that has been created using the Ami Pro Dialog Editor or the Microsoft Windows Software Developer's Kit. The user can make choices from the dialog box. When done, the user can choose a push button to return control to the macro.

Fields within the dialog box should be assigned as follows:

0 - 19 - Push Buttons to close the dialog box
20 - 99 - Push buttons, Group Boxes, Radio Buttons, and Check Boxes
1000 - 7998 - Static Text Fields
7999 - Displays current directory
8000 - 8999 - Edit Boxes
9000 - 9499 - List Boxes
9500 - 9999 - Combo Boxes

The FillList function can be used to supply the contents of a list box prior to display of the dialog box. The FillEdit function can be used to supply the contents of any edit box or list box. It can also be used to supply the state of a button prior to display of the dialog box. Once the dialog box display is complete, the GetDialogField$ function can be used to retrieve the contents of the dialog box fields.

For more information about creating dialog boxes, see Dialog Editor.

Syntax

DialogBox(ResourceFile, DialogBoxName)

ResourceFile is the name of the file which contains the dialog box. If the dialog box was created using the Ami Pro Dialog Editor and is the only dialog box in the file, this parameter should be the null string (""). If the dialog box was created using the Ami Pro Dialog Editor and resides in the currently running macro file, this parameter should be a period (".").

DialogBoxName is the name of the dialog box to be accessed. If the dialog box was created using the Ami Pro Dialog Editor and is the only dialog box in an external file, this parameter should be set to the name of the file.

Return Value

The ID of the push button pressed.

0 (UserCancel) if the user canceled the function.

-1 (NotFound) if the resource file was not found.

Example

```
FUNCTION Example()
Filledit(20,1)     ' Turn apples on
Filledit(8000,"Bob")   ' Put text in Edit Box
Filledit(9001,"*.sam")
Box = DialogBox(".","ExampleBox")
IF Box<>1
    EXIT FUNCTION
ENDIF
Store = GetdialogField$(22)       ' Retrieve Combo Box
File = GetdialogField$(9001)      ' Retrieve List Box result
Name = GetDialogField$(8000)  ' Retrieve Edit Box
Dir = GetCurrentDir$()     ' Get current directory
Message("Goto Store = {Store} Name = {Name} File = {File}")
END FUNCTION

DIALOG ExampleBox
-2134376448 10 71 44 115 98 "" "" "Sample Dialog Box"
FONT 8 "Helv"
```

```
69 3 40 14 1 1342373889 "button" "OK" 0
69 19 40 14 2 1342373888 "button" "Cancel" 0
6 24 52 56 9001 1352728579 "listbox" "" 0
6 13 40 10 7999 1342177280 "static" "" 0
6 4 40 10 1001 1342177280 "static" "&Files:" 0
69 37 37 12 20 1342242825 "button" "&Apples" 0
69 49 42 12 21 1342242825 "button" "&Oranges" 0
6 83 45 12 22 1342242819 "button" "&Goto Store" 0
69 61 42 12 23 1342242825 "button" "&Soap" 0
69 82 43 12 8000 1350631552 "edit" "" 0
END DIALOG
```

See also:

Decide FillEdit FillList GetDialogField$ GetMacPath$ Message MultiDecide Query$ UserControl SetDlgCallback SetDLGCallBack GetDLGItem GetDLGItemText SetDLGItemText

This function sets a callback function that is "called" when the user presses the specified key while processing a macro dialog box. It should be used before the DialogBox statement and is in effect during that dialog box only. This function is passed the dialog box handle (HDLG) and the virtual key.

Syntax

DlgKeyInterrupt(Key, Function)

Key is a virtual key, such as [F1] or [PgDown]. The [F1] key should be set if you are using the "AmiDialog" in the Dialog code. This allows you to imitate the help system.

Function is the function that is executed when the specified key is pressed. This parameter may contain the macro file name or the function within that file to call.

Return Value

This function does not return a value.

Example

```
FUNCTION Example()
' run the HelpFun function if user hits F1 or Help button
DlgKeyInterrupt([F1], "HelpFun")
DialogBox(".", "QT")   'read dialog box from this file
END FUNCTION

FUNCTION HelpFun(hdlg, key)
Message("This is help!")
END FUNCTION

DIALOG QT
-2134376448 4 58 44 160 50 "" "AmiDialog" "Key Int Text"
FONT 8 "Helv"
112 6 40 14 1 1342373889 "button" "OK" 0
112 24 40 14 2 1342373888 "button" "Cancel" 0
14 8 50 12 20 1342242825 "button" "Radio 1" 0
14 22 50 12 21 1342242825 "button" "Radio 2" 0
END DIALOG
```

See also:

KeyInterrupt MouseInterrupt

This function calls a Dynamic Link Library (DLL) function that has been initialized using the DLLLocate or DLLLoadLib function. DLL functions are extremely powerful tools that provide access to Windows functions and to specially written functions. They are for the benefit of experienced Windows programmers.

Before using these functions, make sure that you know how to use them correctly. If they are used incorrectly, you may have problems with Ami Pro and Windows after their use.

Syntax

DLLCall(DLLId[, Parm1[, Parm2...]])

DLLId is the number returned as the ID of the DLL module by the DLLLocate function. It can also be a string, "Lib, Function, Parameters." These are the three parameters passed to the DLLLoadLib and DLLLocate functions. When using this string, the user does not need to call the DLLLoadLib or DLLLocate function.

Parm1 and **Parm2** are the parameters passed to the function.

Return Value

The return value of the DLLCall function defined by the module that is called.

Example

```
FUNCTION Example()
Text = Query$("Enter text for window:")  'Get the Title Text
hGetActiveWindow = DLLLocate("User", "GetActiveWindow", "H")
' Find the handle to the DLL
hWnd = DLLCall(hGetActiveWindow)  'and get the active window handle
hSetActiveText = DLLLocate("User", "SetWindowText", "IHC")
' Find the handle to the DLL
DLLCall(hSetActiveText, hWnd, Text)  'and set the text through the DLL
END FUNCTION
```

See also:

DLLLocate Exec DLLLoadLib DLLFreeLib

This function unloads the library loaded by the DLLLoadLib function.

Syntax

DLLFreeLib(LibID)

LibID is the ID returned by the DLLLoadLib function.

Return Value

1 (TRUE) if the DLL was successfully freed.

0 (FALSE) if the DLL could not be freed.

Example

```
FUNCTION Example()
Text = Query$("Enter text for window:")  'Get new title bar text
hGetActiveWindow = DLLLoadLib("User", "GetActiveWindow", "H")
' Load DLL into memory
hWnd = DLLCall(hGetActiveWindow)  'Call for active window handle
hSetActiveText = DLLLoadLib("User", "SetWindowText", "IHC")
' Load DLL into memory
DLLCall(hSetActiveText, hWnd, Text)  'Call to set title bar text
DLLFreeLib(hGetActiveWindow)  'Free DLL
DLLFreeLib(hSetActiveText)    'Free DLL
END FUNCTION
```

See also:

DLLCall DLLLoadLib DLLLocate

This function is used to verify that a Dynamic Link Library (DLL) function is available. It is also used to load DLL functions into memory so that they can be used later. Any DLL function can be used in your macro. The function must have been declared as a FAR PASCAL function when it was compiled. The parameter list, if described, must be a string containing letters specifying the variable type of the function's return value. This must be followed by one letter per parameter. DLL functions located in USER, GDI, and kernel do not need to be loaded.

These parameters are passed to the function when it is called. The letter codes for variable types are:

A - Boolean integer
B - Far pointer to double
C - Far pointer to string
E - Double
H - Unsigned integer
I - Integer
J - Unsigned long integer
L - Far pointer to word (Word Far *)
M - Far pointer to array or string (CHAR FAR * FAR *)
N - Far pointer to array or words (Word Far *)
O - Far pointer to a long (LONG FAR *)

DLL functions are extremely powerful tools that provide access to Windows functions and to specially written functions. They are for the benefit of experienced Windows programmers.

Before using these functions, make sure that you know how to use them correctly. If they are used incorrectly, you may have problems with Ami Pro and Windows after their use.

Syntax

DLLLoadLib(Module, Function, Parameters)

Module is the file name of the Dynamic Link Library module.

Function is the name of the function within the DLL module to be accessed.

Parameters is a string with the prototype parameters and return code for the function.

Return Value

1 (TRUE) if the DLL was successfully loaded.

0 (FALSE) if the DLL could not be loaded or could not be found.

Example

```
FUNCTION Example()
Text = Query$("Enter text for window:")  'Get new title bar text
hGetActiveWindow = DLLLoadLib("User", "GetActiveWindow", "H")
' Load DLL into memory
hWnd = DLLCall(hGetActiveWindow)  'Call for active window handle
hSetActiveText = DLLLoadLib("User", "SetWindowText", "IHC")
' Load DLL into memory
DLLCall(hSetActiveText, hWnd, Text)   'Call to set title bar text
DLLFreeLib(hGetActiveWindow)  'Free DLL
DLLFreeLib(hSetActiveText)    'Free DLL
END FUNCTION
```

See also:

DLLCall DLLFreeLib DLLLocate

This function is used to verify that a Dynamic Link Library (DLL) function is available for use. It is also used to inform Ami Pro of the function's parameters and return type so that the function can be used later. Any DLL function can be used in your macro. The function should have been declared as a FAR PASCAL function when it was compiled. The parameter list, if described, must be a string containing letters specifying the variable type of the function's return value. This must be followed by one letter per parameter. These parameters are passed to the function when it is called. The letter codes for variable types are:

A - Boolean integer
B - Far pointer to double
C - Far pointer to string
D - Far pointer to a byte string (1 byte count, followed by the string)
E - Double
F - Far pointer to a 256 byte string
G - Far pointer to a 256 byte string
H - Unsigned integer
I - Integer
J - Unsigned long integer
L - Far pointer to word (Word Far *)
M - Far pointer to array or string (CHAR FAR * FAR *)
N - Far pointer to array or words (Word Far *)
O - Far pointer to a long (LONG FAR *)

DLL functions are extremely powerful tools that provide access to Windows functions and to specifically written functions. They are for the benefit of experienced Windows programmers.

Before using these functions, make sure that you know how to use them correctly. If they are used incorrectly, you may have problems with Ami Pro and Windows after their use.

Syntax

DLLLocate(Module, Function, Parameters)

Module is the file name of the Dynamic Link Library module.

Function is the name of the function within the DLL module to be accessed.

Parameters is a string with the prototype parameters and return code for the function.

Return Value

A positive ID number for the routine if the routine was located and the parameter prototype string is correct.

0 (FALSE) if the other DLL module was not found or if the parameter string is incorrect. The ID is used by the DLLCall function to execute the routine.

Example

```
FUNCTION Example()
Text = Query$("Enter text for window:")  'Get the Title Text
hGetActiveWindow = DLLLocate("User", "GetActiveWindow", "H")
' Find the handle to the DLL
hWnd = DLLCall(hGetActiveWindow)  'and get the active window handle
hSetActiveText = DLLLocate("User", "SetWindowText", "IHC")
' Find the handle to the DLL
DLLCall(hSetActiveText, hWnd, Text)  'and set the text through the DLL
END FUNCTION
```

See also:

This function enters new values in the current document's Document Info. Choosing this function is equivalent to choosing File/Doc Info. The DocInfo function does not allow the user to examine the system fields in the document information, nor does it allow the macro to determine their values.

Syntax

DocInfo(Description, Keyword, Flag)

Description is the text that replaces the contents of the Description field in the document description.

Keyword is the text that replaces the contents of the Keyword field in the document description.

Flag indicates whether or not a document is locked. It also indicates whether a frame macro should be executed when the user clicks on a frame. It is one of the following values:

DocResetFlag (0) - Do not lock the document and do not run frame macros

DocRunMacro (1) - Enables you to run frame macros

DocLock (2) - Locks out all editing functions except notes

DocRevLock (4) - Enables you to toggle the locking of revision marking

Lock does not take effect until the document is saved. The notes functions available are editing, inserting, and deleting.

To display the Document Info dialog box and allow the user to examine and fill out the document description: **DocInfo**

Return Value

1 (TRUE) if the new document info fields were saved.

0 (UserCancel) if the user canceled the function.

Example

```
FUNCTION Example()
descrip = "This is my doc description"
keywords = "macro, docinfo"
flag = DocRunMacro + DocLock
DocInfo(descrip, keywords, flag)
END FUNCTION
```

See also:

RenameDocInfoField GetDocInfo$ InsertDocInfo DocInfoFields InsertDocInfoField

This function replaces the contents of the text in a user field. The user fields are located in the document information. Choosing this function is equivalent to choosing File/Doc Info/Other Fields.

Syntax

DocInfoFields(Field1, Field2, Field3, Field4, Field5, Field6, Field7, Field8)

field1 is the text that replaces the contents of user field 1.

field2 is the text that replaces the contents of user field 2.

field3 is the text that replaces the contents of user field 3.

field4 is the text that replaces the contents of user field 4.

field5 is the text that replaces the contents of user field 5.

field6 is the text that replaces the contents of user field 6.

field7 is the text that replaces the contents of user field 7.

field8 is the text that replaces the contents of user field 8.

Return Value

1 (TRUE) if the info fields were successfully changed.

0 (UserCancel) if the user canceled the function.

-2 (GeneralFailure) if the info fields were not changed.

Example

```
FUNCTION Example()
Dim Field(8)
For t = 1 to 8
    Field(t) = Query$("Enter information for field({t})")
Next
DocInfoFields(Field(1), Field(2), Field(3), Field(4), Field(5), Field(6), Field(7), Field(8))
END FUNCTION
```

See also:

RenameDocInfoField GetDocInfo$ InsertDocInfo InsertDocInfoField DocInfo

This function compares two documents. Choosing this function is equivalent to choosing Tools/Doc Compare.

Syntax

DocumentCompare(FileName)

FileName is the name of the file to be compared. The file name must include the path if the file is not in the current or document directory.

Return Value

1 (TRUE) if the documents were successfully compared.

0 (UserCancel) if the user canceled the function.

-2 (GeneralFailure) if the documents were not successfully compared OR IF AN ERROR OCCURRED.

Example

```
FUNCTION Example()
Filename1 = Query$("Please enter the name of the first document (Enter for current):")
Filename2 = Query$("Please enter the name of the second document:")
IF Filename1 != ""
    FileOpen(Filename1, 1, "")
ENDIF
DocumentCompare(Filename2)
END FUNCTION
```

See also:

RevisionMarking RevisionMarkOpts

This function makes the parameter the current DOS directory.
This function does not change the drive.

Syntax

DOSchdir(Dir)

Dir is the path you wish to make the current directory.

Return Value

0 if the directory was successfully changed.

-1 if the directory could not be changed or does not exist.

Example

```
FUNCTION Example()
Dir = Query$("What directory do you want to change to?")
IF -1 = DOSchdir(Dir)
    Message("Could not change to {Dir}")
ENDIF
NewDir = GetCurrentDir$()
Message("The current directory reported by DOS is: {NewDir}")
END FUNCTION
```

See also:

DOSRename DOSCopyFile DOSDelFile DOSGetFileAttr DOSmkdir DOSGetEnv$ DOSSetFileAttr DOSrmdir FileManagement

This function copies an existing file.

Syntax

DOSCopyFile(OldFile, NewFile)

OldFile is the name of the file to be copied. Use the path if the file to be copied is not in the current directory.

NewFile is the name to which the file is being copied. Use the path if you do not want the file copied to the current working directory.

Return Value

1 (TRUE) if the file was successfully copied.

0 (FALSE) if the file was not copied.

Example

```
FUNCTION Example()
OpenFile = GetOpenFileName$() ' get the open file name including path
Backupfile = "C:\curfile.bak"
DOSCopyFile(OpenFile, Backupfile) ' make a backup copy of the open file
DOSDelFile(Backupfile) ' delete the back up file
END FUNCTION
```

See also:

DOSRename DOSDelFile DOSGetFileAttr DOSmkdir DOSchdir DOSGetEnv$ DOSSetFileAttr DOSrmdir FileManagement

This function deletes an existing file.

Syntax

DOSDelFile(FileName)

FileName is the name of the file to be deleted. If the file is not in the current directory, the path must be used.

Return Value

0 if the file was deleted.

-1 (NotFound) if the file was not deleted.

Example

```
FUNCTION Example()
OpenFile = GetOpenFileName$() ' get the open file name including path
Backupfile = "C:\curfile.bak"
DOSCopyFile(OpenFile, Backupfile) ' make a backup copy of the open file
DOSDelFile(Backupfile) ' delete the back up file
END FUNCTION
```

See also:

DOSRename DOSCopyFile DOSGetFileAttr DOSmkdir DOSchdir DOSGetEnv$ DOSSetFileAttr DOSrmdir FileManagement

This function searches the DOS environment for the passed variable.

Syntax

DOSGetEnv$(Var)

Var is the name of the environment variable to search for.

Return Value

The information stored in the environment variable if the variable is found.

The null string ("") if the variable is not found.

Example

```
FUNCTION Example()
Var = "Path"
Path = DosGetEnv$(Var)
Message(Path)
END FUNCTION
```

See also:

DOSRename DOSCopyFile DOSDelFile DOSGetFileAttr DOSmkdir DOSchdir DOSSetFileAttr DOSrmdir FileManagement

This function returns the attributes for a specified file.

Syntax

DOSGetFileAttr(FileName)

FileName is the name of the file for which the attributes are determined. If the file is not in the current directory, the path must be used.

Return Value

A bit number corresponding to the attributes for the specified file. The bit values that a file can have are:

- 1 - Read-only
- 2 - Hidden
- 4 - System
- 8 - Volume ID
- 16 - Subdirectory
- 32 - Archive

These bit numbers are added together if more than one attribute is set for that file. Use the binary operator OR (|) or AND (&) to extract them.

-1 if the attributes could not be found

Example

```
FUNCTION Main()
Filledit(9001,"*.*")
Box = DialogBox(".","GetAttrib")
IF Box<>1
    EXIT FUNCTION
ENDIF
Dir = GetCurrentDir$()
Name = GetdialogField$(9001)
FileName = StrCat$(Dir,Name)
Attrib = DosGetFileAttr(Name)
Message("{Name} is set as {Attrib}")
END FUNCTION

DIALOG GetAttrib
-2134376448 4 90 34 110 85 "" "" "Get File Attributes"
FONT 8 "HELV"
4 14 52 64 9001 1352728579 "listbox" "" 0
5 3 53 10 7999 1342177280 "static" "" 0
65 5 40 14 1 1342373889 "button" "OK" 0
65 22 40 14 2 1342373888 "button" "Cancel" 0
END DIALOG
```

See also:

DOSRename DOSCopyFile DOSDelFile DOSmkdir DOSchdir DOSGetEnv$ DOSSetFileAttr DOSrmdir FileManagement

This function creates a directory.

Syntax

DOSmkdir(Dir)

Dir is the name of the directory to be created.

Return Value

0 if the directory was created.

-1 if the directory could not be created.

Example

```
FUNCTION Example()
Dir = Query$("What is the name of the new directory?")
IF -1 = DOSmkdir(Dir)
    Message("Could not create directory {Dir}")
ELSE
    Message("Directory {Dir} created.")
ENDIF
END FUNCTION
```

See also:

DOSRename DOSCopyFile DOSDelFile DOSGetFileAttr DOSchdir DOSGetEnv$ DOSSetFileAttr DOSrmdir FileManagement

This function renames an existing file.

Syntax

DOSRename(OldName, NewName)

OldName is the name of the file to be renamed. If the file is not in the current directory, the path must be used.

NewName is the name to which the file is being renamed. If you do not want this file to be placed in the current directory, the path must be used.

The NewName file name must not exist in the specified directory.

Return Value

0 if the file was renamed.

Non zero number if the file could not be renamed or if the specified new file name exists.

Example

```
FUNCTION Example()
Filename = Query$("What file (with path) do you want to perform an action on?")
IF Decide("Do you want to Copy {Filename}?")
    Filename2 = Query$("What file do you want to copy {Filename} to?")
    IF 0 = DOSCopyFile(Filename, Filename2)
        Message("Could not copy {Filename} to {Filename2}")
    ENDIF
ELSEIF Decide("Do you want to Rename {Filename}?")
    Filename2 = Query$("What is the new name for {Filename}?")
    IF 0 != DOSRename(Filename, Filename2)
        Message("Could not rename {Filename} as {Filename2}")
    ENDIF
ELSEIF Decide("Do you want to Delete {Filename}?")
    IF Decide("Are you SURE you want to Delete {Filename}?")
        IF -1 = DOSDelFile(Filename)
        Message("Could not delete {Filename}")
        ENDIF
    ENDIF
ENDIF
END FUNCTION
```

See also:

DOSCopyFile DOSDelFile DOSGetFileAttr DOSmkdir DOSchdir DOSGetEnv$ DOSSetFileAttr DOSrmdir FileManagement

This function removes an existing directory.

Syntax

DOSrmdir(Dir)

Dir is the name of the existing directory to be removed.

The directory must be empty and cannot be the current working directory or the root directory.

Return Value

0 if the directory was successfully removed.

-1 if the directory could not be removed.

Example

```
FUNCTION Example()
Dir = Query$("What is the name of the directory you want to delete?")
IF Decide("Are you SURE you want to delete {Dir}?")
    IF -1 = DOSrmdir(Dir)
        Message("Could not remove {Dir}. The directory may not be empty.")
    ELSE
        Message("The directory {Dir} has been removed.")
    ENDIF
ENDIF
END FUNCTION
```

See also:

DOSRename DOSCopyFile DOSDelFile DOSGetFileAttr DOSmkdir DOSchdir DOSGetEnv$ DOSSetFileAttr FileManagement

This function sets the attribute bits for an existing file.

Syntax

DOSSetFileAttr(FileName, Attr)

FileName is the name of the file for which the attribute bits are to be set. If the file is not in the current directory, the path must be used.

Attr is the number of the attribute to be set. The bit values that a file can have are:

- 1 - Read-only
- 2 - Hidden
- 4 - System
- 8 - Volume ID
- 16 - Subdirectory
- 32 - Archive

To set more than one attribute, add the bit values for the attributes you want to set and use that number.

Return Value

0 if the attributes were successfully set.

Example

```
FUNCTION Example()
Filledit(9001,"*.*")
Box = DialogBox(".","GetAttrib")
IF Box<>1
    Exit Function
ENDIF
Dir = GetCurrentDir$()
Name = GetdialogField$(9001)
FileName = StrCat$(Dir,Name)
IF 0 = DosSetFileAttr(Name,1)
    Message("{Name} is set as Read-Only")
ENDIF
END FUNCTION

DIALOG GetAttrib
-2134376448 4 90 34 110 85 "" "" "Get File Attributes"
FONT 8 "HELV"
4 14 52 64 9001 1352728579 "listbox" "" 0
5 3 53 10 7999 1342177280 "static" "" 0
65 5 40 14 1 1342373889 "button" "OK" 0
65 22 40 14 2 1342373888 "button" "Cancel" 0
END DIALOG
```

See also:

DOSRename DOSCopyFile DOSDelFile DOSGetFileAttr DOSmkdir DOSchdir DOSGetEnv$ DOSrmdir FileManagement

This function changes the current Multiple Document Interface (MDI) document in Ami Pro from Layout Mode or Outline Mode to Draft Mode. Choosing this function is equivalent to choosing View/Draft Mode.

Syntax

DraftMode()

Return Value

1 (TRUE) if the view mode was changed.

0 (NoAction) if no action was taken because the document was already in Draft Mode.

Example

```
FUNCTION Example()
DraftMode()
Mode = GetMode()
Message("Mode = {Mode}")
END FUNCTION
```

See also:

EnlargedView FacingView FullPageView GetMode GetViewLevel LayoutMode OutlineMode StandardView

This function initiates the drawing function, bringing the drawing menu and icon bar onto the screen. Choosing this function is equivalent to choosing Tools/Drawing.

Syntax

DrawingMode()

Return Value

1 (TRUE) if the drawing mode was initiated.

-2 (GeneralFailure) if the drawing mode could not be initiated.

Example

```
FUNCTION Example()
LayoutMode()
IF not IsFrameSelected()
    ' get the cursor position and place a frame there
    Pos = CursorPosition$()
    x = strfield$(Pos, 1, ",")
    y = strfield$(Pos, 2, ",")
    AddFrame(x, y, (x + 1440), (y - 1440))
ENDIF
DrawingMode()
END FUNCTION
```

See also:

ChartingMode AddFrame AddFrameDLG GraphicsScaling

This function displays the edit formula dialog box. Choosing this function is equivalent to choosing Table/Edit Formula. This function does not automatically allow a macro to insert a formula into a cell. To insert a formula automatically, use the SetFormula function.

Syntax

EditFormula()

Return Value

1 (TRUE) if the dialog box is displayed.

0 (UserCancel) if the user canceled the function.

Example

```
FUNCTION Example()
OnKey([CtrlAlte], EditFormula, 0)
Message("Press Ctrl+Alt+E to edit the formula in a table cell")
END FUNCTION
```

See also:

SetFormula

This function scrolls the document to a new position in the file without moving the insertion point. Choosing this function is equivalent to dragging the scroll box on the horizontal scroll bar left or right. This function may be used only through macro keystroke record. The parameter for this function is generated by Windows and can not be selected from a list of values.

Syntax

ElevatorLeftRight()

Return Value

This function does not return a value.

Example

```
FUNCTION Example()
ElevatorLeftRight
END FUNCTION
```

See also:

CharLeft CharRight ElevatorUpDown EndOfFile LeftEdge LineDown LineUp RightEdge ScreenDown ScreenLeft ScreenRight ScreenUp TopOfFile

This function scrolls the document to a new position in the file without moving the insertion point. Choosing this function is equivalent to dragging the scroll box on the vertical scroll bar up or down. This function may be used only through macro keystroke record. The parameter for this function is generated by Windows and can not be selected from a list of values.

Syntax

ElevatorUpDown()

Return Value

1 (TRUE) if the document was able to scroll.

-2 (GeneralFailure) if the document could not be scrolled.

Example

```
FUNCTION Example()
ElevatorUpDown
END FUNCTION
```

See also:

CharLeft CharRight ElevatorLeftRight EndOfFile LeftEdge LineDown LineUp RightEdge ScreenDown ScreenLeft ScreenRight ScreenUp TopOfFile

This function scrolls the document to the end of the file without moving the insertion point. Choosing this function is equivalent to dragging the scroll box on the vertical scroll bar to the extreme bottom.

Syntax

EndOfFile()

Return Value

1 (TRUE) if the document was able to scroll.

-2 (GeneralFailure) if the document could not be scrolled.

Example

```
FUNCTION Example()
Action = Query$("Move to (T)op of file, or (E)nd of file?")
Action = UCASE$(Left$(Action, 1))
SWITCH Action
CASE "T"
message(TopOfFile())
CASE "E"
message(EndOfFile())
default
Message("""T"" or ""E"" will do just fine, please.")
ENDSWITCH
END FUNCTION
```

See also:

CharLeft CharRight LeftEdge LineDown LineUp RightEdge ScreenDown ScreenLeft ScreenRight ScreenUp TopOfFile

This function displays the Windows Help for enhancement products. Choosing this function is equivalent to choosing Help/Enhancements.

Syntax

EnhancementProducts()

Return Value

This functions returns:

1 (TRUE) if the Enhancement Products Help window is displayed.

-2 (GeneralFailure) if the Enhancement Products Help window is not displayed.

-6 (NoMemory) if the function failed because of insufficient memory.

Example

```
FUNCTION Example()
EnhancementProducts()
END FUNCTION
```

See also:

About Help HowDoIHelp KeyboardHelp MacroHelp UpgradeHelp UsingHelp

This function changes the current view level to enlarged view. Choosing this function is equivalent to choosing View/Enlarged View.

Syntax

EnlargedView()

Return Value

This function returns 1.

Example

```
FUNCTION Example()
Message("Now switching to Enlarged View")
EnlargedView()
END FUNCTION
```

See also:

FacingView FullPageView GetViewLevel LayoutMode StandardView CustomView ChartingMode DrawingMode AddFrame AddFrameDLG

This function changes to equation mode. Choosing this function is equivalent to choosing Tools/Equations. If a frame is selected, it is used. If no frame is selected, one is created.

This function does not automatically enter equations.

Syntax

Equations()

Return Value

1 (TRUE) if the equations mode was initiated.

-2 (GeneralFailure) if the mode could not be initiated.

Example

```
FUNCTION Example()
OnKey([Ctrlshifte], Equations, 0)
Message("Press Ctrl+Shift+E to activate the equation editor.")
END FUNCTION
```

See also:

ChartingMode DrawingMode AddFrame AddFrameDLG OnKey

This function passes a power field to the field manager for evaluation and then returns the result.

Syntax

EvalField(Field)

Field is the power field you wish to evaluate.

Return Value

The result of the specified power field.

Example

```
FUNCTION Example()
NumberOfPages = EvalField("NumPages")
IF NumberOfPages > 1   'make plural
    Adv = "are"
    s = "s"
ELSE               'make singular
    Adv = "is"
    s = ""
ENDIF
Message("There {Adv} {NumberOfPages} page{s} in this document.")
END FUNCTION
```

See also:

FieldAdd FieldAuto FieldCommand FieldEvaluate FieldLock FieldNext FieldPrev FieldRemove FieldToggleDisplay FieldUpdate FieldUpdateAll

This function executes a program. This function uses Microsoft Windows to start the new program. Choosing this function is equivalent to choosing Run from the Program Manager menu.

Syntax

Exec(App, Parameters[, State])

App is the name of the program, including extension, which should be executed. If the desired program has any parameters, they should be passed in **Parameters**. If the file to be executed is not in the path, you must supply the path. If the program has no parameters, a null string("") must be used.

State is a number corresponding to the state that the window (if it is a Windows application) should be displayed. This parameter can be one of the following:

0 - Exec hidden
1 - Exec and show the window normally
2 - Exec the program minimized
3 - Exec the program maximized
4 - Exec normally, but do not make active
5 - Exec normally
6 - Exec program minimized in the background
7 - Exec the program minimized and do not activate it
8 - Exec the program and display it normally
9 - Exec the program and display it restored

Return Value

A positive number greater than 32 if the program was executed successfully.

Example

```
FUNCTION Example()
Windir = GetWindowsDirectory$() ' get the current windows directory
' run the Cardfile program and open names.crd
Exec("{Windir}Cardfile.exe", "Names.crd", 3)
END FUNCTION
```

See also:

ActivateApp DDEExecute DDEInitiate GetMacPath$ Appclose AppGetAppCount AppGetAppNames AppGetWindowPos AppHide AppIsRunning AppMaximize AppMinimize AppMove AppRestore AppSendMessage AppSize

This function changes the current view level to facing pages View. Choosing this function is equivalent to choosing View/Facing Pages. You can use the GetMode function to determine the current view mode.

You must be in Layout Mode to call this function.

Syntax

FacingView()

Return Value

This function returns 1.

Example

```
FUNCTION Example()
Mode = GetMode()
IF Mode != 1
    ' change to layout mode if not there
    LayoutMode()
ENDIF
FacingView()
END FUNCTION
```

See also:

EnlargedView FullPageView GetViewLevel LayoutMode StandardView CustomView

This function extracts text formatting information from selected text and then applies that formatting to other main document text, table text, or text in a text frame. Text formatting is any typeface, point size, color, attribute (bold, italic, underline, word underline), capitalization, or special effects. This function enables Fast Format if it is not enabled or disables it if it is enabled. Choosing this function is equivalent to choosing Text/Fast Format.

This function can be used to load the text formatting for use with the ApplyFormat function.

Syntax

FastFormat()

Return Value

This function returns 1.

Example

```
FUNCTION Example()
FastFormat()     ' Enable Fast Formatting
UserControl("Press Resume to cancel Fast Formatting")
FastFormat()     ' Disable Fast Formatting
END FUNCTION
```

See also:

ApplyFormat

This function closes an ASCII file that has been opened by the fopen function. All open files must be closed by the macro that created them, otherwise unpredictable results occur.

When using ASCII file functions, be careful to ensure that the file was opened correctly before performing additional ASCII file functions. If the fopen function was unable to open the desired file, unpredictable results can occur if the macro attempts to read or write to that file.

Syntax

fclose(Handle)

Handle is the file handle of the file to close.

Return Value

This function does not return a value.

Example

```
FUNCTION Example()
amidir = GetAmiDirectory$()
' opens for appending; creates this file if it doesn't exist
id = fopen("{amidir}test.txt", "a")
IF id = 0
    exit function
ENDIF
Name=Query$("What is your name?")
fputs(id, Name)
Address= Query$("What is your address?")
fwrite(id, Address)
fwrite(id, "This is right after the address, with no carriage return.")
fclose(id)
Exec("NOTEPAD.EXE", "{amidir}TEST.TXT")
END FUNCTION
```

See also:

fgets$ fopen fputs fseek ftell fwrite fread

This function reads a line of text from the file opened by the fopen function. The line of text is read from the current file pointer position. When reading a line of text, the carriage return/line feed combination is stripped from the end of the line before the line is returned by the function.

When using ASCII file functions, be careful to ensure that the file was opened correctly before performing additional ASCII file functions. If the fopen function was unable to open the desired file, unpredictable results can occur if the macro attempts to read or write to that file.

Syntax

fgets$(Handle)

Handle is the file handle for the file returned by the fopen function.

Return Value

The line of text if the file pointer was not at the end of the file.

-1 if the file pointer was at the end of the file.

Example

```
FUNCTION Example()
ONERROR toast
ONCANCEL toast
DEFSTR id, Line;
WinDir = GetWindowsDirectory$()   'get the windows directory
' open the PRINTERS.TXT file in memory
IF 0 != Assign(&id, fopen("{WinDir}PRINTERS.TXT", "r"))
    ' create a new file
    New("_BASIC.STY", 0, 0)
    WHILE -1 != Assign(&Line, fgets$(id))
        TYPE("{Line}[Enter]")  'type all the text in
    WEND
    toast:
    fclose(id)
ELSE
    Message("Could not open {WinDir}PRINTERS.TXT!")
ENDIF
END FUNCTION
```

See also:

fclose fopen fputs fseek ftell fwrite fread

This function adds a power field. Choosing this function is equivalent to choosing Edit/Power Fields/Insert.

Syntax

FieldAdd(Field)

Field is the name of the field to add. To include a quote mark within the string, use "".

Return Value

The new ID if the power field was successfully added.

-2 (GeneralFailure) if the Power field was not added.

Example

```
FUNCTION Example()
' add a power field that looks like <TOC 1 "Related Functions">
FieldAdd("TOC 1 ""Related Functions""")
' add a power field that looks like <Index "experience" #>
FieldAdd("Index ""experience"" #")
' add a power field that looks like <CreateDate %DB>
FieldAdd("CreateDate %DB")
END FUNCTION
```

See also:

FieldAuto FieldCommand FieldEvaluate FieldLock FieldNext FieldPrev FieldRemove FieldToggleDisplay FieldUpdate FieldUpdateAll

This function sets or clears the auto update bit for a power field. Choosing this function is equivalent to choosing Edit/Power Fields/Insert/Auto run.

Syntax

FieldAuto(ID, Flag)

ID is the ID for the power field to modify.

Flag is the value of the update bit for the field and is either a 1 (On) or a 0 (Off).

Return Value

1 (TRUE) if the bit was set or cleared.

0 (NoAction) if no action was taken.

Example

```
FUNCTION Example()
FieldCommand()     ' let the user make a power field
both = FieldPrev
id = MOD(both, 0x10000)    'see FieldNext/FieldPrev for explanation
FieldAuto(id, 1)
END FUNCTION
```

See also:

FieldAdd FieldCommand FieldEvaluate FieldLock FieldNext FieldPrev FieldRemove FieldToggleDisplay FieldUpdate FieldUpdateAll

This function brings up the Insert Power Fields dialog box. Choosing this function is equivalent to choosing Edit/Power Fields/Insert.

Syntax

FieldCommand()

Return Value

This function does not return a value.

Example

```
FUNCTION Example()
FieldCommand()     ' let the user make a power field
both = FieldPrev
id = MOD(both, 0x10000)    'see FieldNext/FieldPrev for explanation
FieldAuto(id, 1)
END FUNCTION
```

See also:

FieldAdd FieldAuto FieldEvaluate FieldLock FieldNext FieldPrev FieldRemove FieldToggleDisplay FieldUpdate FieldUpdateAll

This function updates the field in the current location of the insertion point. Choosing this function is equivalent to choosing Edit/Power Fields/Update.

Syntax

FieldEvaluate()

Return Value

1 (TRUE) if the field was updated.

0 (FALSE) if the field was not updated.

Example

```
FUNCTION Example()
FieldNext()        ' go to the next power field
FieldEvaluate()    ' update it
END FUNCTION
```

See also:

FieldAdd FieldAuto FieldCommand FieldLock FieldNext FieldPrev FieldRemove FieldToggleDisplay FieldUpdate FieldUpdateAll

This function sets or clears the lock bit for a power field. Choosing this function is equivalent to choosing Edit/Power Fields/Insert/Lock.

Syntax

FieldLock(ID, Flag)

ID is the ID of the power field to modify.

Flag is the value of the lock bit for the field and is either a 1 (On) or a 0 (Off).

Return Value

This function does not return a value.

Example

```
FUNCTION Example()
FieldCommand()     ' let the user make a power field
both = FieldPrev
id = MOD(both, 0x10000)    'see FieldNext/FieldPrev for explanation
FieldLock(id, 1)
END FUNCTION
```

See also:

FieldAdd FieldAuto FieldCommand FieldEvaluate FieldNext FieldPrev FieldRemove FieldToggleDisplay FieldUpdate FieldUpdateAll

These functions go backward to the previous power field or forward to the next power field. Choosing this function is equivalent to choosing Edit/Power Fields/Next Field or Edit/Power Fields/Prev Field.

Syntax

FieldNext()
FieldPrev()

Return Value

0 (NoAction) if no action was taken.

The ID and Type of the next/previous Power Field in a bit number. To extract the ID and Type:

both = FieldNext() ' or both = FieldPrev()

ID = Mod(both, 0x10000)

Type = Round(both / 0x10000)

Example

```
FUNCTION Example()
FieldCommand()
both = FieldPrev   ' or use FieldNext here
id = MOD(both, 0x10000)
FieldAuto(id, 1)
FieldLock(id, 1)
END FUNCTION
```

See also:

FieldAdd FieldAuto FieldCommand FieldEvaluate FieldLock FieldRemove FieldToggleDisplay FieldUpdate FieldUpdateAll

This function removes the power field identified by the ID number.

Syntax

FieldRemove(ID, 8)

ID identifies the power field and may be obtained by using the FieldNext or FieldPrev functions.

8 tells the function to remove the index entry.

Return Value

This function returns 1.

Example

```
FUNCTION Example()
Both = FieldNext()
id = MOD(Both, 0x10000)       ' see Field Next/Field Prev for explanation
FieldRemove(id, 8)
END FUNCTION
```

See also:

FieldAdd FieldAuto FieldCommand FieldEvaluate FieldLock FieldNext FieldPrev FieldToggleDisplay FieldUpdate FieldUpdateAll

This function toggles the field display on or off. Choosing this function is equivalent to choosing View/Show/Hide Power Fields.

Syntax

FieldToggleDisplay()

Return Value

This function returns 1.

Example

```
FUNCTION Example()
UserControl("Click Resume to show power fields.")
FieldToggleDisplay()
END FUNCTION
```

See also:

FieldAdd FieldAuto FieldCommand FieldEvaluate FieldLock FieldNext FieldPrev FieldRemove FieldUpdate FieldUpdateAll

This function updates an existing power field. Choosing this function is equivalent to choosing Edit/Power Fields/Update.

Syntax

FieldUpdate(ID, Type, Field)

ID is the ID of the power field to update.

Type is the type of the power field to update.

Field is the name of the power field to update.

Return Value

1 (TRUE) if the Power field was successfully updated.

-7 (CouldNotFind) if the Power Field could not be located.

Example

```
FUNCTION Example()
id = FieldAdd("NumPages")      ' add Numpages power field
both = FieldPrev()
id = MOD(both, 0x10000)        ' see Field Next/Field Prev for explanation
type = Round(both/0x10000)
FieldUpdate(id, type, "NumPages")     ' updates the Numpages power field
END FUNCTION
```

See also:

FieldAdd FieldAuto FieldCommand FieldEvaluate FieldLock FieldNext FieldPrev FieldRemove FieldToggleDisplay FieldUpdateAll

This function updates all fields in the current document. Choosing this function is equivalent to choosing Edit/Power Fields/Update All.

Syntax

FieldUpdateAll()

Return Value

1 (TRUE) if all of the fields were successfully updated.

Example

```
FUNCTION Example()
UserControl("Click Resume to update all power fields.")
FieldUpdateAll()
END FUNCTION
```

See also:

FieldAdd FieldAuto FieldCommand FieldEvaluate FieldLock FieldNext FieldPrev FieldRemove FieldToggleDisplay FieldUpdate

This function reads and optionally sets the file-changed flag internal to Ami Pro. The file-changed flag determines whether Ami Pro asks the user to save the document before displaying a new document or exiting the program.

Syntax

FileChanged(Action, NewValue)

Action determines whether to read or set the file changed flag. If the Action parameter is 1 (TRUE), the state of the flag changes to the second parameter. If the Action parameter is 0 (FALSE), the parameter is not changed and NewValue is ignored.

NewValue is the new value used for the file changed flag. This should be set to a 1 (changed) or a 0 (not changed).

Return Value

A positive number if the file had been changed prior to using the function.

0 (FALSE) if the file had not been changed.

Changing the value of the file changed parameter does not affect the return value of the function until the subsequent function call.

Example

```
FUNCTION Example()
TYPE("After typing this text, the file will be marked unchanged and then closed.")
FileChanged(1, 0)
FileClose()
END FUNCTION
```

See also:

FileOpen Save SaveAs FilePrint FileClose

This function closes the current Multiple Document Interface (MDI) window in Ami Pro. Choosing this function is equivalent to choosing File/Close.

Syntax

FileClose()

Return Value

1 (TRUE) if the file was successfully closed.

-2 if the file was not closed.

Example

```
FUNCTION Example()
FileClose()    'Close the current file.
END FUNCTION
```

See also:

FileOpen FileChanged FilePrint Save SaveAs

This function opens the Ami Pro File Manager. Choosing this function is equivalent to choosing File/File Management. File Management functions cannot be run directly with this macro function. It opens the Ami Pro File Manager, then immediately returns control to the macro. The user can do file management functions, close the window, and return to word processing.

If later functions in the macro involve screen display, these functions cause the Ami Pro window to obscure the Ami Pro File Management window as the macro continues to run. Either have the macro pause with a UserControl box or allow this function to be the last function in the macro.

Syntax

FileManagement()

Return Value

1 (TRUE) if the File Manager was started.

-2 (GeneralFailure) if the File Manager could not be started.

-6 (NoMemory) if the function failed because of insufficient memory.

Example

```
FUNCTION Example()
OnKey([CtrlAltf], FileManagement, 0)
Message("Press Ctrl+Alt+F to execute the Ami Pro File Manager.")
END FUNCTION
```

See also:

ControlPanel DOSRename DOSCopyFile DOSDelFile DOSGetFileAttr DOSmkdir DOSchdir DOSGetEnv$ DOSSetFileAttr DOSrmdir OnKey

This function opens a document in Ami Pro. Choosing this function is equivalent to choosing File/Open.

Modified in 3.0 The Options parameter has a new value 5.

Syntax

FileOpen(FileName, Options, App)

FileName is the name of the document to open. If the file to be opened is not in the current directory or document directory, the path must be used. To open the 'Untitled' file, use the null string ("") as the file name.

Options is a bit number corresponding to the options for the file you open. To use more than one of the options, add them together. Options can be one or more of the following:

1 - Ami Pro file
5 - ASCII file
8 - Import/Insert
16 - Non-Ami Pro or Non-ASCII file
128 - Close current file

App is the file type of the file. The file type must appear as it does in the AMIPRO.INI file or the list of file types in the Open dialog box. To open an Ami Pro file, use the null string ("") here.

To display the Open dialog box and allow the user to choose the file to open: **FileOpen**

Return Value

1 (TRUE) if the file was opened.

0 (UserCancel) if the user canceled the function.

-2 (GeneralFailure) if the file was not opened.

2 if the user created a file that did not exist.

Example

```
FUNCTION Example()
Filename = Query$("What file do you want to open?")
FileOpen("{Filename}", 1, "")
END FUNCTION
```

See also:

GetOpenFileName$ New SaveAs FileChanged FilePrint Save GetOpenFileCount GetOpenFileNames

This function allows printing the current document to a printer. Choosing this function is equivalent to choosing File/Print. Before using this function, your macro should ensure that the document you want to print is currently active.

The options for setting the sheet feeder bin cannot be set through this command. In order to set the sheet feeder bins, you need to use the PrintOptions function. If the PrintOptions function is not used, Ami Pro uses the bins specified in the Control Panel.

Modified in 3.0 The Flag parameter has a new value 4096, which prints the current page.

Syntax

FilePrint(Copies, StartPage, EndPage, Flag)

Copies is the number of copies of the document to print.

StartPage is the page number to start printing. This parameter is ignored if the PrintAll option is used, but must still be present.

EndPage is the page number after which to stop printing. This parameter is ignored if the PrintAll option is used, but must still be present.

Flag is a number that defines the value of other print options. This parameter defines which of the optional parameters are used and may be one or more of the following:

PrintAll (1) - Prints all pages of document, ignoring StartPage and EndPage.

Reverse (2) - Prints the document in reverse order.

Collate (4) - Collate multiple copy output.

CropMarks (16) - Print the document with crop marks.

PrtDocInfo (32) - Print the document description along with the document.

PrePrinted (64) - Print the document on preprinted forms. Do not print protected text.

PrintUpdateFields (128) - Updates power fields before printing.

PrintNotes (256) - Prints the document with notes.

PrintEven (512) - Prints only the even pages of the document.

PrintOdd (1024) - Prints only the odd pages of the document.

PrintBoth (1536) - Prints both even and odd pages of the document.

PrintNoPictures (2048) - Prints the document without pictures. To print with pictures do not use this value.

4096 - Prints only the current page.

The desired options should be added together to make up the value of the flag parameter.

To display the Print dialog box and allow the user to set print options: **FilePrint**

Return Value

1 (TRUE) if the print job was successfully completed.

0 (UserCancel) if the user canceled the function.

-2 (GeneralFailure) if the print job failed.

Example

```
FUNCTION Example()
' print all of the current file with defaults
FilePrint(1, 1, 9999, 1537)
END FUNCTION
```

See also:

ControlPanel Merge PrintSetup PrintOptions

This function is used to fill the contents of an object that display in a dialog box called by the DialogBox function. If the object is an edit box or static text with an 8000-8999 ID, the text specified in the value parameter fills the edit box. If the object is a radio button or check box, the expression specified in the value parameter is evaluated and the button is activated if the result of the evaluation is TRUE. If the result of the evaluation is FALSE, the button is not activated. If the object is a list box or a combo box (ID of 9000-9999), the text specified as the value parameter is passed. An array can be passed to a list or combo box, filling the list with the contents of the array. To do so, use indirection (&) and the name of the array to pass.

You can use this function to put a wildcard pattern into a combo box or a list box or to append other selection criteria.

Syntax

FillEdit (ID, Value)

ID is the number of the object that receives the value.

Value is either a text string, an expression that evaluates to TRUE or FALSE, or the address of an existing array. It can be any combination of the following values:

1 - Read-only
2 - Hidden
4 - System
16 - Disk volume label
32 - Directories
16384 - Drives
32768 - Files of specified type only

Return Value

This function does not return a value.

Example

```
FUNCTION Example()
' 32768 + 16384 + 16 = 49168
' this displays only drives and directories
FillEdit(9001, "c:\*.*,49168")
dialogBox(".", "DirsOnly")
END FUNCTION

DIALOG DirsOnly
-2134376448 3 52 48 160 90 "" "" "DirsOnly"
32 20 60 64 9001 1352728579 "listbox" "" 0
112 4 40 14 1 1342373889 "button" "OK" 0
112 22 40 14 2 1342373888 "button" "Cancel" 0
END DIALOG
```

See also:

DialogBox FillList GetDialogField$ SetDlgCallback SetDlgItemText GetDlgItem GetDlgItemText

This function is used to fill the contents of a list box that displays in a dialog box that is called by the DialogBox function. If there is only one list box, this list box is filled. If there is more than one list box, the list box with the lowest ID is filled. Each iteration of the FillList function can only contain 80 characters. If there are more items, additional iterations of FillList can be used to add additional items. The FillList function must be used before each call to the DialogBox function.

Syntax

FillList(Item1[, Item2]...)

Item1 and **Item2** are strings that should be placed in the list box.

Return Value

This function does not return a value.

Example

```
FUNCTION Example()
FillList("Charlie","David","Eliza")
FillList("Pete")          ' Add another name
FillList("Peyton","Cassie")       ' Add two more
Box = DialogBox(".","Example")
If Box <>1
    Exit Function         ' User hit cancel
Endif
Name = GetDialogField$(9000)  ' Get the info from the 9000 field
Message(Name,"Show Name")
End Function

DIALOG Example
-2134376448 4 99 55 111 69 "" "" "List of Names"
FONT 8 "Helv"
67 4 40 14 1 1342373889 "button" "OK" 0
67 21 40 14 2 1342373888 "button" "Cancel" 0
7 17 52 46 9000 1352728579 "listbox" "" 0
7 6 40 10 1000 1342177280 "static" "Names:" 0
END DIALOG
```

See also:

DialogBox FillEdit GetDialogField$ SetDlgCallback SetDlgItemText GetDlgItem GetDlgItemText

This function finds files that match the FileSpec given to the function. All files or a subset of files in a directory can be found. The FindFirst$ function retrieves the first matching file and must be used before the FindNext$ function. FindNext$ finds other files that match the FileSpec.

Syntax

FindFirst$(FileSpec, Attr)

FileSpec describes the file specification desired. It may include a drive and/or directory name, if desired. The file name portion of the FileSpec can include the wildcard characters asterisk (*) and question mark (?).

Attr is a number that represents the type of file desired. The possible attributes are:

A_Normal (0) - Normal Files
A_ReadOnly (1) - Read-only Files
A_Hidden (2) - Hidden Files
A_System (4) - System Files
A_Vollabel (8) -Disk's Volume Label
A_Directory (16) - Directories
A_Archive (32) - Archived files

The values listed above may be added together to retrieve different file types.

Return Value

A string with the matching file.

The null string ("") if no files are matched.

Example

```
FUNCTION Example()
Filename = Query$("What file do you want to open?")
' check to see if the file exists
IF FindFirst$(Filename, 0)
    FileOpen("{Filename}", 1, "")
ELSE
    Message("Could not find {Filename}.")
ENDIF
END FUNCTION
```

See also:

findNext$ fclose fgets$ fputs fseek ftell fread fwrite

This function finds files that match the FileSpec given to the FindFirst$ function. All files or a subset of files in a directory can be found. The FindFirst$ function retrieves the first matching file and must be used before the FindNext$ function. FindNext$ finds other files that match the FileSpec.

Syntax

FindNext$()

Return Value

A string with the matching file.

The null string ("") if no files are matched.

Example

```
FUNCTION Example()
DIR = Query$("What directory to report on?")
' see if the file exists
File = FindFirst$("{DIR}*.*", 0)
' Type the file names
WHILE File <> ""
    TYPE ("{File}[Enter]")
    File = FindNext$()
WEND
END FUNCTION
```

See also:

FindNext$ fclose fgets$ fputs fseek ftell fread fwrite

This function allows the user to view the Find & Replace dialog box. Choosing this function is equivalent to choosing Edit/Find & Replace. This function does not automatically replace words. If automatic Find & Replace is desired, use the Replace function instead. A macro must be edited to insert this non-recordable function.

Syntax

FindReplace()

Return Value

1 (TRUE) if the Find & Replace function was started.

0 (UserCancel) if the user canceled the function.

Example

```
FUNCTION Example()
FindReplace()
END FUNCTION
```

See also:

GoToCmd GoToShade Replace

This function adds or removes floating headers and floating footers in a document. Choosing this function is equivalent to choosing Page/Header/Footer and choosing Floating Header/Footer.

You must be in Layout Mode to use this function.

Syntax

FloatingHeader(Function)

Function is the desired function for adding or removing a floating header or footer. The function implemented depends on the Function parameter, according to the following list:

AddHeader (1) - Add a floating header

AddFooter (2) - Add a floating footer

DelHeader (4) - Delete a floating header

DelFooter (8) - Delete a floating footer

The desired header/footer type should be combined with one of the following parameters if you want to add or remove an odd or even header/footer:

OddHF (16) - Odd Header or Footer

EvenHF (32) - Even Header or Footer

To display the Floating Header/Footer dialog box and allow the user to determine the header/footer function desired: **FloatingHeader**

Return Value

1 (TRUE) if the header or footer was created or removed.

0 (UserCancel) if the user canceled the function.

-6 (NoMemory) if the function failed because of insufficient memory.

Example

```
FUNCTION Example()
Again:
Action = Query$("(A)dd or (R)emove a floating header?")
' get the first character and change it to lowercase
Action = lcase$(Left$(Action, 1))
SWITCH Action
    CASE "a"
    FloatingHeader(1)
    CASE "r"
    FloatingHeader(4)
    default
    Message("Please type in an ""A"" or an ""R""."")
    GoTo again
ENDSWITCH
END FUNCTION
```

See also:

GoToCmd PageNumber ModifyLayout HeaderFooter

This function sets the font, family, color, and size to be used for selected text. Choosing this function is equivalent to choosing Text/Font.

Syntax

FontChange(Name, PitchFamily, Color, Size)

Name is the name of the font to use. The name of the font is the font name, as seen in the Font list box in the menu.

PitchFamily is a combination of the pitch of the font and the typestyle parameters. Pitch is one of the following values:

False (0) - Not Specified
FixedPitch (1) - This is a fixed pitch font
VariablePitch (2) - This is a proportional font

Family is one of the following typeface families:

False (0) - Not Specified
Roman (16) - Roman-like typestyle; proportional pitch, serif
Swiss (32) - Swiss-like typestyle; proportional pitch, sans serif
Modern (48) - Modern-like typestyle; fixed pitch, could be either serif or sans serif
Script (64) - Script-like typestyle, such as Brush or Park Avenue
Decorative (80) - Decorative typeface, such as Old English

One value for Pitch and one value for family should be added together to make up the PitchFamily parameter. This parameter is used to determine another font if the font named is not available on the printer.

Color is a numeric representation of the color of the font. It is one of the following values:

White (16777215) - White
Cyan (16776960) - Light Blue
Yellow (65535) - Yellow
Magenta (16711935) - Purple
Green (65280) - Green
Red (255) - Red
Blue (16711680) - Blue
Black (0) - Black

Size is expressed in twips (1 inch = 1440 twips). An 8 point font is 160 twips. 10 point is 200 twips; 12 point is 240 twips, etc. The formula to determine twips from point size is *pointsize * 20*.

To display the dialog box to allow the user to select the desired font:

FontChange

Return Value

1 (TRUE) if the font was changed.

0 (UserCancel) if the user canceled the function.

Example

```
FUNCTION Example()
DEFSTR Name, Color, Size, Family;
' get the current font information
GetCurFontInfo(&Name, &Color, &Size, &Family)
' increase pointsize by two and calculate the twips
Size = ((Size / 20) + 2) * 20
' change font
```

```
FontChange(Name, Family, Color, Size)
END FUNCTION
```

See also:

FontRevert ModifyFont NormalText Spacing FontFaceChange FontPointSizeChange

This function changes the selected font for the active document. Choosing this function is equivalent to choosing a font from the status bar. Through the status bar, the user can only select from the fonts available on the printer. Using this function, any type of font can be requested. When it is time to display or print the font, Ami Pro selects the closest matching font to the specification given.

Syntax

FontFaceChange(Name)

Name is the name of the font to be used. The font name is selected from the list of names displayed in the FontName section of the status bar.

Return Value

1 (TRUE) if the Font Face was changed.

0 (NoAction) if no action was taken because the font was not available.

Example

```
FUNCTION Example()
FontName = Query$("Please enter a font name.")
ReturnValue = FontFaceChange(FontName)
IF ReturnValue = 0
    Message("Font ""{FontName}"" not available!")
ENDIF
END FUNCTION
```

See also:

FontChange FontPointSizeChange FontRevert ModifyFont NormalText

This function changes the point size of the selected font for the active document. Choosing this function is equivalent to choosing a point size from the status bar.

Syntax

FontPointSizeChange(Size)

Size is the point size to use. The point size is selected from the list of sizes displayed in the pointsize section of the status bar.

Return Value

This function returns 1.

Example

```
FUNCTION Example()
PointSize = Query$("Please enter a new point size.")
FontPointSizeChange(PointSize)
Message("The new point size is {PointSize}")
END FUNCTION
```

See also:

FontChange FontFaceChange FontRevert ModifyFont NormalText

This function changes the text font to the font defined by the current paragraph style. Choosing this function is equivalent to choosing Text/Font and selecting the Revert to style check box.

Syntax

FontRevert()

To display the Font dialog box and allow the user to select the font that should be used for text:
FontChange

Return Value

This function does not return a value.

Example

```
FUNCTION Example()
WHILE CurShade$() = ""
    UserControl("Select the text to revert back to the style...")
WEND
FontRevert()
END FUNCTION
```

See also:

FontChange ModifyFont NormalText FontFaceChange FontPointSizeChange

This function sets the options for footnotes, as well as allowing the user to edit, insert, or remove footnotes. Choosing this function is equivalent to choosing Tools/Footnotes.

Syntax

Footnotes(Function, Options, StartNum, Length, Indent)

Function is the desired footnote function. The function parameter is one of the following values:

InsFootnote (1) - Insert new footnote

EditFootnote (2) - Edit existing footnote

SetOptions (3) - Set footnote options only

Options are the desired footnote options. Options are one or more of the following options:

Gather (1) - Gather the notes at the end of the document

Reset (2) - Reset the footnote numbering at the end of each page

CustomLength (4) - The footnote separator line is a custom length, rather than the length of the margins

Options should be added together if more than one option is desired.

StartNum is the starting footnote number. If the Custom length option is chosen, the Length parameter should be set to the length of the line. If the length of the line is not customized, this value should be set to 0. The Indent parameter should be set to the desired indent from the left margin. If no indent is desired, the value for Indent should be 0.

Length is the length of the footnote line in twips (1 inch = 1440 twips).

Indent is the indention of the footnote line from the left margin in twips.

To determine the Length and Indent parameters, multiply the desired number of inches by 1440 to determine the value in twips.

To display the dialog box and allow the user to select the footnote functions and options: **Footnotes**

Return Value

1 (TRUE) if the footnote function was completed.

0 (UserCancel) if the user canceled the function.

-2 (GeneralFailure) if the function could not be completed.

Example

```
FUNCTION Example()
ReturnValue = FootNotes(InsFootnote, Gather, 1, 0, 0)
IF ReturnValue != 1
    SWITCH ReturnValue
    CASE 0
        Message("Footnote function was canceled.")
    CASE -2
        Message("GeneralFailure! Could not insert Footnote.")
    ENDSWITCH
ENDIF
END FUNCTION
```

See also:

GoToCmd

This function is used to open an ASCII or binary file for processing in a macro. This function must be used first when wanting to process an ASCII or binary file. A file that is opened must be closed by the fclose function before the macro ends, otherwise unpredictable results occur.

When using ASCII file functions, be careful to ensure that the file was opened correctly before performing additional ASCII file functions. If the fopen function was unable to open the desired file, unpredictable results can occur if the macro attempts to read or write to that file.

Syntax

fopen(FileName, Mode)

FileName is the name of the file to open. If the file is not in the current directory, the full path must be used.

Mode is the file mode to use and may be one of the following:

r -- Opens the file for reading. If the file is opened for reading, any attempts to write to the file fail. If the file does not exist, the fopen call fail.

w -- Opens the file for writing. If the file already exists, its contents are erased. If the file does not exist, it is created.

a -- Opens the file for appending. If the file does not exist, it is created. If the file already exists, text written to the file is written to the end of the file.

b -- Opens a binary file in conjunction with "r", "w", or "a". Also used with fread and fwrite.

t -- Opens a text file in conjunction with "r", "w", or "a". Also used with fgets$ and fputs.

Return Value

A non-zero file handle if the file was successfully opened.

0 (FALSE) if the file could not be opened.

Example

```
FUNCTION Example()
amidir = GetAmiDirectory$()
' opens for appending; creates this file if it doesn't exist
id = fopen("{amidir}test.txt", "a")
IF id = 0
    exit function
ENDIF
Name=Query$("What is your name?")
fputs(id, Name)
Address= Query$("What is your address?")
fwrite(id, Address)
fwrite(id, "This is right after the address, with no carriage return.")
fclose(id)
Exec("NOTEPAD.EXE", "{amidir}TEST.TXT")
END FUNCTION
```

See also:

fclose fgets$ fputs fseek ftell fread fwrite

This function takes a date (in seconds) from January 1, 1970, and creates a formatted string.

Syntax

FormatDate$(Date, Style)

Date is the date to be converted in seconds format.

Style is one of the formats listed below:

a - 2/18/91
b - February 18, 1991
B - FEBRUARY 18, 1991
c - 18 February 1991
C - 18 FEBRUARY 1991
d - Monday, February 18, 1991
D - MONDAY, FEBRUARY 18, 1991
e - February 18
E - FEBRUARY 18
f - Monday, February 18
F - MONDAY, FEBRUARY 18
g - 2/18
h - 2/18/1991
i - 18. February
I - 18. FEBRUARY
j - 18. February 1991
k - 1991 February 18
K - 1991 FEBRUARY 18
l - February, 1991
L - FEBRUARY, 1991

Return Value

The date converted to the selected style.

Example

```
FUNCTION Example()
Born = Query$("What is your Birthday (MM/DD/YYYY)?")
Date = FormatDate$(Now(), "h")    'current date
Time = FormatTime$(Now(), 6)  'current time
Days = DateDiff(Born, Date)   ' how many days old
TextDate = FormatDate$(Now(), "d")    'format the date
Message("It is now {Time} on {TextDate}. You are {Days} days old.")
END FUNCTION
```

See also:

FormatTime$ InsertDate Now

This function formats a number to a string, adds an optional prefix at the beginning, the specified number of digits to the right of the decimal point, and an optional suffix at the end. The decimal point character and thousands separator character are taken from the Country Settings option in the Control Panel.

Syntax

FormatNum$(Prefix, Suffix, Decimals, Number)

Prefix is a string that is placed at the beginning of the formatted number.

Suffix is a string that is placed at the end of the formatted number.

Decimals is the number of decimal places to use in the number.

Number is the number to format, without commas, prefix, or suffix.

Return Value

The string with the formatted number.

Example

```
FUNCTION Example()
Again:
Number = Query$("How much was the sale?")
Where = Query$("Where was the sale made (E)ngland or (A)merica?")
' get the first character and make lowercase
Where = lcase$(Left$(Where, 1))
IF Where = "e"
    Prefix = ""
    Suffix = "£"
ELSEIF Where = "a"
    Prefix = "$"
    Suffix = ""
ELSE
    Message("Please choose ""E"" or ""A""."")
    GoTo again
ENDIF
NewNumber = FormatNum$(Prefix, Suffix, 2, Number)
Message("The sale was for {NewNumber}.")
END FUNCTION
```

See also:

IsNumeric strcat$ strchr LCASE$ UCASE$ strfield$ MID$ LEN Instr

This function formats the Number parameter to a specified paragraph style as defined in the Style parameter.

Syntax

FormatSeq$(Number, Style)

Number is the number you want to format, without commas, prefixes or suffixes.

Style can be one of the paragraph styles listed below:

1 - 1, 2, 3...
2 - I, II, III...
3 - i, ii, iii...
4 - A, B, C...
5 - a, b, c...

Return Value

The number in the paragraph style specified.

Example

```
FUNCTION Example()
Number = Query$("Enter a number to be formatted into the Roman Numeral equivalent:", "1992")
Number = FormatSeq$(Number, 3)
Message("The Roman Numeral equivalent is ""{Number}""")
END FUNCTION
```

See also:

FormatNum$ ModifyStyle

This function takes the time (in seconds) and formats it to a specified style.

Syntax

FormatTime$(Time, Style)

Time is the time to be formatted.

Style is one of the formats listed below:

1 - 22:28
2 - 9:00AM
3 - 09:00AM
4 - 9:00A
5 - 09:00A
6 - 9:00am
7 - 09:00am
8 - 9:00a
9 - 09:00a

Return Value

The time in the format specified.

Example

```
FUNCTION Example()
Born = Query$("What is your Birthday (MM/DD/YYYY)?")
Date = FormatDate$(Now(), "h")    'current date
Time = FormatTime$(Now(), 6)  'current time
Days = DateDiff(Born, Date)   ' how many days old
TextDate = FormatDate$(Now(), "d")    'format the date
Message("It is now {Time} on {TextDate}. You are {Days} days old.")
END FUNCTION
```

See also:

FormatDate$ Now InsertDate

This function writes a line of ASCII text to the file opened by the fopen function. The line of text is written at the end of the file. When writing a line of text, a carriage return/line feed combination is added to the end of each line before it is written to the file.

When using ASCII file functions, be careful to ensure that the file was opened correctly before performing additional ASCII file functions. If the fopen function was unable to open the desired file, unpredictable results can occur if the macro attempts to read or write to that file.

Syntax

fputs(Handle, Text)

Handle is the file handle for the file returned by the fopen function.

Text is a line of text to be written to the file.

Return Value

This function does not return a value.

Example

```
FUNCTION Example()
amidir = GetAmiDirectory$()
' opens for appending; creates this file if it doesn't exist
id = fopen("{amidir}test.txt", "a")
IF id = 0
    exit function
ENDIF
Name=Query$("What is your name?")
fputs(id, Name)
Address= Query$("What is your address?")
fwrite(id, Address)
fwrite(id, "This is right after the address, with no carriage return.")
fclose(id)
Exec("NOTEPAD.EXE", "{amidir}TEST.TXT")
END FUNCTION
```

See also:

fclose fgets$ fopen fseek ftell fread fwrite

This function displays the Modify Frame Layout dialog box. Choosing this function is equivalent to choosing Frame/Modify Frame Layout. This function does not automatically modify the frame layout.

Syntax

FrameLayout()

Return Value

1 (TRUE) if the frame layout was modified.

0 (UserCancel) if the user canceled the function.

-2 (GeneralFailure) if the Frame layout was not modified.

Example

```
FUNCTION Example()
' create the frame
AddFrame(1440, -1440, 2880, -2880)
IF Decide("Do you want to choose the frame layout options?")
    FrameLayout()  'frame layout dialog box
ELSE
    FrameModInit()     'set up for changes
    Border = (1440 / 5)
    FrameModBorders(1440, 1440, 1440, 1440, Border, Border, Border, Border, 1)
    FrameModLines(20, 3, 1, 0, 16777215, 0, 0, 0, 100, 100)
    FrameModType(Opaque, 0, "")
    FrameModFinish()   'apply changes
ENDIF
END FUNCTION
```

See also:

ModifyLayout ModifyStyle FrameModInit FrameModBorders FrameModLines FrameModType FrameModFinish AddFrame AddFrameDlg FrameModColumns IsFrameSelected BringFrameToFront SendFrameToBack

This function allows you to change the size and placement of the selected frame. Choosing this function is equivalent to choosing Frame/Modify Frame Layout/Size & position.

You must call the FrameModInit function before using this function. Before the frame modifications take effect, the FrameModFinish function must be called.

Syntax

FrameModBorders(Width, Height, Top, Left, LeftMargin, TopMargin, RightMargin, BottomMargin, Units)

Width is the desired width of the frame.

Height is the desired height of the frame.

Top is the distance from the top of the page (vertical starting position).

Left is the distance from the left side of the page (horizontal starting position).

LeftMargin is the desired left margin of the frame.

TopMargin is the desired top margin of the frame.

RightMargin is the desired right margin of the frame.

BottomMargin is the desired bottom margin of the frame.

The value for all parameters except Units should be given in twips (1 inch = 1440 twips). Multiply the desired number of inches by 1440 to determine the value in twips.

Units is the unit of measure to use when using the dialog box to set the values for all other parameters. It is one of the following values:

Inches (1) - units set to inches
CM (2) - units set to centimeters
Picas (3) - units set to picas
Points (4) - units set to points

Return Value

1 (TRUE) if the frame was changed.

-2 (GeneralFailure) if the frame was not changed.

Example

```
FUNCTION Example()
' create the frame
AddFrame(1440, -1440, 2880, -2880)
IF Decide("Do you want to choose the frame layout options?")
    FrameLayout()  'frame layout dialog box
ELSE
    FrameModInit()     'set up for changes
    Border = (1440 / 5)
    FrameModBorders(1440, 1440, 1440, 1440, Border, Border, Border, Border, 1)
    FrameModLines(20, 3, 1, 0, 16777215, 0, 0, 0, 100, 100)
    FrameModType(Opaque, 0, "")
    FrameModFinish()   'apply changes
ENDIF
END FUNCTION
```

See also:

FrameModInit FrameModFinish FrameModLines AddFrame FrameModType AddFrameDLG FrameLayout BringFrameToFront FrameModColumns IsFrameSelected SendFrameToBack

This function allows you to change the columns and tabs of the selected frame. Choosing this function is equivalent to choosing Frame/Modify Frame Layout/Columns & tabs.

You must call the FrameModInit function before using this function. Before the frame modifications take effect, the FrameModFinish function must be called.

Syntax

FrameModColumns(Options, GutterLine, Color, NumCols[, Cols], NumTabs[, Tabs])

Options is a flag parameter and can have one or more of the following values:

(0) - No column balance and no line between columns
(1) - Column balance on
(2) - Line between columns

GutterLine determines the type of line between the columns. Available lines styles are:

Hairline (1) - Hairline
OnePoint (2) - One point rule
TwoPoint (3) - Two point rule
ThreePoint (4) - Three point rule
FourPoint (5) - Four point rule
FivePoint (6) - Five point rule
SixPoint (7) - Six point rule
DoubleOnePoint (8) - Parallel one point rules
DoubleTwoPoint (9) - Parallel two point rules
ThreeLines (10) - Hairline above and below a two point rule
HairBelow (11) - Hairline below a three point rule
HairAbove (12) - Hairline above a three point rule

Color is the color of the line between the columns and can be one of the following:

White (16777215) - White
Cyan (16776960) - Light blue
Yellow (65535) - Yellow
Magenta (16711935) - Purple
Green (65280) - Green
Red (255) - Red
Blue (16711680) - Blue
Black (0) - Black
DarkGray (12566463) - 90% gray scale
MediumGray (8355711) - 50% gray scale
LightGray (4144959) - 20% gray scale
VeryLightGray (1644825) - 10% gray scale

NumCols is the number of columns to use in the frame.

Cols are pairs of numbers that represent the twip offset to the left and right margin for each column. There should be a pair of offsets for each column.

NumTabs is how many tab pairs follow. The pairs are the type of tabs, followed by the offset from the left margin.

Tabs are pairs of numbers that represent the type of tab and its offset from the left margin. The tab type can be one of the following:

TabLeft (1) - Left tab

TabCenter (2) - Center tab
TabRight (3) - Right tab
TabNumeric (4) - Numeric tab

You can enter one of the following values, using the bitwise operator OR (|):

0x4000 - for dashed leaders
0x8000 - for dot leaders
0xc000 - for underline leaders

Return Value

1 (TRUE) if the lines are modified.

-2 (GeneralFailure) if the lines are not modified.

Example

```
FUNCTION Example()
AddFrame(1440 -1440 2880 -2880  )
FrameModInit()
' no column, no lines between columns, one point rule gutter line
' black line between columns, one column, no tabs
FrameModColumns(0 2 0 1 48 -176 0  )
FrameModFinish()
END FUNCTION
```

See also:

AddFrame AddFrameDlg BringFrameToFront FrameLayout FrameModBorders FrameModFinish FrameModInit FrameModLines FrameModType IsFrameSelected SendFrameToBack

This function applies all of your changes made when modifying the frame layout. This function is the last of a series of functions recorded when modifying the frame layout through Frame/Modify Frame Layout. Choosing this function is equivalent to accepting changes entered by choosing Frame/Modify Frame Layout.

You must call the FrameModInit function before you call FrameModFinish.

Syntax

FrameModFinish()

Return Value

1 (TRUE) if the frame was changed.

-2 (GeneralFailure) if the frame was not changed.

Example

```
FUNCTION Example()
' create the frame
AddFrame(1440, -1440, 2880, -2880)
IF Decide("Do you want to choose the frame layout options?")
    FrameLayout()  'frame layout dialog box
ELSE
    FrameModInit()     'set up for changes
    Border = (1440 / 5)
    FrameModBorders(1440, 1440, 1440, 1440, Border, Border, Border, Border, 1)
    FrameModLines(20, 3, 1, 0, 16777215, 0, 0, 0, 100, 100)
    FrameModType(Opaque, 0, "")
    FrameModFinish()   'apply changes
ENDIF
END FUNCTION
```

See also:

FrameModInit FrameModLines FrameModBorders AddFrame AddFrameDLG FrameLayout FrameModType IsFrameSelected BringFrameToFront SendFrameToBack

This function prepares Ami Pro to accept changes to the currently selected frame when modifying the frame layout. Choosing this function is equivalent to initializing changes made when choosing Frame/Modify Frame Layout.

You must have a frame selected before calling this function.

Syntax

FrameModInit()

Return Value

1 (TRUE) if the frame was changed.

-6 (NoMemory) if the function failed because of insufficient memory.

Example

```
FUNCTION Example()
' create the frame
AddFrame(1440, -1440, 2880, -2880)
IF Decide("Do you want to choose the frame layout options?")
    FrameLayout()  'frame layout dialog box
ELSE
    FrameModInit()     'set up for changes
    Border = (1440 / 5)
    FrameModBorders(1440, 1440, 1440, 1440, Border, Border, Border, Border, 1)
    FrameModLines(20, 3, 1, 0, 16777215, 0, 0, 0, 100, 100)
    FrameModType(Opaque, 0, "")
    FrameModFinish()   'apply changes
ENDIF
END FUNCTION
```

See also:

FrameModFinish FrameModLines FrameModBorders AddFrame AddFrameDLG FrameLayout FrameModType IsFrameSelected BringFrameToFront SendFrameToBack

This function applies modifications to the lines around, the background color, the shadow color, and the shadow placement of the selected frame. Choosing this function is equivalent to choosing Frame/Modify Frame Layout/Lines & shadows.

You must call the FrameModInit function before you use this function. Before the frame modifications take effect, the FrameModFinish function must be called.

Syntax

FrameModLines(BorderWhere, PosType, ThickType, ShadeType, BackType, ShadowColor, ShadowLeft, ShadowTop, ShadowRight, ShadowBottom)

BorderWhere is the lines around a frame. It is one of the following values:

1 - All
2 - Left
4 - Right
8 - Top
16 - Bottom

PosType is the position of the border around a frame. It is one of the following values:

1 - Middle
2 - Inside
3 - Outside
5 - Close to outside

You can only choose one value for the PosType parameter.

ThickType is the thickness of the border. It is one of the following values:

Hairline (1) - Hairline
OnePoint (2) - One point rule
TwoPoint (3) - Two point rule
ThreePoint (4) - Three point rule
FourPoint (5) - Four point rule
FivePoint (6) - Five point rule
SixPoint (7) - Six point rule
DoubleOnePoint (8) - Parallel one point rule
DoubleTwoPoint (9) - Parallel two point rule
ThreeLines (10) - Hairline above and below a two point rule
HairBelow (11) - Hairline below a three point rule
HairAbove (12) - Hairline above a three point rule

ShadeType is the line color.

BackType is the background color.

ShadowColor is the value assigned to the colors. It is one of the following values:

Red - 255
Orange - 33279
Yellow - 65535
Green - 65280
Cyan - 16776960
MedBlue - 16744448
Blue - 16727905
Purple - 16711809

Magenta - 16711935
Pink - 8388863
White - 16777215
Black - 0

The following four parameters determine the distance of the shadow from a specific side of the frame. They are either zero or positive integers. Multiply the desired distance in inches by 1440 to determine the value in twips. They are one of the following values, or may be a custom value:

None (0) - No shadow
Shallow (57) - Shallow shadow
Normal (100) - Normal shadow
Deep (172) - Deep shadow

ShadowLeft is the distance that the shadow is offset from the left side of the frame in twips (1 inch = 1440 twips).

ShadowTop is the distance that the shadow is offset from the top of the frame in twips (1 inch = 1440 twips).

ShadowRight is the distance that the shadow is offset from the right side of the frame in twips (1 inch = 1440 twips).

ShadowBottom is the distance that the shadow is offset from the bottom of the frame in twips (1 inch = 1440 twips).

Return Value

1 (TRUE) if the lines are modified.

-2 (GeneralFailure) if the lines are not modified.

Example

```
FUNCTION Example()
' create the frame
AddFrame(1440, -1440, 2880, -2880)
IF Decide("Do you want to choose the frame layout options?")
    FrameLayout()  'frame layout dialog box
ELSE
    FrameModInit()     'set up for changes
    Border = (1440 / 5)
    FrameModBorders(1440, 1440, 1440, 1440, Border, Border, Border, Border, 1)
    FrameModLines(20, 3, 1, 0, 16777215, 0, 0, 0, 100, 100)
    FrameModType(Opaque, 0, "")
    FrameModFinish()   'apply changes
ENDIF
END FUNCTION
```

See also:

FrameModInit FrameModFinish FrameModBorders AddFrame AddFrameDLG FrameLayout FrameModType IsFrameSelected

This function allows you to set the options for wrapping, placement, and roundness of the frame You can also set any macro that is associated with the selected frame. Choosing this function is equivalent to choosing Frame/Modify Frame Layout/Type.

You must call the FrameModInit function before this function. Before the frame modifications take effect, the FrameModFinish function must be called.

Syntax

FrameModType(Type, Rounded, MacroName)

Type is a setting based on how the text should wrap around a frame, whether a frame is transparent or opaque, has square or round corners, where it is placed on a page, and whether a macro is assigned to a frame. It is one of the following values:

TextFrame (512) - Is always used in combination with other values. It is a required value.

Opaque (64) - Hides text or picture behind frame.

Wraparound (128) - Displays text above, below, to the left, or to the right of the frame.

RepeatFrame (256) - Repeats frame on multiple pages. To repeat on all pages, do not use in combination with RepeatEven or RepeatOdd.

RepeatOdd (8192) - Repeats frame on odd pages. Use with the RepeatFrame value.

NoWrapBeside (131072) - Displays text above and below frame, but not to the left or right of the frame.

Borders (65536) - Uses if frame has borders.

AnchorFrame (524288) - Used to anchor frame in its current position or to a carriage return. You can not use any repeat values with this value.

RepeatEven (4194304) - Repeats frame on even pages. Use with the RepeatFrame value.

RunMacro (134217728) - Executes a macro each time the frame is selected.

You can add the values together to get the Type parameter.

Rounded is the amount that the corners are rounded, in percent (100% = circle).

MacroName is the name of the macro to run when the frame is selected.

Return Value

1 (TRUE) if the frame was modified.

-2 (GeneralFailure) if the frame was not modified.

Example

```
FUNCTION Example()
' create the frame
AddFrame(1440, -1440, 2880, -2880)
IF Decide("Do you want to choose the frame layout options?")
    FrameLayout()  'frame layout dialog box
ELSE
    FrameModInit()     'set up for changes
    Border = (1440 / 5)
    FrameModBorders(1440, 1440, 1440, 1440, Border, Border, Border, Border, 1)
    FrameModLines(20, 3, 1, 0, 16777215, 0, 0, 0, 100, 100)
    FrameModType(Opaque, 0, "")
    FrameModFinish()   'apply changes
ENDIF
END FUNCTION
```

See also:

FrameModInit FrameModFinish FrameModLines FrameModBorders AddFrame

AddFrameDLG FrameLayout IsFrameSelected

This function reads a specified number of bytes from the open file. This function is not line oriented.

When using ASCII file functions, ensure that the file was opened correctly before performing additional ASCII file functions. If the fopen function was unable to open the desired file, unpredictable results can occur if the macro attempts to read or write to that file. If you are reading binary information, zeroes can confuse this function. You must read binary files one byte at a time. If the empty string ("") is returned, it indicates a binary zero.

Syntax

fread(Handle, Length)

Handle is the file ID returned by the fopen function.

Len is the number of bytes to read from the file.

Return Value

The data requested from the file.

-1 if the file pointer is at the end of the file.

Example

```
FUNCTION Example()
DEFSTR string;
id = fopen("test.txt", "w")
IF id != 0
    Name = Query$("What is your name?")
    fwrite(id, Name)
    fwrite(id, BracketsToBin([Enter]))
    fputs(id, Query$("What is your address?"))
    fclose(id)
ENDIF
Exec("NOTEPAD.EXE", "TEST.TXT")
UserControl("Click Resume to continue...")
AppClose("Notepad - TEST.TXT")
id2 = fopen("test.txt", "r")
IF id2 != 0
    NameLength = len(Name)
    fseek(id2, 0, 0)
    Name = fread(id2, NameLength)
    EnterKey = BinToBrackets(fread(id2, 2))
    AddressBegins = ftell(id2)
    Address = fgets$(id2)
    Message("Your name is {Name}.")
    Message("Your address is {Address}.")
    Message("EnterKey = {EnterKey}")
    Message("The address begins at the {AddressBegins} byte.")
    fclose(id2)
ENDIF
END FUNCTION
```

See also:

fopen fclose fwrite fseek ftell BinToBrackets BracketsToBin

This function frees allocated memory assigned to a global variable. Global variables retain their values until they are freed or you exit Ami Pro, as opposed to regular variables, which are lost once the function they were created in is finished.

Syntax

FreeGlobalVar(Name)

Name is the string or number of the existing global variable being freed.

Return Value

1 (TRUE) if the global variable was successfully freed.

0 (FALSE) if no global variable with the requested ID number or name exists.

Example

```
FUNCTION Example()
AllocGlobalVar("Name", 1)
AllocGlobalVar("Computer", 1)
AllocGlobalVar("Software", 1)
Name = Query$("What is your name?")
Computer= Query$("What kind of computer do you have?")
Software= Query$("What is your favorite piece of software?")
SetGlobalVar("Name", Name)
SetGlobalVar("Computer", Computer)
SetGlobalVar("Software", Software)
CALL Example2()'Call the following function
END FUNCTION

FUNCTION Example2()
NumGlobs=GetGlobalVarCount()
DIM TempArray(Numglobs)
GetGlobalVarNames(&TempArray)
FOR i = 1 to NumGlobs
    VarName=TempArray(i)
    VarContents=GetGlobalVar$(VarName)
    TYPE ("Item {i}, {VarName}, is {VarContents}.[ENTER]")
    FreeGlobalVar(VarName)
NEXT
END FUNCTION
```

See also:

AllocGlobalVar GetGlobalArray$ GetGlobalVar$ SetGlobalArray SetGlobalVar Variables

This function moves the file pointer to another location in the file. The file pointer is the location in the file from which the next text is read.

The fseek function only works reliably when seeking to a location that is zero bytes from one of the starting locations, or to a location that is a value returned by the ftell function, starting from the beginning of the file.

When using ASCII file functions, ensure that the file was opened correctly before performing additional ASCII file functions. If the fopen function was unable to open the desired file, unpredictable results can occur if the macro attempts to read or write to that file.

Syntax

fseek(Handle, Location, StartPoint)

Handle is the file handle for the file returned by the fopen function.

Location is the number of characters to move the file pointer. It must be a positive integer.

StartPoint is the offset in the file to begin moving the file pointer from. Possible offsets are:

FBegin (0) - The start of the file
FCurrent (1) - The current file pointer location
FEnd (2) - The end of the file

Return Value

This function does not return a value.

Example

```
FUNCTION Example()
file = Query$("find size of which file?")
id = fopen(file, "r")  'fopen file read-only
fseek(id, 0, 2)    'seek to the end of the file
pos = ftell(id)    'report position of pointer
Message("Size of {file} is {pos} bytes.")
END FUNCTION
```

See also:

fclose fgets$ fopen fputs ftell fread fwrite

This function finds the location of the file pointer in the currently open file. The file pointer is the location in the file where the next text is read from or where the previous text was written to.

When using ASCII file functions, be careful to ensure that the file was opened correctly before performing additional ASCII file functions. If the fopen function was unable to open the desired file, unpredictable results can occur if the macro attempts to read or write to that file.

Syntax

ftell(Handle)

Handle is the file handle for the file returned by the fopen function.

Return Value

The current location in the file.

Example

```
FUNCTION Example()
file = Query$("find size of which file?")
id = fopen(file, "r")  'fopen file read-only
fseek(id, 0, 2)    'seek to the end of the file
pos = ftell(id)    'report position of pointer
Message("Size of {file} is {pos} bytes.")
END FUNCTION
```

See also:

fclose fgets$ fopen fputs fseek fread fwrite

This function changes the current view level to Full Page View. Choosing this function is equivalent to choosing View/Full Page. You must be in Layout Mode to call this function.

Syntax

FullPageView()

Return Value

This function returns 1.

Example

```
FUNCTION Example()
FullPageView()
END FUNCTION
```

See also:

EnlargedView FacingView GetViewLevel LayoutMode StandardView CustomView

This function writes data to the open file. Unlike the fputs function, this one does not append a carriage return/line feed to the end of the file and is not line oriented.

When using ASCII file functions, be careful to ensure that the file was opened correctly before performing additional ASCII file functions. If the fopen function was unable to open the desired file, unpredictable results can occur if the macro attempts to read or write to that file.

Syntax

fwrite(Handle, Data[, Length])

Handle is the file ID handle returned by the fopen function.

Data is the string data to place in the file.

Len is the optional length of the string to send. If Len is less than the actual length of the string data, the data is cut off at the Len position. If Len is greater than the length of the string data, the remaining positions are padded with spaces.

Return Value

This function does not return a value.

Example

```
FUNCTION Example()
amidir = GetAmiDirectory$()
' opens for appending; creates this file if it doesn't exist
id = fopen("{amidir}test.txt", "a")
IF id = 0
    exit function
ENDIF
Name=Query$("What is your name?")
fputs(id, Name)
Address= Query$("What is your address?")
fwrite(id, Address)
fwrite(id, "This is right after the address, with no carriage return.")
fclose(id)
Exec("NOTEPAD.EXE", "{amidir}TEST.TXT")
END FUNCTION
```

See also:

fopen fread fclose fwrite fseek ftell BinToBrackets BracketsToBin

This function generates a table of contents and/or an index for the current document. Choosing this function is equivalent to choosing Tools/TOC, Index. If this is the first time a table of contents has been generated, the user needs to select the paragraph styles and format for the table of contents before it is generated.

To set the output file for the table of contents, use the SetTOCFile function prior to calling Generate. To set the output file for the index, use the SetIndexFile function prior to calling Generate.

Syntax

Generate(Which)

Which is which table to generate and may be one of the following:

GenTOC (101) - Generate the table of contents

GenIndex (102) - Generate index

GenBoth (100) - Generate both tables

This parameter can be zero. If you are in a master document, no index or TOC is generated. The files open and the page numbers update.

To display the dialog box and allow the user to decide which table(s) to generate: **Generate**

Return Value

1 (TRUE) if the table was generated.

0 (UserCancel) if the user canceled the function.

-2 (GeneralFailure) if the table could not be generated.

Example

```
FUNCTION Example()
OnKey([CtrlAltg], Generate, 0)
Message("Press Ctrl+Alt+G to generate a TOC or index.")
END FUNCTION
```

See also:

MarkIndexWord

This function returns the path from which Ami Pro is currently running.

Syntax

GetAmiDirectory$()

Return Value

A string containing the path where Ami Pro is currently running, with a trailing backslash.

Example

```
FUNCTION Example()
AmiDir = GetAmiDirectory$()
WinDir = GetWindowsDirectory$()
CurDir = GetCurrentDir$()
DocPath = GetDocPath$()
StylePath = GetStylePath$()
MacPath = GetMacPath$()
Message("Ami Pro is in {AmiDir}.")
Message("Windows is in {WinDir}.")
Message("DOS reports the current directory is {CurDir}.")
Message("Ami Pro's default doc path is {DocPath}.")
Message("Ami Pro's default macro path is {MacPath}.")
Message("Ami Pro's default style path is {StylePath}.")
END FUNCTION
```

See also:

GetDocPath$ GetWindowsDirectory$ GetBackPath$ GetMacPath$ GetStylePath$

This function returns the default path for backup files.

Syntax

GetBackPath$()

Return Value

A string with the current default document backup path, with a trailing backslash.

Example

```
FUNCTION Example()
AmiDir = GetAmiDirectory$()
WinDir = GetWindowsDirectory$()
CurDir = GetCurrentDir$()
DocPath = GetDocPath$()
StylePath = GetStylePath$()
BackPath = GetBackPath$()
MacPath = GetMacPath$()
Message("Ami Pro is in {AmiDir}.")
Message("Windows is in {WinDir}.")
Message("DOS reports the current directory is {CurDir}.")
Message("Ami Pro's default doc path is {DocPath}.")
Message("Ami Pro's default macro path is {MacPath}.")
Message("Ami Pro's default style path is {StylePath}.")
Message("Ami Pro's default backup path is {BackPath}.")
END FUNCTION
```

See also:

GetCurrentDir$ GetDocPath$ GetMacPath$ GetStylePath$ SetBackPath SetDocPath SetStylePath GetAmiDirectory$ GetWindowsDirectory$

This function is used to dimension the arrays for the GetBookMarkNames function.

Syntax

GetBookMarkCount()

Return Value

The number of bookmarks in the current document.

Example

```
FUNCTION Example()
MacFile = GetRunningMacroFile$()
Count = GetBookMarkCount()
IF Count > 0
    DIM BookMarks(Count)
    GetBookMarkNames(&BookMarks)
    DeleteMenu(1, "&BookMarks")
    AddMenu(1, "&BookMarks")
    FOR I = 1 to Count
        ThisBookMark = BookMarks(I)
        AddMenuItem(1, "&BookMarks", ThisBookMark, "{MacFile}!Example2({ThisBookMark})",
ThisBookMark)
    NEXT
ELSE
    Message("No bookmarks in this document!")
ENDIF
END FUNCTION

FUNCTION Example2(Bkmk)
BookPage = GetBookMarkPage(Bkmk)
Message("Now going to {Bkmk} on page {BookPage}.")
MarkBookMark(Bkmk, FindBookMark)
END FUNCTION
```

See also:

GetBookMarkNames MarkBookMark GoToCmd GoToShade

This function shows the names of the arrays that are to receive the bookmark names.

Syntax

GetBookMarkNames(&Array)

&Array is the names of the arrays. Note the use of indirection (&).

Return Value

The number of bookmark names.

Example

```
FUNCTION Example()
MacFile = GetRunningMacroFile$()
Count = GetBookMarkCount()
IF Count > 0
    DIM BookMarks(Count)
    GetBookMarkNames(&BookMarks)
    DeleteMenu(1, "&BookMarks")
    AddMenu(1, "&BookMarks")
    FOR I = 1 to Count
        ThisBookMark = BookMarks(I)
        AddMenuItem(1, "&BookMarks", ThisBookMark, "{MacFile}!Example2({ThisBookMark})",
ThisBookMark)
    NEXT
ELSE
    Message("No bookmarks in this document!")
ENDIF
END FUNCTION

FUNCTION Example2(Bkmk)
BookPage = GetBookMarkPage(Bkmk)
Message("Now going to {Bkmk} on page {BookPage}.")
MarkBookMark(Bkmk, FindBookMark)
END FUNCTION
```

See also:

GetBookMarkCount MarkBookMark GoToCmd GoToShade

This function finds the page number where the specified bookmark begins. This number is relative only to the current document. If the file is one file of a master document, you must use the PhysicalToLogical function to determine the printed page number.

Syntax

GetBookMarkPage(Name)

Name is the name of the bookmark (in the current document) whose page you want to find.

Return Value

The page number on which the specified bookmark begins.

-2 if the bookmark could not be found.

Example

```
FUNCTION Example()
MacFile = GetRunningMacroFile$()
Count = GetBookMarkCount()
IF Count > 0
    DIM BookMarks(Count)
    GetBookMarkNames(&BookMarks)
    DeleteMenu(1, "&BookMarks")
    AddMenu(1, "&BookMarks")
    FOR I = 1 to Count
        ThisBookMark = BookMarks(I)
        AddMenuItem(1, "&BookMarks", ThisBookMark, "{MacFile}!Example2({ThisBookMark})",
ThisBookMark)
    NEXT
ELSE
    Message("No bookmarks in this document!")
ENDIF
END FUNCTION

FUNCTION Example2(Bkmk)
BookPage = GetBookMarkPage(Bkmk)
Message("Now going to {Bkmk} on page {BookPage}.")
MarkBookMark(Bkmk, FindBookMark)
END FUNCTION
```

See also:

GetPageNo MarkBookMark GoToCmd PhysicalToLogical GetBookMarkCount GetBookMarkNames

This function returns information about the current font at the insertion point.

Syntax

GetCurFontInfo(&Name, &Color, &Size, &PitchFamily)

Name is the name of the font.

Color is the color of the font.

Size is the size of the font in twips (1 inch = 1440 twips).

Family is the bit value containing Pitch and Family. To extract these, use the bitwise operator OR (|). Note the use of indirection (&).

Return Value

This function does not return a value.

Example

```
FUNCTION Example()
DEFSTR Name, Color, Size, Family;
' get the current font information
GetCurFontInfo(&Name, &Color, &Size, &Family)
' increase pointsize by two and calculate the twips
Size = ((Size / 20) + 2) * 20
' change font
FontChange(Name, Family, Color, Size)
END FUNCTION
```

See also:

FontChange FontFaceChange FontPointSizeChange FontRevert

This function finds the frame border information for the selected frame.

Syntax

GetCurFrameBorders(&Width, &Height, &Top, &Left, &LeftMargin, &TopMargin, &RightMargin, &BottomMargin)

Width is the width of the selected frame.

Height is the height of the selected frame.

Top is the vertical starting position of the frame.

Left is the horizontal starting position of the frame.

LeftMargin is the size of the left border of the frame.

TopMargin is the size of the top border of the frame.

RightMargin is the size of the right border of the frame.

BottomMargin is the size of the bottom border of the frame.

All values are returned in twips (1 inch = 1440 twips).

Note the use of indirection (&).

Return Value

This function does not return a value.

Example

```
FUNCTION Example()
DEFSTR Width, Height, Top, Left, LM, TM, RM, BM, Units;
' you must have a frame selected
WHILE not IsFrameSelected()
    UserControl("Please select a frame and choose Resume.")
WEND
' get the frame info
GetCurFrameBorders(&Width, &Height, &Top, &Left, &LM, &TM, &RM, &BM, &Units)
Message("The frame's dimensions are {Width} x {Height}.")
Message("The frame starts {Top} twips from the top of the page.")
Message("The frame starts {Left} twips from the left edge of the page.")
Message("The margins (in twips) of the frame are (T,B,L,R): {TM}, {BM}, {LM}, {RM}")
END FUNCTION
```

See also:

FrameModBorders FrameLayout FrameModColumns FrameModInit FrameModType FrameModFinish FrameModLines AddFrame AddFrameDlg

This function finds information pertaining to the lines and shadows of the selected frame.

Syntax

GetCurFrameLines(&BorderWhere, &PosType, &ThickType, &ShadeType, &BackType, &ShadowColor, &ShadowLeft, &ShadowTop, &ShadowRight, &ShadowBottom, Units)

BorderWhere is the lines around a frame. It is one of the following values:

1 - All
2 - Left
4 - Right
8 - Top
16 - Bottom

PosType is the position of the border around a frame. It is one of the following values:

1 - Middle
2 - Inside
3 - Outside
5 - Close to outside

You can only choose one value for the PosType parameter.

ThickType is the thickness of the border. It is one of the following values:

Hairline (1) - Hairline
OnePoint (2) - One point rule
TwoPoint (3) - Two point rule
ThreePoint (4) - Three point rule
FourPoint (5) - Four point rule
FivePoint (6) - Five point rule
SixPoint (7) - Six point rule
DoubleOnePoint (8) - Parallel one point rule
DoubleTwoPoint (9) - Parallel two point rule
ThreeLines (10) - Hairline above and below a two point rule
HairBelow (11) - Hairline below a three point rule
HairAbove (12) - Hairline above a three point rule

ShadeType is the line color.

BackType is the background color.

ShadowColor is the value assigned to the colors. It is one of the following values:

Red - 255
Orange - 33279
Yellow - 65535
Green - 65280
Cyan - 16776960
MedBlue - 16744448
Blue - 16727905
Purple - 16711809
Magenta - 16711935
Pink - 8388863
White - 16777215
Black - 0

The following four parameters determine the distance of the shadow from a specific side of the frame. They are either zero or positive integers. Multiply the desired distance in inches by 1440 to determine the value in twips. They are one of the following values or they may be a custom value:

None (0) - No shadow
Shallow (57) - Shallow shadow
Normal (100) - Normal shadow
Deep (172) - Deep shadow

ShadowLeft is the distance that the shadow is offset from the left side of the frame in twips (1 inch = 1440 twips).

ShadowTop is the distance that the shadow is offset from the top of the frame in twips (1 inch = 1440 twips).

ShadowRight is the distance that the shadow is offset from the right side of the frame in twips (1 inch = 1440 twips).

ShadowBottom is the distance that the shadow is offset from the bottom of the frame in twips (1 inch = 1440 twips).

Units is the type of measurement and can be one of the following:

Inches (1) - units set to inches
CM (2) - units set to centimeters
Picas (3) - units set to picas
Points (4) - units set to points

The amount of indention must be given in twips (1 inch = 1440 twips). Multiply the desired number of inches by 1440 to determine the value in twips.

Note The Use Of Indirection (&).

Return Value

This function does not return a value.

Example

```
FUNCTION Example()
DEFSTR Border, Pos, Thick, Shade, Background, ShadowColor, SL, ST, SR, SB Units;
' you must have a frame selected
WHILE not IsFrameSelected()
    UserControl("Please select or create frame.")
WEND
' get the frame info
GetCurFrameLines(&Border, &Thick, &Shade, &Background, &ShadowColor, &SL, &ST, &SR, &SB, &Units)
TYPE("[ESC]Frame Border: {Border}[Enter]Frame Pos: {Pos}[Enter]")
TYPE("Line Thickness: {Thick}[Enter]Line Shading: {Shade}[Enter]")
TYPE("Shadow Background: {Background}[Enter]Bkgrnd Color: {ShadowColor}[Enter]")
TYPE("Shadow Left: {SL}[Enter]Shadow Right: {SR}[Enter]")
TYPE("Shadow Top: {ST}[Enter]Shadow Bottom: {SB}[Enter]")
END FUNCTION
```

See also:

FrameModBorders FrameModFinish FrameModInit FrameModType FrameModColumns FrameModLines FrameLayout AddFrame AddFrameDlg

This function finds information about the type of frame, the roundness of the frame, and the macro assigned to the frame.

Syntax

GetCurFrameType(&Type, &Rounded, &MacroName)

Type is the frame type.

Rounded is the amount that the corners are rounded, in percent. (100% = circle).

MacroName is the name, if any, of the macro attached to this frame.

Return Value

This function does not return a value.

Example

```
FUNCTION Example()
DEFSTR Type, Rounded, MacroName;
' you must have a frame selected
WHILE not IsFrameSelected()
    UserControl("Please select or create a frame.")
WEND
' get the frame info
GetCurFrameType(&Type, &Rounded, &MacroName)
Message("The type number of this frame is {Type}.")
Message("The percentage of corner rounding on this frame is {Rounded}.")
IF MacroName != ""
    Message("The macro attached to this frame is {MacroName}.")
ELSE
    Message("There is no macro attached to this frame.")
ENDIF
END FUNCTION
```

See also:

FrameModType FrameLayout FrameModBorders FrameModInit FrameModFinish FrameModLines FrameModColumns AddFrameDlg AddFrame

This function returns the current directory as seen by Ami Pro, Windows, and DOS.

Syntax

GetCurrentDir$()

Return Value

A string with the current working directory, including a trailing backslash.

Example

```
FUNCTION Example()
AmiDir = GetAmiDirectory$()
WinDir = GetWindowsDirectory$()
CurDir = GetCurrentDir$()
DocPath = GetDocPath$()
StylePath = GetStylePath$()
MacPath = GetMacPath$()
Message("Ami Pro is in {AmiDir}.")
Message("Windows is in {WinDir}.")
Message("DOS reports the current directory is {CurDir}.")
Message("Ami Pro's default doc path is {DocPath}.")
Message("Ami Pro's default macro path is {MacPath}.")
Message("Ami Pro's default style path is {StylePath}.")
END FUNCTION
```

See also:

GetBackPath$ GetDocPath$ GetMacPath$ GetStylePath$ GetAmiDirectory$ GetWindowsDirectory$

This function is used to retrieve the contents of a dialog box field that has been displayed in the last DialogBox function call.

Before using this function, ensure that the user did not cancel out of the DialogBox function, or the results of this function are invalid.

Syntax

GetDialogField$(ID)

ID is the ID Number of the field whose value is being retrieved, as defined in the resource file.

Return Value

Up to an 80 byte string if the field is an edit box, a list box, or a combo box.

0 (FALSE) if the object was a button and was not checked or selected.

1 (TRUE) if the object was a button and was checked or selected.

Example

```
FUNCTION Example()
Filledit(20,1)     ' Turn apples on
Filledit(8000,"Bob")   ' Put text in Edit Box
Filledit(9001,"*.sam")
Box = DialogBox(".","ExampleBox")
IF Box<>1
    EXIT FUNCTION
ENDIF
Store = GetdialogField$(22)       ' Retrieve Combo Box
File = GetdialogField$(9001)      ' Retrieve List Box result
Name = GetDialogField$(8000)  ' Retrieve Edit Box
Dir = GetCurrentDir$()     ' Get current directory
Message("Goto Store = {Store} Name ={Name} File = {File}")
END FUNCTION

DIALOG ExampleBox
-2134376448 10 71 44 115 98 "" "" "Sample Dialog Box"
FONT 8 "Helv"
69 3 40 14 1 1342373889 "button" "OK" 0
69 19 40 14 2 1342373888 "button" "Cancel" 0
6 24 52 56 9001 1352728579 "listbox" "" 0
6 13 40 10 7999 1342177280 "static" "" 0
6 4 40 10 1001 1342177280 "static" "&Files:" 0
69 37 37 12 20 1342242825 "button" "&Apples" 0
69 49 42 12 21 1342242825 "button" "&Oranges" 0
6 83 45 12 22 1342242819 "button" "&Goto Store" 0
69 61 42 12 23 1342242825 "button" "&Soap" 0
69 82 43 12 8000 1350631552 "edit" "" 0
END DIALOG
```

See also:

DialogBox FillEdit FillList SetDLGCallback GetDLGItem GetDLGItemText SetDLGItemText

This function returns the Microsoft Windows window handle to the specified control or object.

Syntax

GetDlgItem(Handle, ID)

Handle is the handle to the dialog box, passed to a callback function.

ID is the ID number of the object for which you want to get the handle.

Return Value

The Microsoft Windows window handle for the specified control or object.

Example

```
FUNCTION Example()
' get the name of this macro file
MacFile = GetRunningMacroFile$()
' run ClearBox if the Re-Fill box is pressed
SetDlgCallBack(50, "{MacFile}!ClearBox")
FOR I = 1 to 10
    FillEdit(9000, I)  ' fill the edit box
NEXT
Box = DialogBox(".", "ExampleBox")    ' display the example box
END FUNCTION

FUNCTION ClearBox(hdlg, id, text)
' get the handle to the list box
handle = GetDlgItem(hdlg, 9000)
AppSendMessage(handle, 0x0405, 0, 0)
Message("Cleared List Box, Now I Will Re-Fill It...")
FOR I = 1 to 10
    SetDlgItemText(hdlg, 9000, I)
NEXT
END FUNCTION

DIALOG ExampleBox
-2134376448 4 169 46 106 76 "" "" "Example Dialog Box"
FONT 6 "Helv"
62 3 40 14 2 1342242817 "button" "DONE" 0
62 19 40 14 50 1342242816 "button" "Re-Fill Box" 0
4 11 52 61 9000 1352728579 "listbox" "" 0
5 2 51 9 1000 1342177280 "static" "Number:" 0
END DIALOG
```

See also:

DialogBox FillEdit FillList SetDLGCallback GetDLGItemText SetDLGItemText

This function returns the contents of a dialog box object field.

Syntax

GetDlgItemText(Handle, ID)

Handle is the handle to the dialog box, passed to a CallBack function.

ID is the ID number of the desired object. If the ID is a radio button or check box, the text returns a TRUE or FALSE condition.

Return Value

The contents of the specified object.

If the object is a button, this function returns a TRUE or FALSE value.

If the object is an edit box, a list box, or a combo box, this function returns a string.

Example

```
FUNCTION Example()
' get the name of this macro file
MacFile = GetRunningMacroFile$()
' run Message1 function when Run Example is pressed
SetDlgCallBack(50, "{MacFile}!Message1")
' ExampleBox is in this file
Box = DialogBox(".", "ExampleBox")
IF Box = -1
    Message("Could not find dialog box!")
    EXIT FUNCTION
ELSEIF Box = 0
    EXIT FUNCTION
ENDIF
' type the name backwards into a document
TYPE(GetDialogField$(8002))
END FUNCTION

FUNCTION Message1(hdlg, id, text)
' get the type in name
Name = GetDlgItemText(hdlg, 8000)
Message("The contents of the first box are {Name}.")
Message("We will now fill the second box with the inverse of {Name}.")
' reverse the name
FOR I = len(Name) to 1 step -1
    Name2 = strcat$(Name2, MID$(Name, I, 1))
NEXT
' fill the second edit box
SetDlgItemText(hdlg, 8002, Name2)
END FUNCTION


DIALOG ExampleBox
-2134376448 8 106 38 160 54 "" "" "Sample Dialog Box"
FONT 6 "Helv"
50 6 62 12 8000 1350631552 "edit" "" 0
4 6 44 10 1000 1342177280 "static" "Your Name:" 0
50 20 62 12 1003 1342177287 "static" "" 0
4 22 44 10 1002 1342177280 "static" "Reversed:" 0
52 22 58 8 8002 1342177280 "static" "" 0
116 4 40 14 1 1342242817 "button" "OK" 0
116 20 40 14 2 1342242816 "button" "Cancel" 0
```

```
100 36 56 14 50 1342242816 "button" "&Run Example..." 0
END DIALOG
```

See also:

DialogBox FillEdit FillList SetDlgCallback GetDlgItem GetDialogField$

This function retrieves one of the document information fields from the current document.

Syntax

GetDocInfo$(Which)

Which is the desired document info field to retrieve and may be one of the following:

DDFilename (1) - File name
DDPath (2) - Path for this document
DDStylesheet (3) - Style sheet for this document
DDCreated (4) - Date document was created
DDRevised (5) - Date document was revised
DDRevisions (6) - Number of document revisions
DDDescription (7) - Document info
DDUser1 (8) - User Defined Field 1
DDUser2 (9) - User defined Field 2
DDUser3 (10) - User defined Field 3
DDUser4 (11) - User defined Field 4
DDUser5 (12) - User defined Field 5
DDUser6 (13) - User defined Field 6
DDUser7 (14) - User defined Field 7
DDUser8 (15) - User Defined Field 8

Return Value

A string containing the desired document info field.

Example

```
FUNCTION Example()
DIM Field(15)
Message("Ready to type the contents of the Doc Info fields into the document.")
FOR I = 1 to 15
    Field(I) = GetDocInfo$(I)
NEXT
' put the field values in variables
Filename = Field(1)
Path = Field(2)
Style = Field(3)
Created = Field(4)
Revised = Field(5)
Number = Field(6)
Descrip = Field(7)
TYPE("The current filename is {Filename}.[Enter]")
TYPE("The path for this document is {Path}.[Enter]")
TYPE("The style sheet for this document is {Style}.[Enter]")
TYPE("This document was created on {Created}.[Enter]")
TYPE("This document was last revised on {revised}.[Enter]")
TYPE("This document has been revised {Number} times.[Enter]")
TYPE("The document description for this document states:[Enter]{Descrip}.[Enter]")
FOR I = 1 to 8
    ThisField = Field(I + 7)
    IF ThisField != ""
        TYPE("User defined field {I} of this document contains: {ThisField}.[Enter]")
    ELSE
        TYPE("User defined field {I} contains nothing.[Enter]")
```

```
    ENDIF
NEXT
END FUNCTION
```

See also:

DocInfo RenameDocInfoField InsertDocInfo InsertDocInfoField GetDocInfoKeywords$

This function returns the keywords field from the Doc Info dialog box.

Syntax

GetDocInfoKeywords$()

Return Value

This function returns the Keywords field.

Example

```
FUNCTION Example()
Words=GetDocInfoKeywords$()
Message ("The Keywords for this doc are:{Words}.")
END FUNCTION
```

See also:

GetDocInfo$ DocInfo RenameDocInfoField InsertDocInfo InsertDocInfoField

This function returns the drive and directory of the default document storage path.

Syntax

GetDocPath$()

Return Value

A string containing the current default document path, with a trailing backslash.

Example

```
FUNCTION Example()
AmiDir = GetAmiDirectory$()
WinDir = GetWindowsDirectory$()
CurDir = GetCurrentDir$()
DocPath = GetDocPath$()
StylePath = GetStylePath$()
MacPath = GetMacPath$()
Message("Ami Pro is in {AmiDir}.")
Message("Windows is in {WinDir}.")
Message("DOS reports the current directory is {CurDir}.")
Message("Ami Pro's default doc path is {DocPath}.")
Message("Ami Pro's default macro path is {MacPath}.")
Message("Ami Pro's default style path is {StylePath}.")
END FUNCTION
```

See also:

GetBackPath$ GetCurrentDir$ GetMacPath$ GetStylePath$ SetBackPath SetDocPath SetStylePath GetAmiDirectory$ GetWindowsDirectory$

This function returns the value associated with Name. Document variables are kept with the current document. Document variables are similar to WIN.INI entries in that they have a name and a value. They can be retrieved by name using this function or all document variables can be returned using the GetPowerFields function.

Syntax

GetDocVar(Name)

Name is the name of the document variable to retrieve.

Return Value

A string containing the information in the document variable.

Example

```
FUNCTION Example()
val = GetDocVar("answer1")
END FUNCTION
```

See also:

SetDocVar

This function returns the page number of a bookmark and the paragraph style that page number is currently using.

Syntax

GetFmtPageStr$(Name)

Name is the name of the Bookmark for which to find the page number.

Return Value

A string containing the page on which the specified bookmark begins, including the paragraph style the page number is using.

Example

```
FUNCTION Example()
Bkmk = Query$("What is the name of the bookmark to use?")
Page = GetFmtPageStr$(Bkmk)
Message("The page, with formatting that {Bkmk} is on is {Page}.")
END FUNCTION
```

See also:

GetBookMarkPage GetBookMarkCount GetBookMarkNames MarkBookMark GetPageNo GoToCmd

This function returns a global array element.

Global variables cannot be directly used by functions and statements. Their values must be assigned to local variables before they can be used.

Syntax

GetGlobalArray$(Name, Index)

Name is the name or the number of the global variable to use.

Index is the element of the global array you are retrieving.

Return Value

The value of the global array element.

Example

```
FUNCTION Example()
AllocGlobalVar("Names", 5)'Allocate space for a 5 element global variable
AllocGlobalVar("Numbers", 5)'Allocate space for a 5 element global variable
AllocGlobalVar("YourName", 1)'Allocate space for a single element global variable.
FOR I = 1 to 5'Do the following 5 times.
SetGlobalArray("Names", I, Query$("Enter Name Number {I}"))
'Fill a global array with the return from QUERY
SetGlobalArray("Numbers", I, (100/I))'Fill a global array with a number
NEXT
NewName = Query$("What is your name?")'Query the user for his/her name
SetGlobalVar("YourName", NewName)'Set a global variable to that value
CALL Example2()'Call the following function
END FUNCTION

FUNCTION Example2()
Name = GetGlobalVar$("YourName")'Get the value of the global variable
Message("Your name is {Name}.")'Message that value in a box.
FOR I = 1 to 5'Do the following 5 times.
TheirName = GetGlobalArray$("Names", I)'Get the value of the current element from the array
TheirNumber = GetGlobalArray$("Numbers", I)'Get the value of the current element from the array
TYPE("Name #{I} is {TheirName}, and the number is {TheirNumber}.[Enter]")
'Type the values to the screen.
NEXT
FreeGlobalVar("Names")'Clear the space for the first global array
FreeGlobalVar("Numbers")'Clear the space for the second global array
FreeGlobalVar("YourName")'Clear the space for the global variable
END FUNCTION
```

See also:

AllocGlobalVar FreeGlobalVar GetGlobalVar$ SetGlobalArray SetGlobalVar Variables

This function returns a global array variable.

Global variables cannot be directly used by functions and statements. Their values must be assigned to local variables before they can be used.

Syntax

GetGlobalVar$(Name)

Name is the name or the number of the global variable to retrieve the data from.

Return Value

The value of the global variable.

Example

```
FUNCTION Example()
AllocGlobalVar("Name", 1)
AllocGlobalVar("Computer", 1)
AllocGlobalVar("Software", 1)
Name = Query$("What is your name?")
Computer = Query$("What kind of computer do you have?")
Software = Query$("What is your favorite piece of software?")
SetGlobalVar("Name", Name)
SetGlobalVar("Computer", Computer)
SetGlobalVar("Software", Software)
CALL Example2()'Call the following function
END FUNCTION

FUNCTION Example2()
NumGlobs = GetGlobalVarCount()
DIM TempArray(Numglobs)
GetGlobalVarNames(&TempArray)
FOR i = 1 to NumGlobs
    VarName=TempArray(i)
    VarContents=GetGlobalVar$(VarName)
    TYPE ("Item {i}, {VarName}, is {VarContents}.[ENTER]")
    FreeGlobalVar(VarName)
NEXT
END FUNCTION
```

See also:

AllocGlobalVar FreeGlobalVar GetGlobalArray$ SetGlobalArray SetGlobalVar Variables

This function returns the total number of global variables currently allocated. It is used to dimension an array before calling the GetGlobalVarNames function.

Syntax

GetGlobalVarCount()

Return Value

The total number of global variables currently allocated.

Example

```
FUNCTION Example()
AllocGlobalVar("Name", 1)
AllocGlobalVar("Computer", 1)
AllocGlobalVar("Software", 1)
Name = Query$("What is your name?")
Computer = Query$("What kind of computer do you have?")
Software = Query$("What is your favorite piece of software?")
SetGlobalVar("Name", Name)
SetGlobalVar("Computer", Computer)
SetGlobalVar("Software", Software)
CALL Example2()'Call the following function
END FUNCTION

FUNCTION Example2()
NumGlobs = GetGlobalVarCount()
DIM TempArray(Numglobs)
GetGlobalVarNames(&TempArray)
FOR i = 1 to NumGlobs
    VarName=TempArray(i)
    VarContents=GetGlobalVar$(VarName)
    TYPE ("Item {i}, {VarName}, is {VarContents}.[ENTER]")
    FreeGlobalVar(VarName)
NEXT
END FUNCTION
```

See also:

GetGlobalVarNames AllocGlobalVar SetGlobalVar SetGlobalArray GetGlobalVar$ GetGlobalArray$

This function finds the names or ID numbers of all of the currently allocated global variables.

Syntax

GetGlobalVarNames(&Array)

Array is the name of the array to place the names or ID numbers of the currently allocated global variables.

Note the use of indirection (&).

Return Value

The number of global variances.

Example

```
FUNCTION Example()
AllocGlobalVar("Name", 1)
AllocGlobalVar("Computer", 1)
AllocGlobalVar("Software", 1)
Name = Query$("What is your name?")
Computer = Query$("What kind of computer do you have?")
Software = Query$("What is your favorite piece of software?")
SetGlobalVar("Name", Name)
SetGlobalVar("Computer", Computer)
SetGlobalVar("Software", Software)
CALL Example2()'Call the following function
END FUNCTION

FUNCTION Example2()
NumGlobs = GetGlobalVarCount()
DIM TempArray(Numglobs)
GetGlobalVarNames(&TempArray)
FOR i = 1 to NumGlobs
    VarName=TempArray(i)
    VarContents=GetGlobalVar$(VarName)
    TYPE ("Item {i}, {VarName}, is {VarContents}.[ENTER]")
    FreeGlobalVar(VarName)
NEXT
END FUNCTION
```

See also:

GetGlobalVarCount AllocGlobalVar SetGlobalVar SetGlobalArray GetGlobalVar$ GetGlobalArray$

This function returns the name of the current icon set.

Syntax

GetIconPalette()

Return Value

A string containing the name of the icon set.

Example

```
FUNCTION Example()
palette = GetIconPalette()
Message("The icon palette is {palette}.")
END FUNCTION
```

See also:

ChangeIcons IconBottom IconCustomize IconFloating IconLeft IconRight IconTop SetIconSize

This function finds the current information about any lines associated with the current page.

Syntax

GetLayoutLeftLines(&GutterShade, &GutterStyle, &BorderSides, &BorderStyle, &BorderSpace)

GutterShade is the color of the line.

GutterStyle and **BorderStyle** define the width and type of the line and should be one of the following:

- Hairline (1) - Hairline
- OnePoint (2) - One point rule
- TwoPoint (3) - Two point rule
- ThreePoint (4) - Three point rule
- FourPoint (5) - Four point rule
- FivePoint (6) - Five point rule
- SixPoint (7) - Six point rule
- DoubleOnePoint (8) - Parallel one point rule
- DoubleTwoPoint (9) - Parallel two point rule
- ThreeLines (10) - Hairline above and below a two point rule
- HairBelow (11) - Hairline below a three point rule
- HairAbove (12) - Hairline above a three point rule

BorderSides is where the lines are placed on the page and can be one or more of the following:

- (1) - All sides
- (2) - Left
- (4) - Right
- (8) - Top
- (16) - Bottom

To extract any combination, use the bitwise operator OR (|) on this number.

The Right value is also used when you have only one page format.

BorderSpace is how close the border is to the printed page and can be one of the following:

- (1) - Middle
- (2) - Inside
- (3) - Outside
- (4) - Close to the inside
- (5) - Close to the outside

Note the use of indirection (&).

Return Value

This function does not return a value.

Example

```
FUNCTION Example()
DEFSTR Gshade, Gstyle, Bsides, Bstyle, Bspace;
' get the page layout line info
GetLayoutLeftLines(&Gshade, &Gstyle, &Bsides, &Bstyle, &Bspace)
TYPE("The color of the lines is: {Gshade}.[Enter]")
TYPE("The width of the gutter is: {Gstyle}.[Enter]")
TYPE("The lines are placed: {Bsides}.[Enter]")
TYPE("The style of the border is: {Bstyle}.[Enter]")
TYPE("The location of the lines is: {Bspace}.[Enter]")
END FUNCTION
```

See also:

GetLayoutPageSize GetLayoutParameters GetLayoutParmCnt GetLayoutRightLines GetLayoutType ModifyLayout

This function finds the page size information for the current page.

Syntax

GetLayoutPageSize(&Length, &Width, &Units, &PaperType)

Length is the length of the page, in twips (1 inch = 1440 twips).

Width is the width of the page, in twips.

Units is the units the current page is measured in, and can be one of the following:

Inches (1) - units set to inches
CM (2) - units set to centimeters
Picas (3) - units set to picas
Points (4) - units set to points

PaperType is the predetermined type of paper and can be one of the following:

(1) - Letter
(2) - Legal
(3) - A3
(4) - A4
(5) - A5
(6) - B5
(7) - Custom

Note the use of indirection (&).

Return Value

This function does not return a value.

Example

```
FUNCTION Example()
DEFSTR Length, Width, Units, Papertype;
' get the page size info
GetLayoutPageSize(&Length, &Width, &Units, &Papertype)
TYPE("The length of the page is {Length} twips.[Enter]")
TYPE("The width of the page is {Width} twips.[Enter]")
SWITCH Units
    CASE 1
    Units = "inches"
    CASE 2
    Units = "centimeters"
    CASE 3
    Units = "picas"
    CASE 4
    Units = "points"
ENDSWITCH
TYPE("The current page measurements are in {Units}.[Enter]")
TYPE("The current paper type is {Papertype} (where 1 = Letter, 2 = Legal, 3 = A3, 4 = A4, 5 = A5,
6 = B5, and 7 = Custom).[Enter]")
END FUNCTION
```

See also:

GetLayoutLeftLines GetLayoutParameters GetLayoutParmCnt GetLayoutRightLines GetLayoutType ModifyLayout

This function finds the margin, columns, and tab information for the current page.

Syntax

GetLayoutParameters(Which, &Parameters)

Which is the type of page layout you want to get information about. It can be one of the following functions:

- ModLayoutRightPage
- ModLayoutRightHeader
- ModLayoutRightFooter
- ModLayoutLeftPage
- ModLayoutLeftHeader
- ModLayoutLeftFooter

Parameters is an array that was dimensioned according to the return value of GetLayoutParmCnt. Note the use of indirection (&).

Return Value

This function does not return a value.

Example

```
FUNCTION Example()
' Get number of parameters for ModLayoutRightPage
Cnt = GetLayoutParmCnt(ModLayoutRightPage)
DIM Stuff(Cnt)     'Dimension an array for that amount
' Get the parameters and place in the array
GetLayoutParameters(ModLayoutRightPage, &Stuff)
' Query for new headers lengthh
length = Query$("New length of right/all headers?")
' Query for new headers height.
width = Query$("New width of right/all headers?")
Stuff(1) = length * 1440      'Turn into twips
Stuff(2) = width * 1440       'Turn into twips
' Prep Ami Pro to accept Layout changes for all pages
ModLayoutInit(512)
' Apply changes to specified function.
AmiProIndirect(ModLayoutRightPage, &Stuff, Cnt)
' Tell Ami Pro to accept the specified changes
ModLayoutFinish()
END FUNCTION
```

See also:

GetLayoutLeftLines GetLayoutPageSize GetLayoutParmCnt GetLayoutRightLines GetLayoutType ModifyLayout

This function returns the number of parameters to expect from the function GetLayoutParameters. It is used to dimension an array for that function.

Syntax

GetLayoutParmCnt(Which)

Which is the type of page layout you want to get information about. It can be one of the following functions:

- ModLayoutRightPage
- ModLayoutRightHeader
- ModLayoutRightFooter
- ModLayoutLeftPage
- ModLayoutLeftHeader
- ModLayoutLeftFooter

Return Value

The number of parameters for the specified function.

Example

```
FUNCTION Example()
' Get number of parameters for ModLayoutRightPage
Cnt = GetLayoutParmCnt(ModLayoutRightPage)
DIM Stuff(Cnt)     'Dimension an array for that amount
' Get the parameters and place in the array
GetLayoutParameters(ModLayoutRightPage, &Stuff)
' Query for new headers lengthh
length = Query$("New length of right/all headers?")
' Query for new headers height.
width = Query$("New width of right/all headers?")
Stuff(1) = length * 1440      'Turn into twips
Stuff(2) = width * 1440       'Turn into twips
' Prep Ami Pro to accept Layout changes for all pages
ModLayoutInit(512)
' Apply changes to specified function.
AmiProIndirect(ModLayoutRightPage, &Stuff, Cnt)
' Tell Ami Pro to accept the specified changes
ModLayoutFinish()
END FUNCTION
```

See also:

GetLayoutLeftLines GetLayoutPageSize GetLayoutParameters GetLayoutRightLines GetLayoutType ModifyLayout

This function finds the current information about any lines associated with the current page.

Syntax

GetLayoutRightLines(&GutterShade, &GutterStyle, &BorderSides, &BorderStyle, &BorderSpace)

GutterShade is the color of the line.

GutterStyle and **BorderStyle** define the width and type of the line and should be one of the following:

- Hairline (1) - Hairline
- OnePoint (2) - One point rule
- TwoPoint (3) - Two point rule
- ThreePoint (4) - Three point rule
- FourPoint (5) - Four point rule
- FivePoint (6) - Five point rule
- SixPoint (7) - Six point rule
- DoubleOnePoint (8) - Parallel one point rule
- DoubleTwoPoint (9) - Parallel two point rule
- ThreeLines (10) - Hairline above and below a two point rule
- HairBelow (11) - Hairline below a three point rule
- HairAbove (12) - Hairline above a three point rule

BorderSides is where the lines are placed on the page and can be one or more of the following:

- (1) - All sides
- (2) - Left
- (4) - Right
- (8) - Top
- (16) - Bottom

To extract any combination, use the bitwise operator OR (|) on this number.

BorderSpace is how close the border is to the printed page and can be one of the following:

- (1) - Middle
- (2) - Inside
- (3) - Outside
- (4) - Close to the inside
- (5) - Close to the outside

Note the use of indirection (&).

Return Value

This function does not return a value.

Example

```
FUNCTION Example()
DEFSTR Gshade, Gstyle, Bsides, Bstyle, Bspace;
GetLayoutRightLines(&Gshade, &Gstyle, &Bsides, &Bstyle, &Bspace)
TYPE("The color of the lines is: {Gshade}.[Enter]")
TYPE("The width of the gutter is: {Gstyle}.[Enter]")
IF 1 & Bsides Sides = "All" ENDIF
IF 2 & Bsides IF Sides = "" Sides = "Left" ELSE Sides = strcat$(Sides, "and Left") ENDIF ENDIF
IF 4 & Bsides IF Sides = "" Sides = "Right" ELSE Sides = strcat$(Sides, "and Right") ENDIF ENDIF
IF 8 & Bsides IF Sides = "" Sides = "Top" ELSE Sides = strcat$(Sides, "and Top") ENDIF ENDIF
IF 16 & Bsides IF Sides = "" Sides = "Bottom" ELSE Sides = strcat$(Sides, "and Bottom") ENDIF
ENDIF
```

```
TYPE("The lines are placed: {Sides}.[Enter]")
TYPE("The style of the border is: {Bstyle}.[Enter]")
TYPE("The location of the lines is: {Bspace} (where 1=Middle, 2=Inside, 3=Outside, 4=Close to
inside, and 5=Close to outside).[Enter]")
END FUNCTION
```

See also:

GetLayoutLeftLines GetLayoutPageSize GetLayoutParameters GetLayoutParmCnt GetLayoutType ModifyLayout

This function returns the type of page layout for the current page.

Syntax

GetLayoutType()

Return Value

A bit number containing the values of the All Pages bit and the Mirror Image bit and should be one of the following:

0 - Neither
512 - All Pages
1024 - Mirror Image
1536 - Both

To extract the individual bits, use the bitwise operator OR (|).

Example

```
FUNCTION Example()
Type = GetLayoutType()
IF Type & 0
    Type = "Neither all pages nor mirrored"
ELSEIF Type & 512
    Type = "All Pages"
ELSEIF Type & 1024
    Type = "Mirror Image"
ELSEIF Type & 1536
    Type = "Both all pages and mirror image"
ENDIF
Message("The current layout type is {Type}.")
END FUNCTION
```

See also:

GetLayoutLeftLines GetLayoutPageSize GetLayoutParameters GetLayoutParmCnt GetLayoutRightLines ModifyLayout

This function returns the drive and directory of the default macro directory path.

Syntax

GetMacPath$()

Return Value

A string with the drive and directory of the default macro directory path.

Example

```
FUNCTION Example()
AmiDir = GetAmiDirectory$()
WinDir = GetWindowsDirectory$()
CurDir = GetCurrentDir$()
DocPath = GetDocPath$()
StylePath = GetStylePath$()
MacPath = GetMacPath$()
Message("Ami Pro is in {AmiDir}.")
Message("Windows is in {WinDir}.")
Message("DOS reports the current directory is {CurDir}.")
Message("Ami Pro's default doc path is {DocPath}.")
Message("Ami Pro's default macro path is {MacPath}.")
Message("Ami Pro's default style path is {StylePath}.")
END FUNCTION
```

See also:

GetBackPath$ GetCurrentDir$ GetDocPath$ GetStylePath$

This function gets the name of a bookmark, a merge field, contents of a note, or a power field. Before using this function, the insertion point should be at the location of a merge field, a bookmark, or a power field.

You should use the GoToCmd function for bookmarks, merge fields, notes, and fields. Use the FieldNext function to set the insertion point.

Syntax

GetMarkText$()

Return Value

The name of the bookmark, merge field, or power field at the insertion point.

The null string ("") if the insertion point was not on one of these marks.

Example

```
FUNCTION Example()
MarkBookMark("Example", AddBookMark)
TYPE("[CtrlEND]")
MarkBookMark("Example", FindBookMark)
Text = GetMarkText$()
MarkBookMark("Example", DeleteBookMark)
IF Text != ""
    Message("Your bookmark was named ""{Text}""")
ENDIF
END FUNCTION
```

See also:

GoToCmd GoToShade GetTextBeforeCursor$
FieldAdd FieldNext MarkBookMark

This function fills an array with the names of any master file documents associated with the current document. The array may be dimensioned using the GetMasterFilesCount function.

Syntax

GetMasterFiles(&Array)

Array is the name of the array in which to place the names of the master files. Note the use of indirection (&).

Return Value

The number of files.

Example

```
FUNCTION Example()
' get the number of associated documents
Count = GetMasterFilesCount()
IF Count > 1
    ' put the names in the File array
    DIM Files(Count)
    GetMasterFiles(&Files)
    FOR I = 1 to Count
        Message(Files(I))  ' display the names
    NEXT
ELSE
    Message("No master files!")
ENDIF
END FUNCTION
```

See also:

GetMasterFilesCount SetMasterFiles AmiProIndirect

This function returns the total number of master files associated with the current file. It is useful in dimensioning an array to hold the names of all of the master files.

Syntax

GetMasterFilesCount()

Return Value

The number of master files associated with the current document.

Example

```
FUNCTION Example()
' get the number of associated documents
Count = GetMasterFilesCount()
IF Count > 1
    ' put the names in the File array
    DIM Files(Count)
    GetMasterFiles(&Files)
    FOR I = 1 to Count
        Message(Files(I))  ' display the names
    NEXT
ELSE
    Message("No master files!")
ENDIF
END FUNCTION
```

See also:

GetMasterFiles SetMasterFiles AmiProIndirect

This function determines whether the program is in Layout Mode, Outline Mode, or Draft Mode.

Syntax

GetMode()

Return Value

A number representing the current view mode and may be one of the following:

Layout (1) - Layout Mode
Draft (16) - Draft Mode
Outline (48) - Outline Mode

Example

```
FUNCTION Example()
Mode = GetMode()'What mode is the screen in?
IF Mode != 1
    LayoutMode()'If not Layout Mode, make it so.
ENDIF
END FUNCTION
```

See also:

GetViewLevel DraftMode LayoutMode OutlineMode FullPageView CustomView StandardView EnlargedView

Ami Pro documents OLE embedded in Notes 3.0 have special access to Notes. During the startup of OLE, Notes passes a read handle and a write handle to Ami Pro. These handles give Ami Pro the ability to modify or add fields to Notes. You can read whatever you have placed in the write handle.

Syntax

GetNotesWriteHandle()

Return Value

1 (TRUE) if the note was availble.

0 (FALSE) if the note was not available or the Ami Pro document was not OLE embedded.

Example

```
FUNCTION  main()
defstr  foo;
' Set the path for the Notes30 friendly DLL.
dllpath = "D:\SAMMY\SRC\AMINOTES.DLL"
' AmiPro, when embedded into a Notes 3.0 document, is given a read handle and
' a write handle to the note. These handles are used to communicate with the
' note. If the handles are zero, then no note is available. This either means
' there was an error or the current AmiPro document is not OLE embedded.
' The macro can only use the Notes Write Handle. This means that it can add fields,
' but it cannot view existing fields. However, you can read whatever you have placed
' in the write handle.
hWrite = GetNotesWriteHandle()    ' Get the write handle
Message(hWrite, "hWrite")
' Before using the given note handles, NotesInit needs to be called. Also,
' when done using the handles, NotesTerm needs to be called. These calls
' should be called once and only once. These Notes API calls are also found
' in the AMINOTES DLL by the names of NotesFriendlyInit and NotesFriendlyTerm.
' It is recommended that these be called instead of NotesTerm and NotesInit.
if (hWrite != 0)   ' Load the calls we wish to use
    DLLGetString = DllLoadLib(dllpath,"GetNotesStringField","HHCCH")
    DLLAddString = DllLoadLib(dllpath,"SetNotesStringField","HHCC")
    DLLInit = DllLoadLib(dllpath,"NotesFriendlyInit", "A")
    DLLTerm = DllLoadLib(dllpath,"NotesFriendlyTerm", "A")
    ' Were all calls loaded OK?
    if (DLLGetString AND DLLTerm AND DLLInit AND DLLAddString)
        DllCall(DLLAddString, hWrite,"MacroTest", "Macros are great.")
        ' More manipulation of fields
        DLLFreeLib(DLLInit)       ' Free up libraries used.
        DLLFreeLib(DLLTerm)
        DLLFreeLib(DLLAddString)
        DLLFreeLib(DLLGetString)
    endif
endif
end function
```

This function finds the total number of Multiple Document Interface (MDI) documents currently open. It is useful in dimensioning an array to hold the names of these files.

Syntax

GetOpenFileCount()

Return Value

The total number of documents this instance of Ami Pro currently has open.

Example

```
FUNCTION Example()
' get the number of open docs
Count = GetOpenFileCount()
IF Count > 0
    DIM Files(Count)
    ' put the files names in the Files array
    GetOpenFileNames(&Files)
    FOR I = 1 to Count
        Message(Files(I))      'display the names
    NEXT
ELSE
    Message("No open files!")
ENDIF
END FUNCTION
```

See also:

GetOpenFileNames GetOpenFileName$ FileClose FileOpen Save SaveAs

This function retrieves the full path of the current document. If more than one document is open, the name of the file with the focus is returned.

Syntax

GetOpenFileName$()

Return Value

The path of the open file.

The null string ("") if the current file is "Untitled".

Example

```
FUNCTION Example()
OpenFile = GetOpenFileName$()
Message("The current file is {OpenFile}.")
END FUNCTION
```

See also:

GetCurrentDir$ GetDocInfo$ GetDocPath$ GetMacPath$ SetDocPath GetOpenFileNames

This function fills an array with the names of all open Ami Pro documents. The array may be dimensioned using the GetOpenFileCount function.

Syntax

GetOpenFileNames(&Array)

&Array is the name of the array in which to place the list of open files.

Untitled files return the empty string ("").

Return Value

1 (TRUE) if the array was filled.

0 (FALSE) if the array was not filled.

Example

```
FUNCTION Example()
' get the number of open docs
Count = GetOpenFileCount()
IF Count > 0
    DIM Files(Count)
    ' put the file names in the Files array
    GetOpenFileNames(&Files)
    FOR I = 1 to Count
        Message(Files(I))      'display the names
    NEXT
ELSE
    Message("No open files!")
ENDIF
END FUNCTION
```

See also:

GetOpenFileCount FileClose FileOpen Save SaveAs GetOpenFileName$

This function retrieves the page number of the displayed page of the current document.

Syntax

GetPageNo()

Return Value

The number of the displayed page of the current document.

0 if you are in Draft or Outline Mode.

Example

```
FUNCTION Example()
TYPE("[CtrlEND]")
PageNo = GetPageNo()
Message("There are {PageNo} pages in this document.")
END FUNCTION
```

See also:

GetMode AtEOF GetFmtPageStr$ GetBookMarkPage

This function returns the total number of power fields of the power field type requested.

Syntax

GetPowerFieldCount(Type)

Type is the type of power field and can be one of the following:

- (0) - All the power fields (except document variables)
- (3) - General fields
- (4) - Sequence power fields
- (5) - Set power fields
- (6) - Button power fields
- (7) - PrintEscape power fields
- (8) - Index mark power fields
- (9) - User power fields
- (10) - Document variables
- (11) - TOC power fields
- (12) - MergeField power fields

Return Value

The number of power fields of the type requested.

Example

```
FUNCTION Example()
type = 0
' get the number of power fields
Count = GetPowerFieldCount(type)
IF Count > 0
    DIM Power(Count)
    ' get the power fields
    GetPowerFields(type, &Power)
    FOR i = 1 to Count
        ' get the id and type from the power field
        pfid = strfield$(Power(i), 1, ",")
        pftype = strfield$(Power(i), 2, ",")
        pageno = GetPowerFieldPage(pfid, pftype)
        message("Power Field #{i} type {pftype} is on page {pageno}.")
    NEXT
ELSE
    message("No power fields in this document.")
ENDIF
END FUNCTION
```

See also:

GetPowerFieldPage GetPowerFields GoToPowerField

This function returns the page number of the specified power field. The ID and type may be found using the GetPowerFields function.

Syntax

GetPowerFieldPage(ID, Type)

ID is the number assigned to that specific power field.

Type is the type of power field that is located and can be one of the following:

- (0) - All the power fields (except document variables)
- (3) - General fields
- (4) - Sequence power fields
- (5) - Set power fields
- (6) - Button power fields
- (7) - PrintEscape power fields
- (8) - Index mark power fields
- (9) - User power fields
- (10) - Document variables
- (11) - TOC power fields
- (12) - MergeField power fields

Return Value

The page number of the specified power field.

Example

```
FUNCTION Example()
type = 0
' get the number of power fields
Count = GetPowerFieldCount(type)
IF Count > 0
    DIM Power(Count)
    ' get the power fields
    GetPowerFields(type, &Power)
    FOR i = 1 to Count
        ' get the id and type from the power field
        pfid = strfield$(Power(i), 1, ",")
        pftype = strfield$(Power(i), 2, ",")
        pageno = GetPowerFieldPage(pfid, pftype)
        message("Power Field #{i} type {pftype} is on page {pageno}.")
    NEXT
ELSE
    message("No power fields in this document.")
ENDIF
END FUNCTION
```

See also:

GetPowerFieldCount GoToPowerField GetPowerFields

This function fills an array with any power fields located in the current document. The array may be dimensioned using the GetPowerFieldCount function.

Syntax

GetPowerFields(Type, &Array)

Type is the type of power field located and can be one of the following:

- (0) - All the power fields (except document variables)
- (3) - General fields
- (4) - Sequence power fields
- (5) - Set power fields
- (6) - Button power fields
- (7) - PrintEscape power fields
- (8) - Index mark power fields
- (9) - User power fields
- (10) - Document variables
- (11) - TOC power fields
- (12) - MergeField power fields

Array is the name of the array in which to place the power fields. The format returned in the array is "ID, type, power field." Document variables are only returned when type 10 is specified. The format for document variables are "name=value".

Note the use of indirection (&).

Return Value

The number of power fields returned in the array.

Example

```
FUNCTION Example()
Count = GetPowerFieldCount(9) ' Get the User powerfields
If count <1
    Message("No powerfields are in this document.")
    Exit Function          ' No powerfields, Exit macro
Endif
DIM Power(Count)
GetPowerFields(9, &Power)
FOR i = 1 to Count
    pfid = strfield$(Power(i), 1, ",")
    pftype = strfield$(Power(i), 2, ",")
    pageno = GetPowerFieldPage(pfid, pftype)
NEXT
GoToPowerField(pfid, pftype)
END FUNCTION
```

See also:

GetPowerFieldCount GetPowerFieldPage

This function retrieves an entry from a text file such as the Windows' WIN.INI file or the AMIPRO.INI file.

Syntax

GetProfileString$(Section, Key[, FileName])

Section is the section in the file to look in. If this is the empty string (""), the [Ami Pro] section is searched.

Key is the desired entry in the file. If this is zero (0), or a null string (""), all the keys from the [Ami Pro] section are returned separated with a tilde (~).

FileName is the optional file to look in. If this parameter is omitted, the Windows' WIN.INI file is used.

Return Value

A string with the contents of the entry.

The null string ("") if the entry doesn't exist.

Example

```
FUNCTION Example()
'  Retrieve information from the AmiPro.ini and Win.ini
Name = GetProfileString$("AmiPro","UserName","AmiPro.Ini")
WallPaper = GetProfileString$("DeskTop","WallPaper","Win.Ini")
Message("Your name is {Name} and your wallpaper is {WallPaper}.")
'  Retrieve a list of all the key entries under the SmartIcons section
IconEntries = GetProfileString$("SmartIcons",0,"AmiPro.Ini")
Message(IconEntries)
END FUNCTION
```

See also:

GetMacPath$ WriteProfileString GetAmiDirectory$

This function returns the file name, including full path, of the currently running macro. This function is useful in allowing a function to call another function in the same file, even if the file name has been changed.

Syntax

GetRunningMacroFile$()

Return Value

A string containing the full path and file name of the currently running macro.

Example

```
FUNCTION Example()
MacFile = GetRunningMacroFile$()
MacName = GetRunningMacroName$()
Message("This macro file is {MacFile}.")
Message("This function is {MacName}.")
END FUNCTION
```

See also:

GetMacPath$ GetRunningMacroName$

This function returns the name of the currently running function.

Syntax

GetRunningMacroName$()

Return Value

A string containing the name of the currently running macro function.

Example

```
FUNCTION Example()
MacFile = GetRunningMacroFile$()
MacName = GetRunningMacroName$()
Message("This macro file is {MacFile}.")
Message("This function is {MacName}.")
END FUNCTION
```

See also:

GetMacPath$ GetRunningMacroFile$

This function returns the text located at the beginning of a paragraph style. The text is the text that is specified in Style/Modify Style/Bullets & numbers.

Syntax

GetSpecialEffects$()

Return Value

The text that is located at the beginning of the paragraph style.

0 (UserCancel) if the user canceled the function.

-2 (GeneralFailure) if the special effects could not be retrieved.

Example

```
FUNCTION Example()
Effects = GetSpecialEffects$()
IF Effects = ""
    Message("There is no text in special effects on this paragraph.")
ELSE
    Message("{Effects} is appended to this style.")
ENDIF
END FUNCTION
```

See also:

ModifyStyle ModifyLines ModifyAlignment ModifyReflow ModifyBreaks ModifySelect ModifyEffects ModifySpacing ModifyFont ModifyTable

This function returns the number of styles in the current document.

Syntax

GetStyleCount()

Return Value

The number of styles in the current document.

Example

```
FUNCTION Example()
Count = GetStyleCount()
DIM Styles(Count)
GetStyleNames(&Styles)
END FUNCTION
```

See also:

GetStyleNames

This function determines the name of the paragraph style assigned to the current paragraph.

Syntax

GetStyleName$()

Return Value

A string with the name of the paragraph style.

Example

```
FUNCTION Example()
curstyle = GetStyleName$()
newstyle = "Number List"
SetStyle(newstyle)
TYPE("This is what the {newstyle} style looks like.[Enter]")
SetStyle(curstyle)
END FUNCTION
```

See also:

ModifyStyle SelectStyle SetStyle ShowStylesBox ToggleStylesBox

This function fills an array with the names of the styles in the current document. The array may be dimensioned using the GetStyleCount function.

Syntax

GetStyleNames(&Array)

Array is the name of the array in which to place the names of the styles.

Note the use of indirection (&).

Return Value

This function returns the style names.

Example

```
FUNCTION Example()
Count = GetStyleCount()
DIM Styles(Count)
GetStyleNames(&Styles)
END FUNCTION
```

See also:

GetStyleCount

This function returns the drive and directory of the default style sheet path.

Syntax

GetStylePath$()

Return Value

A string containing the current default paragraph style sheet path.

Example

```
FUNCTION Example()
AmiDir = GetAmiDirectory$()
WinDir = GetWindowsDirectory$()
CurDir = GetCurrentDir$()
DocPath = GetDocPath$()
StylePath = GetStylePath$()
MacPath = GetMacPath$()
Message("Ami Pro is in {AmiDir}.")
Message("Windows is in {WinDir}.")
Message("DOS reports the current directory is {CurDir}.")
Message("Ami Pro's default doc path is {DocPath}.")
Message("Ami Pro's default macro path is {MacPath}.")
Message("Ami Pro's default style path is {StylePath}.")
END FUNCTION
```

See also:

GetBackPath$ GetDocPath$ GetMacPath$ SetBackPath SetDocPath SetStylePath

This function retrieves the text between the beginning of the paragraph the insertion point is in and the insertion point.

Syntax

GetTextBeforeCursor$()

Return Value

A string with the contents of the paragraph prior to the insertion point.

Example

```
FUNCTION Example()
Text = GetTextBeforeCursor$()
Length = len(Text)
Message("You are {Length} characters into the current paragraph.")
END FUNCTION
```

See also:

CurChar$ CurShade$ CurWord$ CursorPosition$ GetMarkText$ TYPE

This function determines the amount of time that has passed since Windows was started.

Syntax

GetTime()

Return Value

The number of milliseconds since Windows was started.

Example

```
FUNCTION Example()
CurrentTime = GetTime()
Message("{CurrentTime} milliseconds since Windows was started.")
END FUNCTION
```

See also:

Pause RunLater Now FormatDate$ FormatTime$

This function returns the current Layout Mode View level.

Syntax

GetViewLevel()

Return Value

A number representing the current view level:

FullPage (201) - Full Page View
Custom (202) - Custom View
Standard (203) - Standard View
Enlarged (204) - Enlarged View

Example

```
FUNCTION Example()
Level = GetViewLevel()
SWITCH Level
CASE 201
Level = "Full Page View"
CASE 202
Level = "Custom View"
CASE 203
Level = "Standard View"
CASE 204
Level = "Enlarged View"
ENDSWITCH
Message("The current view level is at {Level}.")
END FUNCTION
```

See also:

GetMode DraftMode LayoutMode FullPageView CustomView StandardView EnlargedView FacingView

This function gets the current view level from the View/View Preferences function. It is used to change the view level in the ViewPreferences function.

This function must be used with the GetViewPrefOpts function.

Syntax

GetViewPrefLevel()

Return Value

The value of the view level which is a number from 10-400.

Example

```
FUNCTION TurnOffNotes()
Viewopts=GetViewPrefOpts() ' get the View Preferences options
Viewlevel=GetViewPrefLevel() ' get the View Preferences level
IF(Viewopts & 2048) ' notes turned on?
    Viewopts=Viewopts - 2048 ' turn notes display off in options
ENDIF
ViewPreferences(Viewopts, Viewlevel) ' set new options
END FUNCTION
```

See also:

GetViewPrefOpts

This function retrieves the current Options parameter from the View Preferences function. This function is required to change the View Preferences in a macro when you need to know the current options to modify that value.

You must get the view options as well as the view level using the GetViewPrefLevel function to change View Preferences.

Syntax

GetViewPrefOpts()

Return Value

The value of the View Preferences option.

Example

```
FUNCTION TurnOffNotes()
Viewopts=GetViewPrefOpts() ' get the View Preferences options
Viewlevel=GetViewPrefLevel() ' get the View Preferences level
IF(Viewopts & 2048) ' notes turned on?
    Viewopts=Viewopts - 2048 ' turn notes display off in options
ENDIF
ViewPreferences(Viewopts, Viewlevel) ' set new options
END FUNCTION
```

See also:

GetViewPrefLevel

This function returns the Windows directory.

Syntax

GetWindowsDirectory$()

Return Value

This function returns the Windows directory.

Example

```
FUNCTION Example()
ASCIIOptions(PCASCII)
WinDir = GetWindowsDirectory$()
FileOpen("{WinDir}PRINTERS.TXT", 16, "ASCII")
END FUNCTION
```

See also:

GetAmiDirectory$ GetDocPath$ GetStylePath$ GetMacPath$

This function inserts the named glossary item into the document at the insertion point. Choosing this function is equivalent to choosing Edit/Insert/Glossary Record. To set the glossary file before inserting items, use the GlossSet function.

Syntax

Glossary(Item)

Item is the name of the glossary item to insert.

To display the Glossary dialog box and allow the user to select the name of the glossary item to insert: **Glossary**

Return Value

1 (TRUE) if a glossary item was inserted.

0 (UserCancel) if the user canceled the menu function.

-2 (GeneralFailure) if the item could not be inserted or if the item was not found.

Example

```
FUNCTION Example()
Gloss = Query$("What glossary file do you want to use?")
GlossSet(Gloss)
Glossary
END FUNCTION
```

See also:

GlossSet GlossaryAdd

This function adds the currently selected text to the current glossary file. It is the equivalent to choosing Edit/Mark Text/Glossary.

Syntax

GlossaryAdd(Item, FileName, Flag)

Item is the glossary item name to insert into the glossary data file.

FileName is the name of the glossary file or NewWave object in which to insert the new item.

Flag is a number that defines whether the Filename parameter is a file name or a NewWave object name.

FALSE (0) - Filename parameter is a file name

TRUE (1) - Filename parameter is an object name

To display the Mark Glossary dialog box and allow the user to select the item name and glossary data file: **GlossaryAdd**

Return Value

1 (TRUE) if the glossary record was inserted.

0 (NoAction) if no action was taken.

-2 (GeneralFailure) if the glossary record could not be inserted, if the data file is not an Ami Pro file, or if the data file is encrypted.

Example

```
FUNCTION Example()
WHILE "" = CurShade$()
    UserControl("Select the text to place in the glossary, then choose Resume...")
WEND
GlossaryAdd
END FUNCTION
```

See also:

GlossSet Glossary

This function selects the named glossary file to use as the source file for insertion of glossary items. Choosing this function is equivalent to choosing Edit/Insert/Glossary Record/Data File.

Syntax

GlossSet(FileName)

FileName is the name of the glossary file from which to select records.

To display the Glossary dialog box and allow the user to select the name of the glossary file: **Glossary**

Return Value

1 (TRUE) if a glossary file was selected.

-2 (GeneralFailure) if the file could not be selected or if the file was not found.

Example

```
FUNCTION Example()
Gloss = Query$("What glossary file do you want to use?")
GlossSet(Gloss)
Glossary
END FUNCTION
```

See also:

Glossary GlossaryAdd

This function goes to the next location of an item after using the GoToCmd function. Choosing this function is equivalent to choosing Edit/Go To and selecting Go To ^H.

Syntax

GoToAgain()

Return Value

1 (TRUE) if the item was located.

0 (UserCancel) if the user canceled the function.

-3 if the input was invalid.

-7 if the item to Go To could not be found.

-2 if any other error.

Example

```
FUNCTION Example()
WHILE not AtEOF()
    GoToCmd(4, 0, GoField)
    GoToShade(4, 0, GoField)
    TYPE("[DEL]")
WEND
END FUNCTION
```

See also:

GoToCmd

This function allows the user to go to a page or a mark. Choosing this function is equivalent to choosing Edit/Go To.

Syntax

GoToCmd(Which, Page, Name)

Which is a flag that determines what type of object to Go To, and can be one of the following:

GoPage (1) - Goes to the page number specified in the Page Parameter.

GoFirst (2) - Goes to the first page of the document.

GoLast (3) - Goes to the last page of the document.

GoMark (4) - Goes to the mark specified in the MarkName parameter.

To go to a specific page, set the Which parameter to 1, set the Page parameter to the page number you want to go to, and set the MarkName parameter to 0.

Parameters not necessary for a specific function may be set to 0. However, Keystroke Record may use other numbers. These numbers are ignored, but must be present.

To go to the first or last page, set the Which parameter to either 2 or 3, and set the Page parameter and the MarkName parameter to 0.

To go to a mark, set the Which parameter to 4, set the Page parameter to 0, and set the MarkName parameter to the value for the mark you want to go to.

Page is the page number to go to if you want to go to a page number.

Name is the type of mark to go to if you want to go to a mark. It can be one of the following:

GoFrame (1) - Go to the next frame

GoHeader (2) - Go to the header area

GoFooter (3) - Go to the footer area

GoField (4) - Go to the next field

GoTab (5) - Go to the tab ruler

GoRuler (6) - Go to the next tab ruler mark in the text

GoFootnote (7) - Go to the next footnote mark

GoFoottext (8) - Go to the text of the footnote

GoPageBreak (9) - Go to the next hard page break

GoNote (10) - Go to the next note

GoLayout (11) - Go to the next layout change

GoBookmark (12) - Go to the next Bookmark

GoHF (13) - Go to the next floating Header/Footer mark

To display the Go To dialog box and allow the user to select the parameters for the Go To function: **GoToCmd**

Return Value

1 (TRUE) if the desired location was reached.

0 (UserCancel) if the user canceled the function.

-3 if the desired mark could not be found (for example, going to a frame in Draft Mode).

-7 if the desired location could not be found.

-2 if any other error.

Example

```
FUNCTION Example()
WHILE not AtEOF()
```

```
    GoToCmd(4, 0, GoField)
    GoToShade(4, 0, GoField)
    TYPE("[DEL]")
WEND
END FUNCTION
```

See also:

GoToShade GetMarkText$ Replace GoToAgain

This function moves the insertion point to the specified power field. The ID and type may be found using the GetPowerFields function.

Syntax

GoToPowerField(ID, Type)

ID is the number assigned to that specific power field.

Type is the type of power field that is located and can be one of the following:

- (0) - All the power fields
- (3) - General fields
- (4) - Sequence power fields
- (5) - Set power fields
- (6) - Button power fields
- (7) - PrintEscape power fields
- (8) - Index mark power fields
- (9) - User power fields
- (11) - TOC power fields
- (12) - MergeField power fields

Return Value

1 (TRUE) if the power field is located.

0 (FALSE) if the power field is not located.

Example

```
FUNCTION Example()
Count = GetPowerFieldCount(9) ' Get the User powerfields
If count <1
    Message("No powerfields are in this document.")
    Exit Function          ' No powerfields, Exit macro
Endif
DIM Power(Count)
GetPowerFields(9, &Power)
FOR i = 1 to Count
    pfid = strfield$(Power(i), 1, ",")
    pftype = strfield$(Power(i), 2, ",")
    pageno = GetPowerFieldPage(pfid, pftype)
NEXT
GoToPowerField(pfid, pftype)
END FUNCTION
```

See also:

GetPowerFieldCount GetPowerFieldPage GetPowerFields

This function allows the user to go to a mark. The difference between this function and the GoToCmd function is that when this function is executed, the text between the insertion point and location gone to is shaded.

Syntax

GoToShade(4, 0, Name)

Name is the type of mark to go to if you want to go to a mark. It may be one of the following:

GoField (4) - Go to the next field
GoRuler (6) - Go to the next tab ruler mark in the text
GoFootnote (7) - Go to the next footnote mark
GoPageBreak (9) - Go to the next hard page break
GoNote (10) - Go to the next note
GoLayout (11) - Go to the next layout change
GoBookmark (12) - Go to the next Bookmark
GoHF (13) - Go to the next floating Header/Footer mark

Return Value

1 (TRUE) if the desired location was reached.
-3 if the input is invalid.
-7 if the item could not be found.
-2 if any other error.

Example

```
FUNCTION Example()
WHILE not AtEOF()
    GoToCmd(4, 0, GoField)
    GoToShade(4, 0, GoField)
    TYPE("[DEL]")
WEND
END FUNCTION
```

See also:

GoToCmd GetMarkText$

This function scales a graphic. Choosing this function is equivalent to choosing Frame/Graphics Scaling.

Syntax

GraphicsScaling(Options, Percentage, Units, Height, Width, Rotate)

Options is a bit number with scaling options, and should be set to one of the following:

OriginalSize (1) - Display as original size
FitInFrame (2) - Size the graph to fit in the frame
PercentSize (4) - Use the Percentage parameter to size the graph
CustomSize (8) - Use the height and width parameters to determine the graph's size
(16) - Maintain the aspect ratio of the graph

One of the four sizing parameters must be used. If the OriginalSize option is used, all other parameters are ignored. If the FitInFrame parameter is used, the MaintainAspect option can also be used and the other parameters are ignored. If the PercentSize option is used, the Percentage parameter is used to determine the size of the graph and all other parameters are ignored. If the CustomSize option is used, the Height and Width options are used to determine the size of the graph.

The values may be combined to determine the Options parameter.

Percentage is the percentage of original size to display the graph if that is the option.

Units is the units of measurement to use when using the dialog box to set the height and width and can be one of the following:

Inches (1) - Display measurements in inches
CM (2) - Display measurements in centimeters
Picas (3) - Display measurements in picas
Points (4) - Display measurements in points

Height is the height of the graph in the selected units.

Width is the width of the graph in the selected units.

Rotate is the amount to rotate the graph and must be between 0 and 360 degrees to indicate the amount the graph should be rotated.

To display the Graphics Scaling dialog box and allow the user to select the options for graphics scaling: **GraphicsScaling**

Return Value

This function returns 1.

Example

```
FUNCTION Example()
' The following example causes the graphic to Fit in Frame and maintain
'  its aspect ratio. A frame with a graphic must be selected before executing
' this function
Options = FitInFrame + 16
GraphicsScaling(Options, 0, Inches, 0, 0, 90)
END FUNCTION
```

See also:

AddFrame AddFrameDLG ImportPicture DrawingMode ChartingMode ImageProcessing

This function grays or ungrays a menu item. When a menu item is grayed, it appears on the menu, but the user cannot select it. Use this function to indicate that the function is currently unavailable.

Syntax

GrayMenuItem(BarID, MenuName, Item, State)

BarID is the identification number of the menu bar returned from the AddBar function. To use the Ami Pro menu bar, use 1.

MenuName is the name of the menu where the item to be grayed is located. The name must include any ampersand (&) characters used in the menu name. An ampersand is placed before a character that has an underline.

Item is the name of the menu item to be grayed, and must match the actual menu item, including any ampersand (&) characters. If the menu item has a shortcut key, you must press **TAB**, type a ^, and then type the appropriate letter as part of the item name.

State is either 1 (TRUE) to gray the menu item, or 0 (FALSE) to ungray the menu item.

Return Value

1 (TRUE) if the item was successfully grayed or ungrayed.

0 (FALSE) if the item could not be grayed/ungrayed, or if an invalid BarID, MenuName, or ItemName was used.

Example

```
FUNCTION Example()
' This example might be used if you wanted to prevent users from applying special
' effects attributes to text of a file that may be exported to another format
GrayMenuItem(1, "&Text", "Special &Effects...", 1) ' gray the menu item
OnCancel RestoreIt ' in case the user cancels
' pause the macro
UserControl("Inspect the ""Text"" menu then select Resume")
RestoreIt:
GrayMenuItem(1, "&Text", "Special &Effects...", 0) ' ungray the menu item
END FUNCTION
```

See also:

AddBar AddMenu AddMenuItem AddMenuItemDDE ChangeMenuAction CheckMenuItem DeleteMenu DeleteMenuItem RenameMenuItem ShowBar

This function groups the selected frames. Choosing this function is equivalent to choosing Frame/Group. If the frames were already grouped, this function ungroups them.

Syntax

GroupFrames()

Return Value

This function returns 1.

Example

```
FUNCTION Example()
' note - two or more frames must be selected before using this function
GroupFrames()
END FUNCTION
```

See also:

AddFrame AddFrameDLG GotoCmd

This function goes to the header/footer or displays the Floating Header/Footer dialog box. Choosing this function is equivalent to choosing Page/Header/Footer.

Syntax

HeaderFooter(Function)

Function is a flag that can be one of the following:

(1) - Go to the header
(2) - Go to the footer
(3) - Display Floating Header/Footer dialog box

To display the Headers & Footers dialog box: **HeaderFooter**

Return Value

This function returns:

0 (UserCancel) if the user cancels the function.

1 (TRUE) if the header or footer is found or the Floating Header/Footer dialog box displays.

-6 (NoMemory) if the function fails because of insufficient memory.

Example

```
FUNCTION Example()
HeaderFooter(1)   ' Go to the header
Type("This is a test.")
END FUNCTION
```

See also:

FloatingHeader GoToCmd ModifyLayout PageNumber

This function assigns the selected rows of a table to be the heading rows for each page of a table. This function toggles on and off. Choosing this function is equivalent to choosing Table/Headings.

Syntax

Heading()

Return Value

This function returns:

1 (TRUE) if the rows were successfully set to be the heading for each page.

-2 (GeneralFailure) if the rows were not set.

Example

```
FUNCTION Example()
Heading()
END FUNCTION
```

See also:

ConnectCells DeleteColumnRow DeleteEntireTable InsertColumnRow TableLayout Tables

This function displays the online Help for Ami Pro. Choosing this function is equivalent to choosing Help/Contents, but it does not select a Help topic automatically. Because Help displays in a window other than the regular Ami Pro window, further macro functions which cause a repainting of the Ami Pro window cause the Help window to be replaced by the Ami Pro window.

If this function is used, it should be the last function used in the macro.

Syntax

Help()

Return Value

This Function Returns:

1 (TRUE) if the Help window was displayed.

-2 (GeneralFailure) if the Help window could not be displayed for some other reason.

-6 (NoMemory) if the function failed because of insufficient memory.

Example

```
FUNCTION Example()
Help()
END FUNCTION
```

See also:

About EnhancementProducts HowDoIHelp KeyboardHelp MacroHelp UpgradeHelp UsingHelp

This function hides the set of SmartIcons. Choosing this function is equivalent to choosing View/Hide SmartIcons.

Syntax

HideIconBar()

Return Value

1 (TRUE) if the icon bar was hidden.

0 (NoAction) if no action was taken because the icon bar was already hidden.

Example

```
FUNCTION Example()
ReturnValue = HideIconBar()
IF ReturnValue != 1
    IF Decide("Icon bar already hidden! Would you like it displayed?")
        ShowIconBar()
    ENDIF
ENDIF
END FUNCTION
```

See also:

ChangeIcons GetIconPalette IconCustomize SetIconPath SetIconSize ShowIconBar ToggleIconBar ViewPreferences

This function removes the Styles Box from the screen. Choosing this function is equivalent to choosing View/Hide Styles Box.

Syntax

HideStylesBox()

Return Value

1 (TRUE) if the Styles Box is hidden.

0 (NoAction) if the Styles Box was already hidden.

Example

```
FUNCTION Example()
ReturnValue = HideStylesBox()
IF ReturnValue != 1
    IF Decide("Styles Box already hidden! Would you like it displayed?")
        ShowStylesBox()
    ENDIF
ENDIF
END FUNCTION
```

See also:

ShowStylesBox ToggleStylesBox ViewPreferences

This function removes the tab ruler from the top of the screen. Choosing this function is equivalent to choosing View/Hide Ruler.

Syntax

HideTabRuler()

Return Value

1 (TRUE) if the tab ruler was hidden.

0 (NoAction) if no action was taken because the tab ruler was already hidden.

Example

```
FUNCTION Example()
HideTabRuler()
END FUNCTION
```

See also:

ShowTabRuler ToggleTabRuler ViewPreferences

This function turns on or off the Windows hourglass mouse pointer. The original state of the hourglass can be determined from the function's return value. If a macro changes the state of the hourglass mouse pointer, it must restore the original state before exiting. If it does not restore the mouse pointer, the insertion point is incorrect from that point on.

Syntax

HourGlass(State)

State is either On (1) to turn on the hourglass mouse pointer or Off (0) to turn it off.

Return Value

1 (TRUE) if the hourglass was on prior to using the hourglass function.

0 (FALSE) if the hourglass was off.

Example

```
FUNCTION Example()
IgnoreKeyboard(1)
HourGlass(1)
StatusBarMsg("Importing graphic file...")
ImportPicture("BMP", "C:\AMIPRO\ICONS\123W.BMP", ".BMP", 0)
StatusBarMsg("")
HourGlass(0)
IgnoreKeyboard(0)
END FUNCTION
```

See also:

AnswerMsgBox Messages Pause SingleStep StatusBarMsg

This function displays the online Help for Ami Pro. Choosing this function is equivalent to choosing Help/How Do I?, but it does not select a Help topic automatically. Because Help displays in a window other than the regular Ami Pro window, further macro functions which cause a repainting of the Ami Pro window cause the Help window to be replaced by the Ami Pro window.

If this function is used, it should be the last function used in the macro.

Syntax

HowDoIHelp()

Return Value

1 (TRUE) if the Help window was displayed.

-2 (GeneralFailure) if the Help window could not be displayed for some other reason.

-6 (NoMemory) if the function failed because of insufficient memory.

Example

```
FUNCTION Example()
HowDoIHelp()
END FUNCTION
```

See also:

About EnhancementProducts Help KeyboardHelp MacroHelp UpgradeHelp UsingHelp

This function moves the set of SmartIcons to the bottom of the screen. Choosing this function is equivalent to choosing Tools/SmartIcons and selecting Bottom for the position. A macro must be edited to insert this non-recordable function.

Syntax

IconBottom()

Return Value

This function returns 1.

Example

```
FUNCTION Example()
IconBottom()
END FUNCTION
```

See also:

IconCustomize IconFloating IconLeft IconRight IconTop ToggleIconBar ShowIconBar HideIconBar SetIconSize

This function displays the SmartIcons dialog box that allows the user to customize the icon set. Choosing this function is equivalent to choosing Tools/SmartIcons. A macro must be edited to insert this non-recordable function.

Syntax

IconCustomize()

Return Value

1 (TRUE) if the icons were customized.

0 (UserCancel) if the user canceled the function.

Example

```
FUNCTION Example()
IconCustomize()
END FUNCTION
```

See also:

IconBottom IconFloating IconLeft IconRight IconTop ToggleIconBar ShowIconBar HideIconBar SetIconSize

This function moves the set of SmartIcons to a floating position on the screen. Choosing this function is equivalent to choosing Tools/SmartIcons and selecting Floating for the position. A macro must be edited to insert this non-recordable function.

Syntax

IconFloating()

Return Value

This function returns 1.

Example

```
FUNCTION Example()
IconFloating()
END FUNCTION
```

See also:

IconBottom IconCustomize IconLeft IconRight IconTop ToggleIconBar ShowIconBar HideIconBar SetIconSize

This function moves the set of SmartIcons to the left side of the screen. Choosing this function is equivalent to choosing Tools/SmartIcons and selecting Left for the position. A macro must be edited to insert this non-recordable function.

Syntax

IconLeft()

Return Value

This function returns 1.

Example

```
FUNCTION Example()
IconLeft()
END FUNCTION
```

See also:

IconBottom IconCustomize IconFloating IconRight IconTop ToggleIconBar ShowIconBar HideIconBar SetIconSize

This function moves the set of SmartIcons to the right side of the screen. Choosing this function is equivalent to choosing Tools/SmartIcons and selecting Right for the position. A macro must be edited to insert this non-recordable function.

Syntax

IconRight()

Return Value

This function returns 1.

Example

```
FUNCTION Example()
IconRight()
END FUNCTION
```

See also:

IconBottom IconCustomize IconFloating IconLeft IconTop ToggleIconBar ShowIconBar HideIconBar SetIconSize

This function moves the set of SmartIcons to the top of the screen. Choosing this function is equivalent to choosing Tools/SmartIcons and selecting Top for the position. A macro must be edited to insert this non-recordable function.

Syntax

IconTop()

Return Value

This function returns 1.

Example

```
FUNCTION Example()
IconTop()
END FUNCTION
```

See also:

IconBottom IconCustomize IconFloating IconLeft IconRight ToggleIconBar ShowIconBar HideIconBar SetIconSize

This function determines whether macro execution is interrupted if the user presses any key or the **ESC** key. Normally, macro execution pauses if the user presses any key.

Syntax

IgnoreKeyboard(State)

State is either On (1) to ignore keyboard interruptions or Off (0) to honor them.

Return Value

This function does not return a value.

Example

```
FUNCTION Example()
IgnoreKeyboard(1)
HourGlass(1)
StatusBarMsg("Importing graphic file...")
ImportPicture("BMP", "C:\AMIPRO\ICONS\123W.BMP", ".BMP", 0)
StatusBarMsg("")
HourGlass(0)
IgnoreKeyboard(0)
END FUNCTION
```

See also:

AnswerMsgBox Messages SingleStep HourGlass Pause StatusBarMsg

This function adds the Image menu to the main menu bar. Choosing this function is equivalent to choosing Tools/Image Processing.

A frame containing a gray scale TIFF image must be selected before calling this function.

Syntax

ImageProcessing()

Return Value

1 (TRUE) if the Image menu was added to the main menu bar.

-2 (GeneralFailure) if the Image menu could not be added.

Example

```
FUNCTION Example()
LayoutMode()
WHILE not IsFrameSelected()
    UserControl("Select a frame with a gray scale tiff and choose Resume...")
WEND
ImageProcessing()
END FUNCTION
```

See also:

AddFrame AddFrameDLG GraphicsScaling FrameLayout ManualFrame

This function allows you to import a non-Ami Pro file or export a file to another file format. If you are importing a file, the macro should make sure the insertion point is at the location in the file where the new file should be imported. If exporting a file, the file to be exported should be on the screen.

Syntax

ImportExport(Which, FileName, FilterName, CopyImage)

Which indicates if the file should be imported or exported. The Which parameter should be set according to the list below:

Export (1) - Exports the file onscreen to a new file

Import (0) - Imports the specified file to the current document

FileName is the full path of the file being imported or the destination path of the file to export to.

FilterName is the name of the filter as listed in the Open dialog box or in the AMIPRO.INI.

CopyImage is a flag indicating whether to copy the graphic file or use the information in the original graphics file. The CopyImage parameter should be set to 1 (True) if the graphic file should be copied into the document. It should be set to 0 (False) if the original graphic file should be referred to when the graphic is displayed.

ASCII

IMPORT and EXPORT syntax:

ImportExport(Which, FileName, FilterName, CopyImage, CRLF, Type, KeepStyle)

CRLF - Carriage return/linefeed options

LINE - Use carriage return/linefeed characters at end of line

PARA - Use carriage return/linefeed characters at end of paragraph

Type - is the type of ASCII file to import or export

ASCII (4) - 7 bit ASCII file

PCASCII (8) - 8 bit PC ASCII file

ANSI (16) - 8 bit ANSI file

KeepStyle allows you to keep the style names

KeepStyle - Keep style names

DisplayWrite 4

EXPORT Syntax:

ImportExport(Which, FileName, FilterName, CopyImage, StyleBox)

StyleBox - Choices for the Styles box

Apply - Apply styles

Ignore (5) - Ignore styles

DCA/RFT

EXPORT Syntax:

ImportExport(Which, FileName, FilterName, CopyImage, StyleBox)

StyleBox - Choices for the Styles box

Apply - Apply styles

Ignore (5) - Ignore styles

Lotus 1-2-3 Version 3.0 and higher

IMPORT Syntax:

ImportExport(Which, FileName, FilterName, CopyImage, RangeBox[, Range], StyleBox)

RangeBox - Choices for the Range box

Entire - Entire file

Active - Active worksheet

Range - Range. This parameter must be followed by another giving the exact range.

Range - Range if the RangeBox parameter has a range

StyleBox - Choices for the Styles box

Convert - Convert styles

Apply - Apply styles

Keep - Keep style names

Ignore (5) - Ignore styles

Manuscript

IMPORT Syntax:

ImportExport(Which, FileName, FilterName, CopyImage, StyleBox, Flag)

StyleBox - Choices for the Styles box

Convert - Convert levels into styles

Apply - Apply levels

Flag tells Manuscript whether to prompt the user if a graphics file cannot be found to import. The default is False.

TRUE (1) - the dialog box prompts you if Manuscript cannot find the file in the specified path.

FALSE (0) - the dialog box does not prompt you if Manuscript cannot find the file and the graphic is not to be imported.

Rich Text Format, Word for Windows 1.x and 2.0

IMPORT Syntax:

ImportExport(Which, FileName, FilterName, CopyImage, StyleBox)

StyleBox - Choices for the Styles box

Convert - Convert styles

Apply - Apply styles

Keep - Keep style names

Ignore (5) - Ignore styles

WordPerfect 5.0 and 5.1

IMPORT Syntax:

ImportExport(Which, FileName, FilterName, CopyImage, StyleBox)

StyleBox - Choices for the Styles box

Apply - Apply styles

Ignore (5) - Ignore styles

Style - Import style sheet

EXPORT Syntax:

ImportExport(Which, FileName, FilterName, CopyImage, StyleBox, Version)

StyleBox - Choices for the Styles box

Convert - Convert styles

Apply - Apply styles

Ignore (5) - Ignore styles

Version is the version of WordPerfect

TRUE (1) - Wordperfect 5.0 format

FALSE (0) - Wordperfect 5.1 format

For all exports: If the export file already exists, a message box prompts you whether or not to overwrite the existing file.

This function does not display a dialog box.

Return Value

1 (TRUE) if the file was successfully imported/exported.

0 (UserCancel) if the user canceled the function.

-2 (GeneralFailure) if the file could not be imported because the file was not the appropriate type or if the file could not be found.

Example

```
FUNCTION Example()
ImportExport(Import, "C:\AUTOEXEC.BAT", "ASCII", 0, "PARA", PCASCII, 0)
END FUNCTION
```

See also:

FileOpen SaveAs ImportPicture ASCIIOptions ImportText

This function imports a graphic into the current document. If an empty frame is selected, the graphic is placed in it. If no frame is selected, this function creates a 1" by 1" frame and imports the graphic into it. This function is equivalent to choosing File/Import Picture.

Syntax

ImportPicture(App, FileName, FileExt, CopyImage)

App is the name of the application filter name. This name may be extracted from the list of file types in the Import Picture dialog box. The name may also be found in the AMIPRO.INI file.

FileName is the name of the file and extension to be imported and the full path, if necessary.

FileExt is the extension of the file to be imported. There must be a period (.) before the extension name.

CopyImage can be one of the following values:

(1) - Graphics file copied into the document.

(0) - Original graphics file referred to when the graphic is displayed.

Return Value

1 (TRUE) if the picture was imported.

0 (UserCancel) if the user canceled the function.

-2 (GeneralFailure) if the picture was not imported.

-6 (NoMemory) if the function failed because of insufficient memory.

Example

```
FUNCTION Example()
IgnoreKeyboard(1)
HourGlass(1)
StatusBarMsg("Importing graphic file...")
ImportPicture("BMP", "C:\AMIPRO\ICONS\123W.BMP", ".BMP", 0)
StatusBarMsg("")
HourGlass(0)
IgnoreKeyboard(0)
END FUNCTION
```

See also:

GetProfileString$ FileOpen ImportExport AddFrame AddFrameDLG GraphicsScaling ImageProcessing

This function allows you to import text. Choosing this function is equivalent to choosing Action/Import Text.

If you import an Ami Pro object and the object is located on the NewWave Office, the path to the object includes only the backslash.

Syntax

ImportText(FileName, Options, App)

FileName is the name of the file to import. For Ami Pro objects, this should be the object name.

Options is the type of action performed. It is one of the following values:

(1) - Ami Pro file or object
(8) - Import
(16) - Non Ami Pro file or object
(128) - Required

Values may be combined to create the Options parameter.

App is the name of the application. It is taken from the AMIPRO.INI file.

Note that Ami Pro Object is not listed in the AMIPRO.INI file when importing an Ami Pro object.

To show the ImportText dialog box and allow the user to select the parameters: **ImportText**

Return Value

1 (TRUE) if the file and object were imported.

0 (UserCancel) if the user canceled the function.

-2 (GeneralFailure) if the file or object was not imported.

Example

```
FUNCTION Example()
ImportText("\data2" 137 "Ami Pro Object") 'import another Ami pro object called data2
ImportText("c:\datafile.sam" 137 "") 'import an Ami Pro file
ImportText("c:\amipro\macros\count.smm" 137 "") 'import an Ami Pro macro file
END FUNCTION
```

See also:

CreateANew IsNewWave ListObjects NWGetContainerCount NWGetContainerNames NWGetCurrentContainer NWGetCurrentObject$ NWGetObjectCount NWGetObjectNames NWGetParent NWReferenceToFile$ ObjectAttributes OpenObject SaveAsMaster SaveAsObject Share ShowLinks

This function sets the amount of indention for the current paragraph. Choosing this function is equivalent to choosing text/Indention.

Syntax

Indent(AllIndent, FirstIndent, RestIndent, RightIndent)

AllIndent is the amount to indent all the lines.

FirstIndent is the amount to indent the first line.

RestIndent is the amount to indent the rest of the lines after the first line.

RightIndent is the amount to indent from the right side.

All amounts are in twips (1 inch = 1440 twips). Multiply the desired number of inches by 1440 to determine the value in twips.

To display the Indention dialog box: **Indent**

Return Value

1 (TRUE) if the indention was set.

0 (UserCancel) if the user canceled the function.

-6 (InsufficientMemory) if the function failed because of insufficient memory.

Example

```
FUNCTION Example()
' indent all lines 1 inch and the first line another half inch
Indent(1440, 720, 0, 0)
END FUNCTION
```

See also:

Spacing Center Justify NormalText IndentAll IndentFirst IndentRest FastFormat LeftAlign RightAlign

This function allows you to increase the amount of indention for the selected text. Choosing this function is equivalent to clicking the Indent All icon. The amount of indention to be increased is determined by the value set in Text/Indention/Indent All. If the units in Page/Modify Page Layout are set to centimeters, the amount to increase is 1 centimeter. For the other units (inches, picas, or points), the amount to increased is one half of an inch.

Syntax

IndentAll()

Return Value

1 (TRUE) if the indention was increased.

-6 (InsufficientMemory) if the function failed because of insufficient memory.

Example

```
FUNCTION Example()
' indent the selected text to the specified indention
IndentAll()
END FUNCTION
```

See also:

Indent IndentFirst IndentRest

This function allows you to increase the amount of indention for the selected text. Choosing this function is equivalent to clicking the Indent First icon. The amount of indention to be increased is determined by the value set in Text/Indention/Indent First. If the units in Page/Modify Page Layout are set to centimeters, the amount to increase is 1 centimeter. For the other units (inches, picas, or points), the amount to increased is one half of an inch.

Syntax

IndentFirst()

Return Value

1 (TRUE) if the indention was increased.

-6 (InsufficientMemory) if the function failed because of insufficient memory.

Example

```
FUNCTION Example()
' indent the first line to the specified indention
IndentFirst()
END FUNCTION
```

See also:

Indent IndentAll IndentRest

This function allows you to increase the amount of indention for the selected text. Choosing this function is equivalent to clicking the Indent Rest icon. The amount of indention to be increased is determined by the value set in Text/Indention/Indent Rest. If the units in Page/Modify Page Layout are set to centimeters, the amount to increase is 1 centimeter. For the other units (inches, picas, or points), the amount to increased is one half of an inch.

Syntax

IndentRest()

Return Value

1 (TRUE) if the indention was increased.

-6 (InsufficientMemory) if the function failed because of insufficient memory.

Example

```
FUNCTION Example()
' indent the rest of the lines to the specified indention
IndentRest()
END FUNCTION
```

See also:

Indent IndentAll IndentFirst

This function sets the initial capitalization for selected text or for all following text if no text is selected. It acts as a toggle, turning off initial caps if it is currently on or turning on initial caps if it is currently off. Choosing this function is equivalent to choosing Text/Caps/Initial Caps.

Syntax

InitialCaps()

Return Value

1 (TRUE) if the text is set to initial caps.

0 (UserCancel) if the user canceled the function.

-6 (NoMemory) if the function failed because of insufficient memory.

Example

```
FUNCTION Example()
WHILE "" = CurShade$()
    UserControl("Shade the text to modify, then choose Resume.")
WEND
InitialCaps()
END FUNCTION
```

See also:

UpperCase LowerCase SmallCaps

This function inserts a bullet into the text at the cursor position. Choosing this function is equivalent to choosing Edit/Insert/Bullet.

Syntax

InsertBullet(Type)

Type is the type of bullet to insert and can be one of the following:

(1) - Small round bullet
(2) - Large round bullet
(3) - Small square bullet
(4) - Large square bullet
(5) - Large outline square bullet
(6) - Small diamond bullet
(7) - Large diamond bullet
(8) - Small open circle bullet
(9) - Large open circle bullet
(10) - Check mark
(11) - Tack
(12) - Square shadow below bullet
(13) - Square shadow above bullet
(14) - Check box
(15) - Square with X bullet
(16) - Rounded arrowhead top shaded
(17) - Rounded arrowhead bottom shaded

To display the Insert Bullet dialog box to allow the user to select a bullet: **InsertBullet**

Return Value

1 (TRUE) if the bullet is inserted.

-2 (GeneralFailure) is the insertion failed.

Example

```
FUNCTION Example()
' type every kind of bullet
FOR type = 1 to 17
    InsertBullet(type)
    type("[Enter]")
NEXT
END FUNCTION
```

See also:

InsertDate InsertDocInfo ModifyEffects ModifyStyle

This function inserts a cascade menu in an existing pull down menu.

Syntax

InsertCascadeMenu(BarID, Menu, Position, NewMenu)

BarID is the identification number of the menu bar returned from the AddBar function. To use the default Ami Pro menu bar, use 1.

Menu is the name of the pull down menu to insert this cascade menu in. This string must contain any ampersand (&) characters used in the menu name. An ampersand is placed before a character that has an underline.

Position is the position in the pull down menu to insert the cascade menu.

This parameter begins with 0 (to insert as the first position in the pull down menu).

NewMenu is the name of the cascade menu to insert. Placing an ampersand (&) character in front of a character in the string makes that string appear underlined and causes that character to become a shortcut character.

Return Value

1 (TRUE) if the cascade menu was inserted.

0 (UserCancel) if the user canceled the function.

-2 (GeneralFailure) if the cascade menu could not be inserted or if the pull down menu name does not match or exist.

Example

```
FUNCTION Example()
' get the name of this macro file
MacFile = GetRunningMacroFile$()
InsertCascadeMenu(1, "&File", 7, "&Example")
AddCascadeMenuItem(1, "&File", "&Example", "Example1", "{MacFile}!Example2(1)", "")
AddCascadeMenuItem(1, "&File", "&Example", "Example2", "{MacFile}!Example2(2)", "")
InsertCascadeMenuItem(1, "&File", "&Example", 1, "Example1.5", "{MacFile}!Example2(3)", "")
END FUNCTION

FUNCTION Example2(Msg)
Message(Msg)
END FUNCTION
```

See also:

AddBar AddCascadeMenu AddCascadeMenuItem AddMenu AddMenuItem AddMenuItemDDE ChangeCascadeAction DeleteMenu DeleteMenuItem InsertMenu InsertMenuItem InsertCascadeMenuItem

This function inserts a cascade menu item on an existing cascade menu.

Syntax

InsertCascadeMenuItem(BarID, Menu, Position, Item, MacroName[!Function[(parm1[, parm2...])]], Help)

BarID is the identification number of the menu bar returned from the AddBar function. To use the default Ami Pro menu bar, use 1.

Menu is the name of the cascade menu this item should be inserted in. This must match exactly the name of the menu item you want to use, including any ampersand (&) characters in the name of the menu. An ampersand is placed before a character that has an underline.

Position is where to locate the item in the cascade menu.

This parameter begins with 0 (to insert as the first position in the cascade menu).

Item is the text to insert. This is the text that the user sees on the cascade menu bar. Placing an ampersand (&) in front of a character causes that character to appear underlined and makes that character a shortcut key. If the menu item has a shortcut key, you must press **TAB**, type a ^, and then type the appropriate letter as part of the item name.

MacroName is the name of the macro to run if this menu item is selected. This parameter may contain the macro filename, the function within that file to call, and any parameters that the function may require. At a minimum, this parameter must contain the macro filename.

Help is the one-line Help text that appears in the title bar of Ami Pro when this menu item is highlighted. This parameter is not optional for this function.

Return Value

1 (TRUE) if the cascade menu item was inserted.

0 (UserCancel) if the user canceled the function.

-2 (GeneralFailure) if the cascade menu item could not be inserted, or if the pull down menu or the cascade menu names did not match the parameters or if either did not exist.

Example

```
FUNCTION Example()
' get the name of this macro file
MacFile = GetRunningMacroFile$()
InsertCascadeMenu(1, "&File", 7, "&Example")
AddCascadeMenuItem(1, "&File", "&Example", "Example1", "{MacFile}!Example2(1)", "")
AddCascadeMenuItem(1, "&File", "&Example", "Example2", "{MacFile}!Example2(2)", "")
InsertCascadeMenuItem(1, "&File", "&Example", 1, "Example1.5", "{MacFile}!Example2(3)", "")
END FUNCTION

FUNCTION Example2(Msg)
Message(Msg)
END FUNCTION
```

See also:

AddBar AddCascadeMenu AddCascadeMenuItem AddMenu AddMenuItem AddMenuItemDDE ChangeCascadeAction DeleteMenu DeleteMenuItem InsertMenu InsertMenuItem InsertCascadeMenu

This function inserts table columns or rows. Choosing this function is equivalent to choosing Table/Insert Column/Row.

Syntax

InsertColumnRow(Number, Where, Which)

Number is the number of columns or rows to insert.

Where is a number representing where to insert the columns or rows and can be one of the following:

InsAfter (0) - Inserts after the current column/row

InsBefore (1) - Inserts before the current column/row

Which is a number determining whether to insert columns or rows and can be one of the following:

Column (1) - Inserts a column

Row (0) - Inserts a row

To display the Insert Column/Row dialog box to allow the user to decide the number and type of columns/rows to insert: **InsertColumnRow**

Return Value

1 (TRUE) if the column or row was inserted.

0 (UserCancel) if the user canceled the function.

-2 (GeneralFailure) if the row or column was not inserted.

Example

```
FUNCTION Example()
' Insert 1 column after the current column
InsertColumnRow(1, 0, 1)
END FUNCTION
```

See also:

DeleteColumnRow DeleteEntireTable SelectColumn SelectEntireTable SelectRow SizeColumnRow

This function inserts a date into the document. Choosing this function is equivalent to choosing Edit/Insert/Date/Time.

Syntax

InsertDate(Type, Style)

Type is the type of date to insert and can be one of the following:

Today (2) - Inserts the current system date and always displays this date

SysDate (3) - Inserts the current system date and updates this date each time you open the document

LastRev (5) - Inserts the date the document was last revised

CreateDate (7) - Inserts the date the document was created

Style is the style of date to insert and can be one of the following:

(0) NMDY - 9/11/91

(1) MDY - September 11, 1991

(2) UMDY - SEPTEMBER 11, 1991

(3) DMY - 11 September 1991

(4) UDMY - 11 SEPTEMBER 1991

(5) DMDY - Friday, September 11, 1991

(6) UDMDY - FRIDAY, SEPTEMBER 11, 1991

(7) MD - September 11

(8) UMD - SEPTEMBER 11

(9) DMD - Friday September 11

(10) UDMD - FRIDAY SEPTEMBER 11

(11) NMD - 9/11

(12) NMDLY - 9/11/91

(13) DPM - 11. September

(14) UDPM - 11. SEPTEMBER

(15) DPMY - 11. September 1991

(16) UDPMY - 11. SEPTEMBER 1991

(17) YMD - 1991 September 11

(18) UYMD - 1991 SEPTEMBER 11

(19) MY - September, 1991

(20) UMY - SEPTEMBER, 1991

To display the Insert Date/Time dialog box and allow the user to select the date type and style: **InsertVariable**

Return Value

1 (TRUE) if the date was inserted.

-2 (GeneralFailure) if no date was inserted.

-6 (NoMemory) if the function failed because of insufficient memory.

Example

```
FUNCTION Example()
InsertDate(2, DMDY)
END FUNCTION
```

See also:

InsertVariable InsertDocInfo Now FormatDate$ FormatTime$ GetTime InsertBullet

This function displays the Insert Doc Info dialog box. Choosing this function is equivalent to choosing Edit/Insert/Doc Info Field. To insert the Doc Info field without the dialog box, see the InsertDocInfoField function.

Syntax

InsertDocInfo

Return Value

This function does not return a value.

Example

```
FUNCTION Example()
InsertDocInfo
END FUNCTION
```

See also:

DocInfo RenameDocInfoField GetDocInfo$ InsertDocInfoField

This function inserts the Doc Info field without the Insert Doc Info field dialog box. Choosing this function is equivalent to choosing Edit/Insert/Doc Info Field/Insert. To display the Insert Doc Info field dialog box, see the InsertDocInfo function.

Syntax

InsertDocInfoField(Style, Which)

Style is the style of date to insert and can be one of the following:

(0) NMDY - 9/11/91
(1) MDY - September 11, 1991
(2) UMDY - SEPTEMBER 11, 1991
(3) DMY - 11 September 1991
(4) UDMY - 11 SEPTEMBER 1991
(5) DMDY - Friday, September 11, 1991
(6) UDMDY - FRIDAY, SEPTEMBER 11, 1991
(7) MD - September 11
(8) UMD - SEPTEMBER 11
(9) DMD - Friday September 11
(10) UDMD - FRIDAY SEPTEMBER 11
(11) NMD - 9/11
(12) NMDY - 9/11/91
(13) DPM - 11. September
(14) UDPM - 11. SEPTEMBER
(15) DPMY - 11. September 1991
(16) UDPMD - 11. SEPTEMBER 1991
(17) YMD - 1991 September 11
(18) UYMD - 1991 SEPTEMBER 11
(19) MY - September, 1991
(20) UMY - SEPTEMBER, 1991

Which is the desired document info field and can be one of the following:

DDFilename (1) - File name
DDPath (2) - Path for this document
DDStylesheet (3) - Style sheet for this document
DDRevisions (6) - Number of document revisions
DDDescription (7) - Document description
DDUser1 (8) - User defined field 1
DDUser2 (9) - User defined field 2
DDUser3 (10) - User defined field 3
DDUser4 (11) - User defined field 4
DDUser5 (12) - User defined field 5
DDUser6 (13) - User defined field 6
DDUser7 (14) - User defined field 7
DDUser8 (15) - User defined field 8

Return Value

1 (TRUE) if the field was inserted.
0 (FALSE) if the field was not inserted.

-6 (InsufficientMemory) if not enough memory.

Example

```
FUNCTION Example()
InsertDocInfoField(0, 6)
END FUNCTION
```

See also:

DocInfo InsertDocInfo RenameDocInfoField GetDocInfo$

This function inserts a page break at the current insertion point and displays the Modify Page Layout dialog box. Choosing this function is equivalent to choosing Page/Insert Page Layout/Insert. You must be in Layout Mode to use this function.

Syntax

InsertLayout()

Return Value

This function returns 1.

Example

```
FUNCTION Example()
InsertLayout()
END FUNCTION
```

See also:

CreateStyle ModifyLayout ModifyStyle HeaderFooter RemoveLayout

This function inserts a new pull down menu on the specified menu bar.

Syntax

InsertMenu(BarID, Position, NewMenu)

BarID is the identification number of the menu bar returned from the AddBar function. To use the default Ami Pro menu bar, use 1.

Position is where to locate the menu.

This parameter begins with 0 (to insert as the first position on the menu bar).

NewMenu is the name of the new menu name to insert. Placing an ampersand (&) in front of a character causes that character to appear underlined and makes that character a shortcut key.

Return Value

1 (TRUE) if the menu was inserted.

0 (UserCancel) if the user canceled the function.

-2 (GeneralFailure) if the menu could not be inserted.

Example

```
FUNCTION Example()
InsertMenu(1, 1, "&NewMenu")
END FUNCTION
```

See also:

AddBar AddCascadeMenu AddCascadeMenuItem AddMenu AddMenuItem AddMenuItemDDE DeleteMenu DeleteMenuItem InsertMenuItem InsertCascadeMenu InsertCascadeMenuItem RenameMenuItem

This function inserts a new menu item on the specified pull down menu.

Syntax

InsertMenuItem(BarID, Menu, Position, Item, MacroName[!Function[(parm1[, parm2...])]][, Help])

BarID is the identification number of the menu bar returned from the AddBar function. To use the default Ami Pro menu bar, use 1.

Menu is the name of the existing menu to insert this item in. The exact name of the menu must be used, including any ampersand (&) characters. An ampersand is placed before a character that has an underline.

Position is where to locate the menu item in the pull down menu.

This parameter begins with 0 (to insert as the first position in the pull down menu).

Item is the text to appear for this menu item. Placing an ampersand (&) in front of a character causes that character to appear underlined and makes that character a shortcut key. If the menu item has a shortcut key, you must press **TAB**, type a ^, and then type the appropriate letter as part of the item name.

MacroName is the name of the macro to run if this menu item is selected. This parameter may contain the macro filename, the function within that file to call, and any parameters that function may require. At a minimum, this parameter must contain the macro filename.

Help is the one-line Help text that appears in the title bar of Ami Pro when this menu item is highlighted.

Return Value

1 (TRUE) if the menu item was inserted.

0 (UserCancel) if the user canceled the function.

-2 (GeneralFailure) if the menu item could not be inserted.

Example

```
FUNCTION Example()
' get the name of this macro file
MacFile = GetRunningMacroFile$()
State = 0
DeleteMenuItem(1, "&Text", "B&old+Italic+Underline")
InsertMenuItem(1, "&Text", 10, "B&old+Italic+Underline", "{MacFile}!Example2({State})", "Bold and
Italicize and Underline Text.")
END FUNCTION

FUNCTION Example2(State)
MacFile = GetRunningMacroFile$()
State = Right$((State - 1), 1)
Bold(State)
Underline(State)
Italic(State)
ChangeMenuAction(1, "&Text", "B&old+Italic+Underline", "{MacFile}!Example2({State})", "Bold and
Italicize and Underline Text.")
CheckMenuItem(1, "&Text", "B&old+Italic+Underline", State)
END FUNCTION
```

See also:

AddBar AddCascadeMenu AddCascadeMenuItem AddMenu AddMenuItem AddMenuItemDDE DeleteMenu DeleteMenuItem InsertMenu InsertCascadeMenu InsertCascadeMenuItem RenameMenuItem

This function displays the Insert Merge Field dialog box. The user must select the merge field to insert. Choosing this function is equivalent to choosing Edit/Insert/Merge Field.

To insert the field automatically, see the FieldAdd function.

Syntax

InsertMerge

Return Value

This function returns 0.

Example

```
FUNCTION Example()
InsertMerge
END FUNCTION
```

See also:

FieldAdd FieldRemove Merge MergeAction MergeMacro MergeToFile

This function inserts a merge field into a standard document. It is equivalent to choosing Edit/Insert from the menu. This function inserts the field without checking to make sure it is a valid field name.

Make sure field names that are inserted match with the data files that will be used to merge.

In Ami Pro release 3.0, the FieldAdd function has replaced this function.

Syntax

InsertMergeField(Field)

Field is the name of the merge field that must be inserted.

To display the Insert Merge Field dialog box and let the user select the field name to insert: **InsertMerge**.

Return Value

1 (TRUE) if the field name was inserted.

-2 (GeneralFailure) if the field name was not inserted.

-6 (NoMemory) if the function failed because of insufficient memory.

Example

```
FUNCTION Example()
InsertMergeField(Query$("What is the name of the merge field to insert?"))
END FUNCTION
```

See also:

InsertDate InsertDocInfo InsertMerge InsertVariable

This function inserts an OLE object into a frame. Choosing this function is equivalent to choosing Edit/Insert/New Object.

Syntax

InsertNewObject(Description)

Description is the class description of the object to be inserted. The name may be extracted from the list of names displayed in the Object type list box when viewing the Insert New Object dialog box. The list is taken from the embedding section of the WIN.INI file. In this file, the true class name, an equals sign, and the class description name are shown.

Return Value

1 (TRUE) if the object was successfully inserted.

-2 (GeneralFailure) if the insertion failed.

Example

```
FUNCTION Example()
AddFrameDLG
InsertNewObject("Package")
END FUNCTION
```

See also:

AddFrame AddFrameDLG

This function inserts a note with the specified text at the insertion point. The note is automatically inserted, and the note window is closed using this function. Choosing this function is equivalent to choosing Edit/Insert/Note, typing the note, and closing the note. A macro must be edited to insert this non-recordable function.

Syntax

InsertNote(Text)

Text is the text that should go inside the note.

To allow the user to determine note options or to type the note text: **Notes**

Return Value

This function returns:

1 (TRUE) if the note was inserted.

-2 (GeneralFailure) if the note was not inserted.

Example

```
FUNCTION Example()
InsertNote("Ami Pro is the easiest word processor to learn.")
END FUNCTION
```

See also:

Notes UserSetup ViewPreferences

This function displays the Insert Date/Time dialog box. Choosing this function is equivalent to choosing Edit/Insert/Date/Time. This function does not insert variables automatically. Variables can be directly inserted using the InsertDate, InsertDocInfoField, and FieldAdd functions.

Syntax

InsertVariable

Return Value

This function returns 0.

Example

```
FUNCTION Example()
InsertVariable
END FUNCTION
```

See also:

InsertDate InsertDocInfo FieldAdd

This function searches a text string for a pattern.

Syntax

Instr(Offset, Text, Pattern)

Offset is the position in the string to begin searching.

Text is the string to search.

Pattern is the pattern to search for.

Return Value

The value of the offset of the beginning of the pattern, if it is found.

0 (FALSE) if the offset is not found.

Example

```
FUNCTION Example()
Name = UCASE$(Query$("What is your name?"))
FOR I = 65 to 90
    Ltr = CHR$(I)
    IF 0 != Instr(0, Name, Ltr)
        TYPE("The letter {Ltr} is in your name.[Enter]")
    ENDIF
NEXT
END FUNCTION
```

See also:

CHR$ strcat$ strchr strfield$ Left$ Right$ MID$

This function determines whether or not a frame is currently selected.

Syntax

IsFrameSelected()

Return Value

1 (TRUE) if the frame is currently selected.

0 (FALSE) if the frame is not currently selected.

Example

```
FUNCTION Example()
LayoutMode()
IF not IsFrameSelected()
    Pos = CursorPosition$()
    x = strfield$(Pos, 1, ",")
    y = strfield$(Pos, 2, ",")
    AddFrame(x, y, (x + 1440), (y - 1440))
ENDIF
DrawingMode()
END FUNCTION
```

See also:

AddFrame AddFrameDLG BringFrameToFront FrameLayout ManualFrame SelectFrameByName

This function checks if the current running copy of Ami Pro is running under NewWave.

Syntax

IsNewWave()

Return Value

1 (TRUE) if Ami Pro for NewWave is running.

0 (FALSE) if Ami Pro for NewWave is not running.

Example

```
FUNCTION Example()
IF IsNewWave()
' NewWave statements
ENDIF
END FUNCTION
```

See also:

CreateANew ImportText ListObjects NWGetContainerCount NWGetContainerNames NWGetCurrentContainer NWGetCurrentObject$ NWGetObjectCount NWGetObjectNames NWGetParent NWReferenceToFile$ ObjectAttributes OpenObject SaveAsMaster SaveAsObject Share ShowLinks

This function determines whether a string is suitable for numeric operations.

Syntax

IsNumeric(Text)

Text is the string which is to be evaluated.

Return Value

1 (TRUE) if the passed string is numeric.

0 (FALSE) if the passed string is not numeric.

Example

```
FUNCTION Example()
again:
Num1 = Query$("What is the first number?")
Num2 = Query$("What is the second number?")
IF (not IsNumeric(Num1)) OR (not IsNumeric(Num2))
    GoTo again
ENDIF
sum = Num1 + Num2
Message("The sum of {Num1} and {Num2} is {sum}.")
END FUNCTION
```

See also:

FormatNum$ Mod Round

This function sets the italic attribute for selected text or for all following text if no text is selected. It acts as a toggle, turning off the attribute if it is currently on or turning on the attribute if it is currently off. Choosing this function is equivalent to choosing Text/Italic.

Syntax

Italic()

Return Value

0 if the italic attribute is toggled on and there are no attributes assigned to the text.

4 if the italic attribute is toggled on and the bold attribute is already assigned.

8 if the italic attribute is toggled off.

16 if the italic attribute is toggled on and the underline attribute is already assigned.

32 if the italic attribute is toggled on and the word underline attribute is already assigned.

-2 (GeneralFailure) if the attribute was not changed.

The returns values may be added together to identify the attributes that were previously assigned.

Example

```
FUNCTION Example()
text = Query$("Enter some text:")
New("_default.sty", 0, 0)     'open a new file
TYPE("{text}")
TYPE("[enter]")
TYPE("[ctrlhome][ctrlshiftend]")
Copy()  'copy text to clipboard
FOR I = 1 to 10
    Paste()     'paste text 10 times
NEXT
Message("The text will be shaded, bolded, italicized, underlined, and centered.")
TYPE("[ctrlhome][ctrlshiftend]")
Bold()
Italic()
Underline()
Center()
TYPE("[ctrlhome]")
END FUNCTION
```

See also:

Bold NormalText Underline WordUnderline

This function acts as a toggle to turn justification on or off for a paragraph. Choosing this function is equivalent to choosing Text/Alignment/Justify.

Syntax

Justify()

Return Value

1 (TRUE) if the text was justified or if justification was removed.

-2 (GeneralFailure) if the alignment was not changed.

Example

```
FUNCTION Example()
String = "This is a line of Text. "
FOR i = 1 to 10
    Paste()
NEXT
Justify()
END FUNCTION
```

See also:

Center LeftAlign NormalText RightAlign

This function displays the online Help for Ami Pro. Choosing this function is equivalent to choosing Help/Keyboard, but it does not select a Help topic automatically. Because Help displays in a window other than the regular Ami Pro window, further macro functions which cause a repainting of the Ami Pro window cause the Help window to be replaced by the Ami Pro window.

If this function is used, it should be the last function used in the macro.

Syntax

KeyboardHelp()

Return Value

1 (TRUE) if the Help window was displayed.

-2 (GeneralFailure) if the Help window could not be displayed for some other reason.

-6 (NoMemory) if the function failed because of insufficient memory.

Example

```
FUNCTION Example()
KeyboardHelp()
END FUNCTION
```

See also:

About Help EnhancementProducts HowDoIHelp MacroHelp UpgradeHelp UsingHelp

This function sets a macro function to be called if the user hits a key while the current macro is running. The called macro function is passed the virtual key. If the interrupt routine needs to pass information to the main function, it must pass it through a global variable and the main function must check that global variable.

Syntax

KeyInterrupt(Function)

Function is the function that runs when a key is pressed. This parameter may contain the macro file name or the function within that file to call.

Return Value

This function returns the previously set KeyInterrupt function.

Example

```
FUNCTION Example()
KeyInterrupt("KeyInt")
TYPE("Touch any key")
FOR i = 1 to 5   ' Kill some time.
    FOR j = 1 to 100
    NEXT
    TYPE("{i}.")
NEXT
END FUNCTION

FUNCTION KeyInt(key)
Message("KeyInt {key}")
END FUNCTION
```

See Also:

MouseInterrupt DlgKeyInterrupt Hourglass Pause

This function changes the current Multiple Document Interface (MDI) document in Ami Pro from Draft Mode or Outline Mode to Layout Mode. Choosing this function is equivalent to choosing View/Layout Mode.

Syntax

LayoutMode()

Return Value

1 (TRUE) if the view mode was changed.

0 (NoAction) if no action was taken because Ami Pro is already in the mode selected.

Example

```
FUNCTION Example()
Mode = GetMode()'What mode is the screen in?
IF Mode != 1
    LayoutMode()'If not Layout Mode, make it so.
ENDIF
END FUNCTION
```

See also:

DraftMode OutlineMode EnlargedView FacingView FullPageView GetMode GetViewLevel StandardView CustomView

This function converts uppercase letters in the source string to lowercase and returns the resulting string. It does not change punctuation or numbers.

Syntax

LCASE$(Text)

Text is the string that is converted to lowercase.

Return Value

The string with all lowercase letters.

Example

```
FUNCTION Example()
Again:
Number = Query$("How much was the sale?")
Where = Query$("Where was the sale made (E)ngland, or (A)merica?")
Where = lcase$(Left$(Where, 1))
IF Where = "e"
    Prefix = ""
    Suffix = "£"
ELSEIF Where = "a"
    Prefix = "$"
    Suffix = ""
ELSE
    Message("Please choose ""E"" or ""A""."")
    GoTo again
ENDIF
NewNumber = FormatNum$(Prefix, Suffix, 2, Number)
Message("The sale was for {NewNumber}.")
END FUNCTION
```

See also:

ASC CHR$ strcat$ strchr strfield$ UCASE$ MID$ LEN FormatNum$ Instr

This function applies dots for leader characters in the current table cell or selected table cells. Choosing this function is equivalent to choosing Table/Leaders and selecting dots for the leader type.

Syntax

LeaderDots()

Return Value

1 (TRUE) if the leader dots were applied.

0 (NoAction) if no action was taken.

-6 (NoMemory) if the function failed because of insufficient memory.

Example

```
FUNCTION Example()
' create a table
Tables(1, TRUE, 5, 5)
LeaderDots()
TYPE("[Right]")
LeaderHyphs()
TYPE("[Right]")
LeaderLines()
IF Decide("Press OK to remove leaders.")
    TYPE("[Left][Left]")
    FOR i = 1 to 3
        LeaderNone()
        TYPE("[Right]")
    NEXT
ENDIF
END FUNCTION
```

See also:

TableLines LeaderHyphs LeaderLines LeaderNone Tables TableLayout

This function applies hyphens for leader characters in the current table cell or selected table cells. Choosing this function is equivalent to choosing Table/Leaders and selecting hyphens for the leader type.

Syntax

LeaderHyphs()

Return Value

1 (TRUE) if the leader dots were applied.

0 (NoAction) if no action was taken.

-6 (NoMemory) if the function failed because of insufficient memory.

Example

```
FUNCTION Example()
' create a table
Tables(1, TRUE, 5, 5)
LeaderDots()
TYPE("[Right]")
LeaderHyphs()
TYPE("[Right]")
LeaderLines()
IF Decide("Press OK to remove leaders.")
    TYPE("[Left][Left]")
    FOR i = 1 to 3
        LeaderNone()
        TYPE("[Right]")
    NEXT
ENDIF
END FUNCTION
```

See also:

TableLines LeaderDots LeaderLines LeaderNone Tables TableLayout

This function applies underlines for leader characters in the current table cell or selected table cells. Choosing this function is equivalent to choosing Table/Leaders and selecting underline for the leader type.

Syntax

LeaderLines()

Return Value

1 (TRUE) if the leader dots were applied.

0 (NoAction) if no action was taken.

-6 (NoMemory) if the function failed because of insufficient memory.

Example

```
FUNCTION Example()
' create a table
Tables(1, TRUE, 5, 5)
LeaderDots()
TYPE("[Right]")
LeaderHyphs()
TYPE("[Right]")
LeaderLines()
IF Decide("Press OK to remove leaders.")
    TYPE("[Left][Left]")
    FOR i = 1 to 3
        LeaderNone()
        TYPE("[Right]")
    NEXT
ENDIF
END FUNCTION
```

See also:

TableLines LeaderDots LeaderHyphs LeaderNone Tables TableLayout

This function turns off the leader characters in the current table cell or selected table cells. Choosing this function is equivalent to choosing Table/Leaders/None.

Syntax

LeaderNone()

Return Value

1 (TRUE) if the leader dots were applied.

0 (NoAction) if no action was taken.

-6 (NoMemory) if the function failed because of insufficient memory.

Example

```
FUNCTION Example()
' create a table
Tables(1, TRUE, 5, 5)
LeaderDots()
TYPE("[Right]")
LeaderHyphs()
TYPE("[Right]")
LeaderLines()
IF Decide("Press OK to remove leaders.")
    TYPE("[Left][Left]")
    FOR i = 1 to 3
        LeaderNone()
        TYPE("[Right]")
    NEXT
ENDIF
END FUNCTION
```

See also:

TableLines LeaderDots LeaderHyphs LeaderLines Tables TableLayout

This function returns the specified number of characters from the left of the specified string.

Syntax

Left$(Text, Length)

Text is the string to be parsed.

Length is the number of characters from the left end of the string to be parsed.

Return Value

The specified number of characters from the left end of the string.

0 (UserCancel) if the user canceled the function.

-2 (GeneralFailure) if the string could not be parsed.

Example

```
FUNCTION Example()
Message(Left$(Query$("What is your name?"), 5))
END FUNCTION
```

See also:

Right$ Instr MID$

This function acts as a toggle to turn on/off left alignment for a paragraph. Choosing this function is equivalent to choosing Text/Alignment/Left.

Syntax

LeftAlign()

Return Value

1 (TRUE) if the text was left aligned, or if alignment was removed.

-2 (GeneralFailure) if the alignment was not changed.

Example

```
FUNCTION Example()
String = "This is a line of Text. "
FOR i = 1 to 10
    Paste()
NEXT
LeftAlign()
END FUNCTION
```

See also:

Center Justify NormalText RightAlign

This function scrolls the document to the left edge of the page without moving the insertion point. Choosing this function is equivalent to dragging the elevator on the horizontal scroll bar to the left using the mouse.

Syntax

LeftEdge()

Return Value

This function returns 0.

Example

```
FUNCTION Example()
LeftEdge()
END FUNCTION
```

See also:

CharLeft CharRight EndOfFile LineDown LineUp RightEdge ScreenDown ScreenLeft ScreenRight ScreenUp TopOfFile

This function determines the number of characters in the specified string.

Syntax

LEN(Text)

Text is the string whose length is determined.

Return Value

The length of the string.

0 (FALSE) if the string is empty.

Example

```
FUNCTION Example()
DEFSTR Char;
' get the name of the open file
OpenFile = GetOpenFileName$()
' get the length of the name
I = len(OpenFile)
' take off the path
WHILE "\" != Assign(&Char, MID$(OpenFile, I, 1))
    I = I - 1
WEND
FileName = Right$(OpenFile, (len(OpenFile) - I))
Message("The current file is {FileName}.")
END FUNCTION
```

See also:

strcat$ LCASE$ UCASE$ strfield$ MID$ strchr FormatNum$ Right$ Left$

This function scrolls the document down one line without moving the insertion point. Choosing this function is equivalent to clicking once on the vertical scroll bar up arrow.

Syntax

LineDown()

Return Value

This function returns 0.

Example

```
FUNCTION Example()
LineDown()
END FUNCTION
```

See also:

CharLeft CharRight EndOfFile LeftEdge LineUp RightEdge ScreenDown ScreenLeft ScreenRight ScreenUp TopOfFile

This function places line numbers in the left margin of the document. Choosing this function is equivalent to choosing Page/Line Numbering.

Modified in 3.0 There is a new value, 128, for the Which parameter in Ami Pro release 3.0.

Syntax

LineNumber(Which, Style, Custom)

Which determines the type of line to number and the frequency of the line numbering. It is a combination of the following values:

Type of line to number:

Off (0) - Do not number lines

NumberLines (1) - Number any type of line

NumberTextLines (128) - Number text lines

Use one of the following in combination with one of the above:

NumberEvery (2) - Number all lines (of the type selected above)

NumberEveryOther (4) - Number every other line (of the type selected above)

NumberEveryFifth (8) - Number every fifth line (of the type selected above)

NumberEveryCustom (32) - Number every nth line (of the type selected above), where n is a custom number

Additional Optional Value:

ResetEachPage (16) - Reset the numbering sequence on every page of the document

Style is the paragraph style to use in determining the spacing for the line numbering. The Style parameter must be used, even if line numbering is being turned off.

Custom is the number indicating the frequency of the line numbering.

To display the Line Numbering dialog box to allow the user to choose the line numbering options:
LineNumber

Return Value

1 (TRUE) if the numbering sequence was set.

0 (UserCancel) if the user canceled the function, or if no action was taken.

Example

```
FUNCTION Example()
LineNumber(3, "Body Text", 0)
END FUNCTION
```

See also:

PageNumber

This function scrolls the document up one line without moving the insertion point. Choosing this function is equivalent to clicking once on the vertical scroll bar down arrow.

Syntax

LineUp()

Return Value

This function returns 0.

Example

```
FUNCTION Example()
LineUp()
END FUNCTION
```

See also:

CharLeft CharRight EndOfFile LeftEdge LineDown RightEdge ScreenDown ScreenLeft ScreenRight ScreenUp TopOfFile

This function either opens or prints an object. Choosing this function is equivalent to choosing Objects/List Objects.

Syntax

ListObjects (Object, Which)

Object is the full path to the object.

Which is one of the following values:

0 - Open the object

1 - Print the object

To display the ListObjects dialog box and allow the user to select the parameters: **ListObjects**

Return Value

0 (UserCancel) if the user canceled the function.

Example

```
FUNCTION Example()
TYPE ("[CtrlHome]") 'go to top of doc
GoToCmd(4 2 1) 'select the frame
ListObjects("memo1" 0) 'opens the object "memo1"
ListObjects("memo1" 1) 'prints the object "memo1"
END FUNCTION
```

See also:

CreateANew ImportText IsNewWave NWGetContainerCount NWGetContainerNames NWGetCurrentContainer NWGetCurrentObject$ NWGetObjectCount NWGetObjectNames NWGetParent NWReferenceToFile$ ObjectAttributes OpenObject SaveAsMaster SaveAsObject Share ShowLinks

This function sets initial display preferences for using Ami Pro. Choosing this function is equivalent to choosing Tools/User Setup/Load.

Modified in 3.0 There are three new parameters, 256, 1024, and 2048, in Ami Pro release 3.0. There is one parameter, 64, available in Ami Pro release 2.0 only.

Syntax

LoadOptions(InitialView, Options, StyleSheet)

InitialView determines whether Standard or C ustom view initially displays. The InitialView parameter should be set to one of the following options.

StartCustom (2) - Start in Custom View

StartStandard (3) - Start in Standard View

Options is a flag variable determining other display defaults.

If any of the following options parameters are set, they take effect when the program starts. If the options are not set, they do not take effect. Options can be one or more of the following:

StartLayout (1) - Start the program in Layout Mode. If this value or StartOutline is not set, the program starts in Draft Mode. This value may not be combined with StartOutline.

StylesBox (2) - Display the styles box when the program starts; otherwise, do not display the styles box.

StartMax (8) - Maximize the Ami Pro window when the program is loaded; otherwise, use the previously set window size.

ShowLogo (64) - Display the Ami Pro logo when the program starts; otherwise do not display the logo.

StartOutline (256) - Start the program in Outline Mode. This value may not be combined with StartLayout.

ShowDesc (1024) - List the style sheets by description name (rather than style sheet file name) in the Load Defaults dialog box.

StartClean (2048) - Start the program in Clean Screen mode.

To set multiple options, add the option values together before passing them to the function.

StyleSheet is the default style sheet to use for new documents.

To show the Load Defaults dialog box and allow the user to select initial display preferences:

LoadDefaults

Return Value

1 (TRUE) if the load options were successfully set.

0 (UserCancel) if the user canceled the function.

-2 (GeneralFailure) if the options could not be set.

Example

```
FUNCTION Example()
LoadOptions(StartStandard, StartLayout, "_MACRO.STY")
END FUNCTION
```

See also:

ViewPreferences SetDefOptions UserSetup

This function sets lowercase for selected text or for all following text if no text is selected. It acts as a toggle, turning off lowercase if it is on or turning on lowercase if it is off. Choosing this function is equivalent to choosing Text/Caps/Lower Case.

Syntax

LowerCase()

Return Value

1 (TRUE) if the attribute was changed.

-6 (NoMemory) if the function failed because of insufficient memory.

Example

```
FUNCTION Example()
LowerCase()
END FUNCTION
```

See also:

UpperCase InitialCaps SmallCaps

This function displays the Edit Macro dialog box. Choosing this function is equivalent to choosing Tools/Macros/Edit.

A macro must be edited to insert this non-recordable function.

Syntax

MacroEdit()

Return Value

This function returns 1.

Example

```
FUNCTION Example()
MacroEdit()
END FUNCTION
```

See also:

AssignMacroToFile MacroPlay MacroOptions ChangeShortcutKey OnKey

This function displays the online Ami Pro Macro Documentation Help. Choosing this function is equivalent to choosing Help/Macro Doc.

Syntax

MacroHelp()

Return Value

1 (TRUE) if the macro Help window displayed.

-2 (GeneralFailure) if the macro Help window could not be displayed for some reason other than insufficient memory.

-6 (NoMemory) if the function failed because of insufficient memory.

Example

```
FUNCTION Example()
MacroHelp()
END FUNCTION
```

See also:

About EnhancementProducts Help HowDoIHelp KeyboardHelp UpgradeHelp UsingHelp

This function displays the Quick Record Macro Options dialog box. Choosing this function is equivalent to choosing Tools/Macros/Record/Options. The dialog box allows you to select shortcut keys for Quick Macro Record and Quick Macro Play. A macro must be edited to insert this non-recordable function.

Syntax

MacroOptions()

Return Value

This function returns 1.

Example

```
FUNCTION Example()
MacroOptions()
END FUNCTION
```

See also:

AssignMacroToFile MacroPlay MacroEdit ChangeShortcutKey OnKey

This function displays the Play Macro dialog box. Choosing this function is equivalent to choosing Tools/Macros/Playback. A macro must be edited to insert this non-recordable function.

Syntax

MacroPlay()

Return Value

This function returns 1.

Example

```
FUNCTION Example()
MacroPlay()
END FUNCTION
```

See also:

AssignMacroToFile MacroOptions MacroEdit ChangeShortcutKey OnKey

This function displays the frame arrow and allows the user to manually create a frame. Choosing this function is equivalent to choosing Frame/Create Frame/Manual.

Syntax

ManualFrame()

Return Value

This function returns 1.

Example

```
FUNCTION Example()
ManualFrame()
END FUNCTION
```

See also:

AddFrame AddFrameDlg FrameLayout FrameModBorders FrameModFinish FrameModInit FrameModLines FrameModType IsFrameSelected SelectFrameByName SetFrameDefaults

This function allows the user to add a bookmark, remove a bookmark, or go to a specific bookmark. Choosing this function is equivalent to choosing Edit/Bookmarks.

Syntax

MarkBookMark(Name, Which)

Name is the name of the new bookmark.

Which is the desired bookmark function and can be one of the following:

AddBookmark (4003) - Add a new bookmark with the name provided

DeleteBookmark (4004) - Delete the named bookmark

FindBookmark (4002) - Go to the named bookmark

To display the Bookmarks dialog box to allow the user to select the bookmark function and name: **MarkBookMark**

Return Value

1 (TRUE) if the bookmark function was successfully completed.

-2 (GeneralFailure) if the bookmark function could not be completed.

Example

```
FUNCTION Example()
MarkBookMark("Example", AddBookmark)
TYPE("[CtrlEND]")
PageNo = GetPageNo()
MarkBookMark("Example", FindBookmark)
Text = GetMarkText$()
MarkBookMark("Example", DeleteBookmark)
IF Text != ""
    Message("Your bookmark was named ""{Text}"", and there are {PageNo} pages in this document.")
ENDIF
END FUNCTION
```

See also:

GoToCmd GoToShade GetMarkText$

This function displays the Mark Index Entry dialog box to allow you to mark selected text for inclusion in the index. Choosing this function is equivalent to choosing Edit/Mark Text/Index Entry.

To automatically mark a word for the index, use the FieldAdd function. To automatically remove an index entry, use the FieldRemove function.

Syntax

MarkIndexWord

Return Value

This function returns 0.

Example

```
FUNCTION Example()
MarkIndexWord
END FUNCTION
```

See also:

Generate FieldAdd FieldRemove SetIndexFile

This function displays the TOC Entry dialog box to allow you to mark selected text for a table of contents entry. Choosing this function is equivalent to choosing Edit/Mark Text/TOC Entry.

To automatically add a TOC entry, use the FieldAdd function. To automatically remove a TOC entry, use the FieldRemove function.

Syntax

MarkTOCEntry

Return Value

This function returns 0.

Example

```
FUNCTION Example()
MarkTOCEntry
END FUNCTION
```

See also:

Generate FieldAdd FieldRemove SetTOCFile TOCOptions

This function displays the Master Document dialog box and allows you to select files to add or remove from the master document. Choosing this function is equivalent to choosing File/Master Document.

This function does not automatically add or remove files from the master document. Please refer to the SetMasterFiles function.

Syntax

MasterDoc

Return Value

This function returns 0.

Example

```
FUNCTION Example()
MasterDoc
END FUNCTION
```

See also:

MasterDocOpts GetMasterFilesCount GetMasterFiles SetMasterFiles

This function displays the Master Document Options dialog box. Choosing this function is equivalent to choosing File/Master Document/Options.

This function does not automatically set the master document options. Please refer to the SetIndexFile and SetTOCFile functions.

Syntax

MasterDocOpts

Return Value

This function returns 0.

Example

```
FUNCTION Example()
MasterDocOpts
END FUNCTION
```

See also:

GetMasterFilesCount GetMasterFiles SetMasterFiles SetIndexFile SetTOCFile MasterDoc Generate TOCOptions

This function maximizes the Ami Pro window, causing it to fill the entire screen. Choosing this function is equivalent to choosing System/Maximize. If the window size is already maximized, executing this function restores the previous screen size. All other windows are moved behind the Ami Pro window when this function is executed.

Syntax

Maximize()

Return Value

This function does not return a value.

Example

```
FUNCTION Example()
Maximize()
END FUNCTION
```

See also:

Minimize Restore

This function merges a merge document with a data file. Choosing this function is equivalent to choosing File/Merge, selecting option 3, choosing OK, and then selecting Merge & print. Using this function sends the merge documents to the printer.

Before using the merge function, your macro should ensure that the merge document you want to merge to is on the screen.

Syntax

Merge(Flag, RecFile[, DescFile][, LabelsAcross, LabelsDown, OffsetRight, OffsetDown][, SelectionKey, SelectionKey, Operator, FieldNumber, NextOp...])

Flag is a number that defines which optional parameters are used and can be one of the following:

(0) - 0 selection criteria given
(1) - 1 selection criterion given
(2) - 2 selection criteria given
(3) - 3 selection criteria given
Description (8) - This merge uses a description file and includes the description filename
Labels (16) - This merge is for labels and includes label specifications

The number of criteria used (0 - 3) should be added to the description value and the label value, if they are used, to make up the value of the Flag parameter.

RecFile is the data file the merge document is merged with.

DescFile is the optional description file used when the data file is not an Ami Pro file. The DescFile should be used as the next parameter only if a description file is used.

All four label parameters must be given if merging labels. If this is a regular merge, all four label parameters should be skipped.

LabelsAcross is the optional number of labels across the page.

LabelsDown is the optional number of labels down the page.

OffsetRight is the optional distance to offset the first label to the right, in twips (1 inch = 1440 twips).

OffsetDown is the optional distance to offset the first label down, in twips (1 inch = 1440 twips).

SelectionKey is a string or number to use in the selection of records to merge. The compare key is a string, if using an alphanumeric comparison, or a number, if using a numeric comparison.

Operator is the operator to use in the selection comparison and can be one of the following values.

Equal (0) - The field's value must be equal to the key to merge.
LessThan (1) - The field's value must be less than the key to merge.
GreaterThan (2) - The field's value must be greater than the key to merge.
NotEqual (3) - The field's value must be different than the key to merge.
Lteq (4) - The field's value must be less than or equal to the key to merge.
Gteq (5) - The field's value must be greater than or equal to the key to merge.

FieldNumber is the number of the field to use for this selection. It is zero based. To compare against the first field in the data file, set field number to 0.

NextOp is a number that defines whether to select based on this criterion AND the next one, or this one OR the next one. If there are no more specifications this field should be zero. The values for this field are:

AndNext (1) - This selection AND the following selection must match to merge this record.
OrNext (2) - This selection OR the following selection must match to merge this record.

To display the Merge dialog box and allow the user to set merge specifications: **Merge**

Return Value

1 (TRUE) if the merge was completed.

0 (UserCancel) if the user canceled the function.

-3 if the input was invalid.

-2 (GeneralFailure) if the merge failed.

Example

```
FUNCTION Example()
Merge(0, "TEST.SAM")
END FUNCTION
```

See also:

MergeAction MergeMacro MergeToFile

This function is used in conjunction with the MergeMacro function to decide whether a merge document should be printed, not printed, canceled, or the rest of the documents should be printed. Choosing this function is equivalent to choosing File/Merge, selecting option 3, choosing OK, and then selecting Merge, view & print.

This function must be called once for each record merged and again after either the end of the data file has been reached or the user cancels the printing of the merged records.

Modified in 3.0 There is a new value, 5, for the Which parameter in Ami Pro release 3.0.

Syntax

MergeAction(Which)

Which is the action to take on the merged document and can be one of the following:

PrintDoc (1) - Print this one

MergeNext (2) - Do not print this one; prepare the next one

MergeCont (3) - Print this one; print the rest of them without stopping

MergeStop (4) - Do not print this one; cancel the rest of the merge

(5) - Print this one; prepare the next one

To display the Merge dialog box and allow the user to set merge specifications: **Merge**

Return Value

1 (TRUE) if the record was merged.

0 (UserCancel/FALSE) at the end of the data file, or if the user cancels printing of the merged record following a PrintDoc or MergeCont.

-2 (GeneralFailure) if the record was not merged.

Example

```
FUNCTION Example()
docdir = GetDocPath$()
datafile = "{docdir}mercdata.sam"
descfile = ""
numacross = 3
numdown = 10
labelindrt = 0.025
labelindtop = 0.5
'configure mergefile for printing
MergeMacro(24, datafile, descfile, numacross, numdown, labelindrt, labelindtop)
'dont actually print them; this associates the datafile
'and assigns the label configuration without printing.
MergeAction(mergestop)
END FUNCTION
```

See also:

Merge MergeMacro MergeToFile

This function merges a merge document with a data file, then allows the macro to examine the merged document, edit it, and decide if it should be printed. Choosing this function is equivalent to choosing File/Merge, selecting option 3, choosing OK, and then selecting Merge, view & print. Once the first document has been merged, the MergeAction function is used to decide what should be done with the document. Choosing this function is equivalent to choosing the dialog box that appears onscreen during Merge, view & print.

Before using the merge function, your macro should ensure that the merge document you want to merge to is on the screen.

Syntax

MergeMacro(Flag, RecFile[, DescFile][, LabelsAcross, LabelsDown, OffsetRight, OffsetDown][, SelectionKey, SelectionKey, Operator, FieldNumber, NextOp...])

The Flag parameter and optional parameters are identical to those used in the Merge function. For a list of their values, refer to the section of the documentation on the Merge function.

Flag is a number that defines which optional parameters are used and can be one of the following:

(0) - 0 selection criteria given
(1) - 1 selection criterion given
(2) - 2 selection criteria given
(3) - 3 selection criteria given
Description (8) - This merge uses a description file and includes the description filename
Labels (16) - This merge is for labels and includes label specifications

The number of criteria used (0-3) should be added to the description value and the label value, if they are used, to make up the value of the Flag parameter.

RecFile is the data file the merge document is merged with. If RecFile is a NewWave object, this parameter must be the full path to the object name.

DescFile is the description file used when the data file is not an Ami Pro file. If DescFile is a NewWave object, this parameter must be the full path to the object name.

All four label parameters must be given if merging labels. If this is a regular merge, all four label parameters should be skipped.

LabelsAcross is the number of labels across the page if merging labels.

LabelsDown is the number of labels down the page if merging labels.

OffsetRight is the distance to offset the first label to the right, in twips (1 inch = 1440 twips).

OffsetDown is the distance to offset the first label down, in twips (1 inch = 1440 twips).

SelectionKey is a string or number to use in the selection of records to merge. The compare key is a string, if using an alphanumeric comparison, or a number, if using a numeric comparison.

Operator is the operator to use in the selection comparison and can be one of the following values:

Equal (0) - The field's value must be equal to the key to merge.
LessThan (1) - The field's value must be less than the key to merge.
GreaterThan (2) - The field's value must be greater than the key to merge.
NotEqual (3) - The field's value must be different than the key to merge.
Lteq (4) - The field's value must be less than or equal to the key to merge.
Gteq (5) - The field's value must be greater than or equal to the key to merge.

FieldNumber is the number of the field to use for this selection. It is zero based. To compare against the first field in the data file, set field number to 0.

NextOp is a number that defines whether to select based on this criterion AND the next one, or this one OR the next one. If there are no more specifications this field should be zero. The values for this field are:

AndNext (1) - This selection AND the following selection must match to merge this record.
OrNext (2) - This selection OR the following selection must match to merge this record.

To display the Merge dialog box and allow the user to set merge specifications: **Merge**

Return Value

1 (TRUE) if the merge was completed.

0 (UserCancel) if the user canceled the function.

-2 (GeneralFailure) if the merge failed.

Example

```
FUNCTION Example()
docdir = GetDocPath$()
datafile = "{docdir}mercdata.sam"
descfile = ""
numacross = 3
numdown = 10
labelindrt = 0.025
labelindtop = 0.5
'configure mergefile for printing
MergeMacro(24, datafile, descfile, numacross, numdown, labelindrt, labelindtop)
'dont actually print them; this associates the datafile
'and assigns the label configuration without printing.
MergeAction(mergestop)
END FUNCTION
```

See also:

Merge MergeAction MergeToFile

This function merges a merge document with a data file and places the result in another file for later editing or printing. Choosing this function is equivalent to choosing File/Merge, selecting option 3, choosing OK, and then selecting Merge & save as.

Before using the merge function, your macro should ensure that the merge document you want to merge to is on the screen.

Modified in 3.0 There are two values, 64 and 136, that are only available in Ami Pro release 2.0.

Syntax

MergeToFile(Flag, RecFile, OutFile[, DescFile][, LabelsAcross, LabelsDown, OffsetRight, OffsetDown][, SelectionKey, SelectionKey, Operator, FieldNumber, NextOp]...)

Flag is a number that defines which optional parameters are used. The flag parameter defines which of the optional parameters are used, according to the following list:

(0) - 0 selection criteria given

(1) - 1 selection criterion given

(2) - 2 selection criteria given

(3) - 3 selection criteria given

Description (8) - This merge uses a description file, and includes the description filename

Labels (16) - This merge is for labels, and includes label specifications

ToFile (32) - This merge is merged to a file

For NewWave, the following additional values apply for the Flag parameter:

RecObject (64) - If the RecFile is an object name

DescObjects (136) - If the DescFile is an object name

The number of criteria used (0 - 3) must be added to the description value and the label value, if they are used. Because this file merges, 32 must be added to this result to make up the value of the flag parameter.

RecFile is the data file the merge document is merged with. If RecFile is a NewWave object, this parameter must be the full path to the object name.

OutFile is the file to which the merged document is saved.

DescFile is the description file used when the data file is not an Ami Pro file. If DescFile is a NewWave object, this parameter must be the full path to the object name.

All four label parameters must be given if merging labels. If this is a regular merge, all four label parameters should be skipped.

LabelsAcross is the number of labels across the page if merging labels.

LabelsDown is the number of labels down the page if merging labels.

OffsetRight is the distance to offset the first label to the right, in twips (1 inch = 1440 twips).

OffsetDown is the distance to offset the first label down, in twips (1 inch = 1440 twips).

SelectionKey is a string or number to use in the selection of records to merge. The compare key is a string, if using an alphanumeric comparison, or a number, if using a numeric comparison.

Operator is the operator to use in the selection comparison and can be one of the following values:

Equal (0) - The field's value must be equal to the key to merge.

LessThan (1) - The field's value must be less than the key to merge.

GreaterThan (2) - The field's value must be greater than the key to merge.

NotEqual (3) - The field's value must be different than the key to merge.

Lteq (4) - The field's value must be less than or equal to the key to merge.

Gteq (5) - The field's value must be greater than or equal to the key to merge.

FieldNumber is the number of the field to use for this selection. It is zero based. To compare against the

first field in the data file, set field number to 0.

NextOp is a number that defines whether to select based on this criterion AND the next one or this one OR the next one. If there are no more specifications, this field should be zero. The values for this field are:

AndNext (1) - This selection AND the following selection must match to merge this record.

OrNext (2) - This selection OR the following selection must match to merge this record.

To display the Merge dialog box and allow the user to set merge specifications: **Merge**

Return Value

0 (UserCancel) if the user canceled the function.

-2 (GeneralFailure) if the merge failed.

Example

```
FUNCTION Example()
MergeToFile(32 "mrgdata.SAM"  "output.SAM")
END FUNCTION
```

See also:

Merge MergeAction MergeMacro

This function displays a Windows Message box with the specified title, prompt, and an OK push button. It waits for the user to acknowledge the message by selecting OK.

Modified in 3.0 There is a new parameter, Title, in Ami Pro release 3.0.

Syntax

Message(Prompt[, Title])

Prompt is a string used as a prompt to the user. It can be a maximum of 80 characters.

Title is the title for the message box. The default is "Ami Pro Macro".

Return Value

This function does not return a value.

Example

```
FUNCTION Example()
name = "Mark Olsen"
Message("Hello, {name}.", "My Message")
END FUNCTION
```

See also:

Decide DialogBox Messages MultiDecide Query$ UserControl

This function allows you to determine whether the user should respond to certain program messages during macro execution. Many Ami Pro functions display message boxes asking the user to confirm an action before it is taken. If messages display is off, the macro forces Ami Pro to take the default action proposed by the message box (as if the user had pressed **ENTER**). If messages display is on, the message box displays, and the user needs to intervene before macro execution can continue.

If a specific reply is required to a specific message, the AnswerMsgBox function must be used to reply to a specific function.

Syntax

Messages(State)

State determines whether unexpected messages appear during macro execution.

On (1) - Display the messages

Off (0) - Accept the default replies and not display the messages

Return Value

This function does not return a value.

Example

```
FUNCTION Example()
Messages(Off)
FileClose()
Messages(On)
END FUNCTION
```

See also:

AnswerMsgBox SingleStep UserControl IgnoreKeyboard

This function is used to extract a portion of a string.

Syntax

MID$(Text, Offset, Length)

Text is the string from which a shorter string is extracted.

Offset is the location in the specified string to start parsing the new string. Offset is one based; to extract from the beginning of the longer string, use an offset of 1 or the function Left$.

Length is the number of characters to extract out of the specified string.

Return Value

The requested text.

The null string ("") if the starting offset is beyond the end of the larger string.

Example

```
FUNCTION Example()
txt="Hello, my name is Susan Butler."
newtext=MID$(txt, 7, 11)
Message(newtext)   'returns "my name is"
END FUNCTION
```

See also:

ASC CHR$ strcat$ LCASE$ UCASE$ strfield$ LEN strchr FormatNum$

This function minimizes the Ami Pro window by reducing it to an icon. Choosing this function is equivalent to choosing System/Minimize. If the window is already reduced to an icon, executing this function restores the previous window size. When Ami Pro is minimized, macro execution continues in the background.

Syntax

Minimize()

Return Value

This function does not return a value.

Example

```
FUNCTION Example()
Minimize()
END FUNCTION
```

See also:

Maximize Restore

This function returns the remainder of the first parameter divided by the second.

Syntax

Mod(Numerator, Denominator)

Numerator is the number to be divided.

Denominator is the number to divide into the numerator.

Return Value

The remainder of the numerator divided by the denominator

-2 (GeneralFailure) if the numbers could not be divided

Example

```
FUNCTION Example()
Top = Query$("Enter the value for the numerator:")
Bottom = Query$("Enter the value for the denominator:")
Result = Mod(Top, Bottom)
Message(Result)
END FUNCTION
```

See also:

Round IsNumeric

This function modifies paragraph style alignment options. Choosing this function is equivalent to choosing Style/Modify Style/Alignment.

ModifySelect should be called before this function to define the style to modify.

Before the new alignment options take effect for the selected paragraph style, the ModifyReflow function should be called in the macro.

Syntax

ModifyAlignment(Options, AllLevel, FirstLevel, RestLevel, Units, RightIndent)

Options is a flag parameter containing the paragraph style's alignment options. Set the Options parameter to one or more of the following options:

AlignLeft (1) - Left align the paragraph
AlignRight (2) - Right align the paragraph
AlignCenter (4) - Center the paragraph
AlignJustify (8) - Justify the paragraph
IndentBoth (16) - Indent both sides of the paragraph
NoIndentAll (32) - Do not use entire paragraph indention option
NoIndentFirst (64) - Do not use first line indention option
NoIndentRest (128) - Do not use rest of lines indention options
Hyphenate (256) - Hyphenate the paragraph
HangingIndent (512) - This paragraph has a hanging indent

One of the four alignment options should always be chosen. Multiple options should be added together before they are passed to the function. All levels are in twips (1 inch = 1440 twips).

AllLevel is the level to indent all lines in the paragraph. This parameter is ignored if the NoIndentAll flag is set.

FirstLevel is the level to indent the first line of the paragraph. This parameter is ignored if the NoIndentFirst flag is set.

RestLevel is the level to indent the rest of the lines in the paragraph. This parameter is ignored if the NoIndentRest flag is set.

Units is the amount of indention to be selected from the Modify Style dialog box.

Inches (1) - units set to inches
CM (2) - units set to centimeters
Picas (3) - units set to picas
Points (4) - units set to points

RightIndent is the level to indent all lines in the paragraph from the right. This parameter is ignored if the NoIndentRest flag is set.

To display the Modify Style dialog box and allow the user to select the options for modifying paragraph styles: **ModifyStyle**

To modify the tabs used when modifying the style, refer to the ModifyBreaks function.

Return Value

1 (TRUE) if the alignment options were set.

0 (UserCancel) if the user canceled the function.

-2 (GeneralFailure) if the options were not set.

Example

```
FUNCTION Example()
NewName = Query$("What do you want to name the new style?", "TestStyle")
```

```
BaseName = GetStyleName$()
CreateStyle(NewName, BaseName, 0)
ModifySelect(NewName)
ModifyAlignment(AlignLeft, 0, 0, 0, 0, 0)
ModifyBreaks(4, 0, 0)
ModifyEffects("<·10>", SpaceIndent, ".20", 0, 0, 0, 0)
ModifySpacing(2, 0, 0, 0, 90, 0, 100)
ModifyTable(3, 2, ".", ",", "$", (8 + 16 + 128 + 256))
ModifyLines(1, 2, 180, 0, 0, 0, 0, 65535)
ModifyFont("TimesNewRomanPS", (20 * 20), 255, 1)
SetStyle(NewName)
ModifyReflow()
TYPE("This[Enter]is what the new[Enter]style looks[Enter]Like...[Enter]")
SetStyle(BaseName)
END FUNCTION
```

See also:

CreateStyle DefineStyle ModifyBreaks ModifyEffects ModifyFont ModifyLines ModifyReflow ModifySelect ModifySpacing ModifyStyle ModifyTable

This function modifies paragraph style page break options. Choosing this function is equivalent to choosing Style/Modify Style/Breaks.

ModifySelect should be called before this function to define the style to modify.

Before the new page break options take effect for the selected paragraph style, the ModifyReflow function must be called in the macro.

Syntax

ModifyBreaks(Options, Style, NumTabs[, Type, Offset])

Options is a flag parameter specifying the break options. Set the Options parameter to one or more of the following options:

NoBreaks (0) - Do not allow page breaks within this paragraph

PageBreakBefore (1) - Put a page break before this paragraph

PageBreakAfter (2) - Put a page break following this paragraph

BreakWithin (4) - Allow page breaks within the paragraph

KeepPrevious (8) - Keep this paragraph with the previous paragraph

KeepNext (16) - Keep this paragraph with the next paragraph

(64) - If set, the style defined in the Style parameter is applied to the next paragraph

ColBreakBefore (128) - Put a column break before this paragraph

ColBreakAfter (256) - Put a column break after this paragraph

One of the four break options should always be chosen. The KeepPrevious option may be chosen if the NoBreak, PageBreakAfter, or ColBreakAfter option is chosen. The KeepNext option may be chosen if the NoBreak, PageBreakBefore, or ColBreakBefore option is chosen.

Style is the paragraph style to use for the next paragraph. The next style (64) option must also be set in order to apply this paragraph style.

NumTabs is the number of tabs to set for this style. You must enter a Type and Offset for each tab.

To modify the tabs in the Modify Style dialog box, select Alignment.

Type is the type of tab. The Offset parameter must also be used for each tab.

TabLeft (1) - Left tab

TabCenter (2) - Center tab

TabRight (3) - Right tab

TabNumeric (4) - Numeric tab

To display leaders preceding the tab, add one of the following values to the type of tab:

TabHyph (16384) - Displays hyphens

TabDot (32768) - Displays dots

TabLine (49152) - Displays a line

Offset is the distance of the tab from the left margin and must be given in twips. (1 inch = 1440 twips). The Type parameter must also be used for each tab.

To display the Modify Style dialog box and allow the user to select the options for modifying paragraph styles: **ModifyStyle**

Return Value

1 (TRUE) if the break options were set.

0 (UserCancel) if the user canceled the function.

-2 (GeneralFailure) if the options were not set.

Example

```
FUNCTION Example()
NewName = Query$("What do you want to name the new style?", "TestStyle")
BaseName = GetStyleName$()
CreateStyle(NewName, BaseName, 0)
ModifySelect(NewName)
ModifyAlignment(AlignLeft, 0, 0, 0, 0, 0)
ModifyBreaks(4, 0, 0)
ModifyEffects("<·10>", SpaceIndent, ".20", 0, 0, 0, 0)
ModifySpacing(2, 0, 0, 0, 90, 0, 100)
ModifyTable(3, 2, ".", ",", "$", (8 + 16 + 128 + 256))
ModifyLines(1, 2, 180, 0, 0, 0, 0, 65535)
ModifyFont("TimesNewRomanPS", (20 * 20), 255, 1)
SetStyle(NewName)
ModifyReflow()
TYPE("This[Enter]is what the new[Enter]style looks[Enter]Like...[Enter]")
SetStyle(BaseName)
END FUNCTION
```

See also:

CreateStyle DefineStyle ModifyAlignment ModifyEffects ModifyFont ModifyLines ModifyReflow ModifySelect ModifySpacing ModifyStyle ModifyTable

This function modifies paragraph style bullets and numbers. Choosing this function is equivalent to choosing Style/Modify Style/Bullets & numbers.

ModifySelect should be called before this function to define the style to modify.

Before the new bullets and numbering options take effect for the selected paragraph style, the ModifyReflow function must be called in the macro.

If the ModifyEffects function is included in a macro created in Ami Pro 1.2B, the attribute values must be changed in Ami Pro 3.0 or the macro plays back unpredictable results.

Modified in 3.0 There are two new values, #6 and #7, for Text in Ami Pro release 3.0. In Ami Pro release 2.0, the parameter Align is the parameter Reset.

Syntax

ModifyEffects(Text, Spacing, Indent, Attr, LevelNum, Align, Units)

Text is the leading text for the paragraph. The Text parameter specifies the text, bullet, or numbers that precede each paragraph using this paragraph style. The text must be typed as it would appear in the edit box in the Bullets & numbers panel in the Modify Style dialog box. Bullets can be inserted by typing the following text where the bullet should be placed:

<·1> - Small Round Bullet
<·2> - Large Round Bullet
<·3> - Small Square Bullet
<·4> - Large Square Bullet
<·5> - Large Outline Square Bullet
<·6> - Small Diamond Bullet
<·7> - Large Diamond Bullet
<·8> - Small Open Circle Bullet
<·9> - Large Open Circle Bullet
<·10> - Check Mark
<·11> - Tack
<·12> - Square shadow below bullet
<·13> - Square shadow above bullet
<·14> - Check box
<·15> - Square with X bullet
<·16> - Rounded arrowhead top shaded
<·17> - Rounded arrowhead bottom shaded

To type the dot in front of the bullet number, turn on Num Lock, hold the ALT key, type 0183 and release the ALT key. Numbers can be inserted by typing the following text where the number should be inserted:

<#1> - 1 2 3 4 5...
<#2> - A B C D E...
<#3> - a b c d e...
<#4> - I II III IV V...
<#5> - i ii iii iv v...
<#6> - * ** *** ****...
<#7> - (dagger characters)...

Spacing is a flag indicating how to space between the text and the paragraph body. The Spacing parameter should be set to one of the following options:

TabIndent (0) - Separate leading text from body with a tab

SpaceIndent (1) - Separate leading text from body with the space specified in the Indent parameter

Indent is the amount of space to use between the text and the paragraph body. The SpaceIndent option should be set for this spacing to be used.

The amount of indention must be given in twips (1 inch = 1440 twips).

The right alignment option must have enough indention space to line up on the right. For example, when using asterisks, there must be enough space to line up on the rightmost asterisk.

Attr is a flag parameter specifying the attribute options for the leading text and can be one of the following:

NormalAttr (0) - Text has no attributes
BoldText (4) - Bold text
ItalicText (8) - Italicized text
UnderlineText (16) - Underlined text
WordUnderlineText (32) - Word underlined text
(2048) - Capitalized text
SuperScript (64) - Superscripted text

Multiple attributes can be added together. Not all fonts have all attributes and the underline attributes cannot be combined.

LevelNum is the outlining level number to use and should be a number from 0 to 9. If this is not an outline style, this option should be set to 0.

Align is a flag determining whether to right align the text in the special effect and can be one of the following:

(0) - Do not right align
(16) - Right align

Reset is a flag determining when to reset the number and can be one of the following:

ResetNo (0) - Do not reset numbering or no numbering used
ResetLesser (2) - Reset number after a paragraph style with a lesser numbering level
ResetIntervene (4) - Reset number after any intervening paragraph style
LegalNumbering (8) - This is a legal numbering paragraph style; for example, 1.1.1

Units is the amount of indention to be selected from the Modify Style dialog box.

Inches (1) - units set to inches
CM (2) - units set to centimeters
Picas (3) - units set to picas
Points (4) - units set to points

The amount of indention must be given in twips (1 inch = 1440 twips).

To display the Modify Style dialog box and allow the user to select the options for modifying paragraph styles: **ModifyStyle**

Return Value

1 (TRUE) if the effects options were set.

0 (UserCancel) if the user canceled the function.

-2 (GeneralFailure) if the options were not set.

Example

```
FUNCTION Example()
NewName = Query$("What do you want to name the new style?", "TestStyle")
BaseName = GetStyleName$()
CreateStyle(NewName, BaseName, 0)
ModifySelect(NewName)
ModifyAlignment(AlignLeft, 0, 0, 0, 0, 0)
ModifyBreaks(4, 0, 0)
```

```
ModifyEffects("<·10>", SpaceIndent, ".20", 0, 0, 0, 0)
ModifySpacing(2, 0, 0, 0, 90, 0, 100)
ModifyTable(3, 2, ".", ",", "$", (8 + 16 + 128 + 256))
ModifyLines(1, 2, 180, 0, 0, 0, 0, 65535)
ModifyFont("TimesNewRomanPS", (20 * 20), 255, 1)
SetStyle(NewName)
ModifyReflow()
TYPE("This[Enter]is what the new[Enter]style looks[Enter]Like...[Enter]")
SetStyle(BaseName)
END FUNCTION
```

See also:

CreateStyle DefineStyle ModifyAlignment ModifyBreaks ModifyFont ModifyLines ModifyReflow ModifySelect ModifySpacing ModifyStyle ModifyTable

This function modifies paragraph style font. Choosing this function is equivalent to choosing Style/Modify Style/Font. Through the menu commands, the user can only select from the fonts available on the printer. Using the macro commands, any type of font can be requested.

ModifySelect should be called before this function to define the style to modify.

Before the new font options take effect for the selected paragraph style, the ModifyReflow function must be called in the macro.

If the ModifyFont function is included in a macro created in Ami Pro 1.2B, the attribute values must be changed in Ami Pro release 2.0 and higher or the macro plays back unpredictable results.

Syntax

ModifyFont(FontName, Size, Color, Options)

FontName is the name of the font as listed in the Face list box.

Size is the size of the font, in twips. The formula for finding the proper point size is *pointsize * 20 = twips.*

Color is the color of the font and can be one of the following:

White (16777215) - White
Cyan (16776960) - Light blue
Yellow (65535) - Yellow
Magenta (16711935) - Purple
Green (65280) - Green
Red (255) - Red
Blue (16711680) - Blue
Black (0) - Black

Options is a flag parameter specifying the attribute options and can be one or more of the following:

NormalAttr (0) - Text has no attributes
BoldText (4) - Bold text
ItalicText (8) - Italicized text
UnderlineText (16) - Underlined text
WordUnderlineText (32) - Word underlined text
UpperCase (2048) - All caps text
DoubleLineAttr (256) - Double underline text
InitialCaps (8192) - Initial caps text
(32768) - First line bold text
VarPitch (1) - Variable pitch font
Serifs (1024) - Serifs in font

Multiple attributes can be added together. Not all the fonts have all the attributes and some attributes, such as the caps attributes and the underline attributes, cannot be combined.

To display the Modify Style dialog box and allow the user to select the options for modifying paragraph styles: **ModifyStyle**

Return Value

1 (TRUE) if the font options were set.

-2 (GeneralFailure) if the options were not set.

Example

```
FUNCTION Example()
NewName = Query$("What do you want to name the new style?", "TestStyle")
BaseName = GetStyleName$()
```

```
CreateStyle(NewName, BaseName, 0)
ModifySelect(NewName)
ModifyAlignment(AlignLeft, 0, 0, 0, 0, 0)
ModifyBreaks(4, 0, 0)
ModifyEffects("<·10>", SpaceIndent, ".20", 0, 0, 0, 0)
ModifySpacing(2, 0, 0, 0, 90, 0, 100)
ModifyTable(3, 2, ".", ",", "$", (8 + 16 + 128 + 256))
ModifyLines(1, 2, 180, 0, 0, 0, 0, 65535)
ModifyFont("TimesNewRomanPS", (20 * 20), 255, 1)
SetStyle(NewName)
ModifyReflow()
TYPE("This[Enter]is what the new[Enter]style looks[Enter]Like...[Enter]")
SetStyle(BaseName)
END FUNCTION
```

See also:

CreateStyle DefineStyle ModifyAlignment ModifyBreaks ModifyEffects ModifyLines ModifyReflow ModifySelect ModifySpacing ModifyStyle ModifyTable

This function displays the Modify Page Layout dialog box. Choosing this function is equivalent to choosing Page/Modify Page Layout. This function does not automatically modify the page layout.

You must be in Layout Mode to use this function.

Syntax

ModifyLayout()

Return Value

1 (TRUE) if the page layout was modified.

0 (UserCancel) if the user canceled the function.

-2 (GeneralFailure) if the layout was not modified.

Example

```
FUNCTION Example()
ModifyLayout()
END FUNCTION
```

See also:

FrameLayout ModLayoutFinish ModLayoutInit ModLayoutLeftFooter ModLayoutLeftHeader ModLayoutLeftLines ModLayoutLeftPage ModLayoutPageSize ModLayoutRightFooter ModLayoutRightHeader ModLayoutRightLines ModLayoutRightPage ModifyStyle

This function modifies paragraph style lines. Choosing this function is equivalent to choosing Style/Modify Style/Lines.

ModifySelect should be called before this function to define the style to modify.

Before the new line options take effect for the selected paragraph style, the ModifyReflow function must be called in the macro.

Syntax

ModifyLines(Options, StyleAbove, SpaceAbove, StyleBelow, SpaceBelow, Length, Color, Units)

Options is a flag parameter specifying the line options. The Options parameter consists of one or more of the following options:

Off (0) - No lines above or below
LineAbove (1) - Put a line above the paragraph
LineBelow (2) - Put a line below the paragraph
LengthOfText (4) - Line is the length of the text
LengthMargins (8) - Line is the length of the margins
LengthCustom (16) - Line's length is determined by the length parameter

StyleAbove and **StyleBelow** determine the type of line to print above and below the paragraph, respectively. Available line styles are:

Hairline (1) - Hairline
OnePoint (2) - One point rule
TwoPoint (3) - Two point rule
ThreePoint (4) - Three point rule
FourPoint (5) - Four point rule
FivePoint (6) - Five point rule
SixPoint (7) - Six point rule
DoubleOnePoint (8) - Parallel one point rules
DoubleTwoPoint (9) - Parallel two point rules
ThreeLines (10) - Hairline above and below a two point rule
HairBelow (11) - Hairline below a three point rule
HairAbove (12) - Hairline above a three point rule

SpaceAbove is the spacing between the font and the line above the text, in twips. If none, use 0 for this parameter.

SpaceBelow is the spacing between the font and the line below the text, in twips. If none, use 0 for this parameter.

Length is the length of the line, in twips, if a custom length.

Color is the color of the line and can be one of the following:

White (16777215) - White
Cyan (16776960) - Light blue
Yellow (65535) - Yellow
Magenta (16711935) - Purple
Green (65280) - Green
Red (255) - Red
Blue (16711680) - Blue
Black (0) - Black
DarkGray (12566463) - 90% gray scale

MediumGray (8355711) - 50% gray scale
LightGray (4144959) - 20% gray scale
VeryLightGray (1644825) - 10% gray scale

Units is the amount of indention to be selected from the Modify Style dialog box.

Inches (1) - units set to inches
CM (2) - units set to centimeters
Picas (3) - units set to picas
Points (4) - units set to points

The amount of indention must be given in twips (1 inch = 1440 twips).

To display the Modify Style dialog box and allow the user to select the options for modifying paragraph styles: **ModifyStyle**

Return Value

1 (TRUE) if the line options were set.

-2 (GeneralFailure) if the options were not set.

Example

```
FUNCTION Example()
NewName = Query$("What do you want to name the new style?", "TestStyle")
BaseName = GetStyleName$()
CreateStyle(NewName, BaseName, 0)
ModifySelect(NewName)
ModifyAlignment(AlignLeft, 0, 0, 0, 0, 0)
ModifyBreaks(4, 0, 0)
ModifyEffects("<·10>", SpaceIndent, ".20", 0, 0, 0, 0)
ModifySpacing(2, 0, 0, 0, 90, 0, 100)
ModifyTable(3, 2, ".", ",", "$", (8 + 16 + 128 + 256))
ModifyLines(1, 2, 180, 0, 0, 0, 0, 65535)
ModifyFont("TimesNewRomanPS", (20 * 20), 255, 1)
SetStyle(NewName)
ModifyReflow()
TYPE("This[Enter]is what the new[Enter]style looks[Enter]Like...[Enter]")
SetStyle(BaseName)
END FUNCTION
```

See also:

CreateStyle DefineStyle ModifyAlignment ModifyBreaks ModifyEffects ModifyFont ModifyReflow ModifySelect ModifySpacing ModifyStyle ModifyTable

This function applies changes that have been made to paragraph styles using the other modify paragraph style functions. This function should be called after using the modify paragraph style functions and before editing other text. Choosing this function is equivalent to choosing Style/Modify Style.

This function should be called after using other modify paragraph style functions.

Syntax

ModifyReflow()

Return Value

1 (TRUE) if the changed paragraph styles were applied.

-2 (GeneralFailure) if the paragraph styles were not applied.

Example

```
FUNCTION Example()
NewName = Query$("What do you want to name the new style?", "TestStyle")
BaseName = GetStyleName$()
CreateStyle(NewName, BaseName, 0)
ModifySelect(NewName)
ModifyAlignment(AlignLeft, 0, 0, 0, 0, 0)
ModifyBreaks(4, 0, 0)
ModifyEffects("<·10>", SpaceIndent, ".20", 0, 0, 0, 0)
ModifySpacing(2, 0, 0, 0, 90, 0, 100)
ModifyTable(3, 2, ".", ",", "$", (8 + 16 + 128 + 256))
ModifyLines(1, 2, 180, 0, 0, 0, 0, 65535)
ModifyFont("TimesNewRomanPS", (20 * 20), 255, 1)
SetStyle(NewName)
ModifyReflow()
TYPE("This[Enter]is what the new[Enter]style looks[Enter]Like...[Enter]")
SetStyle(BaseName)
END FUNCTION
```

See also:

CreateStyle DefineStyle ModifyAlignment ModifyBreaks ModifyEffects ModifyFont ModifyLines ModifySelect ModifySpacing ModifyStyle ModifyTable

This function selects a paragraph style to modify. The other Modify Paragraph Style functions act on the paragraph style selected by this function. Choosing this function is the equivalent to choosing Style/Select a Style.

This function should be called prior to using other modify paragraph style functions.

Syntax

ModifySelect(Style)

Style is the name of an existing paragraph style.

To display the Modify Style dialog box and allow the user to select the options for modifying paragraph styles: **ModifyStyle**

Return Value

1 (TRUE) if the paragraph style was selected.

-2 (GeneralFailure) if the paragraph style was not selected.

Example

```
FUNCTION Example()
NewName = Query$("What do you want to name the new style?", "TestStyle")
BaseName = GetStyleName$()
CreateStyle(NewName, BaseName, 0)
ModifySelect(NewName)
ModifyAlignment(AlignLeft, 0, 0, 0, 0, 0)
ModifyBreaks(4, 0, 0)
ModifyEffects("<·10>", SpaceIndent, ".20", 0, 0, 0, 0)
ModifySpacing(2, 0, 0, 0, 90, 0, 100)
ModifyTable(3, 2, ".", ",", "$", (8 + 16 + 128 + 256))
ModifyLines(1, 2, 180, 0, 0, 0, 0, 65535)
ModifyFont("TimesNewRomanPS", (20 * 20), 255, 1)
SetStyle(NewName)
ModifyReflow()
TYPE("This[Enter]is what the new[Enter]style looks[Enter]Like...[Enter]")
SetStyle(BaseName)
END FUNCTION
```

See also:

CreateStyle DefineStyle ModifyAlignment ModifyBreaks ModifyEffects ModifyFont ModifyLines ModifyReflow ModifySpacing ModifyStyle ModifyTable

This function modifies paragraph style spacing. Choosing this function is equivalent to choosing Style/Modify Style/Spacing.

ModifySelect should be called before this function to define the style to modify.

Before the new spacing options take effect for the selected paragraph style, the ModifyReflow function must be called in the macro.

Syntax

ModifySpacing(Options, Amount, ParaAbove, ParaBelow, Tightness, SpacingUnits, ParaUnits)

Options is a flag parameter with spacing options. The options parameter should be set to one of the following options:

SingleSpacing (1) - Use single line spacing
OneOneHalfSpacing (2) - One and one half line spacing
DoubleSpacing (4) - Double line spacing
CustomSpacing (8) - Custom line spacing
AddAlways (16) - Always use line above spacing
AddNotBreak (32) - Use line above spacing only if not at page break

Amount is the line spacing to use if custom spacing is selected.

ParaAbove is the spacing to use above the paragraph, in twips (1 inch = 1440 twips).

ParaBelow is the spacing to use below the paragraph, in twips (1 inch = 1440 twips).

Tightness is the line tightness factor to use and can be one of the following:

TightLines (90) - 90% line tightness
NormalLines (100) - 100% line tightness (normal)
LooseLines (115) - 115% line tightness

SpacingUnits is the amount of line spacing to be selected from the Modify Style dialog box.

Inches (1) - units set to inches
CM (2) - units set to centimeters
Picas (3) - units set to picas
Points (4) - units set to points

The amount of indention must be given in twips (1 inch = 1440 twips).

ParaUnits is the amount of spacing for above and below the paragraph to be selected from the Modify Style dialog box.

Inches (1) - units set to inches
CM (2) - units set to centimeters
Picas (3) - units set to picas
Points (4) - units set to points

To display the Modify Style dialog box and allow the user to select the options for modifying paragraph styles: **ModifyStyle**

Return Value

1 (TRUE) if the spacing was set.

-2 (GeneralFailure) if the spacing was not set.

Example

```
FUNCTION Example()
NewName = Query$("What do you want to name the new style?", "TestStyle")
BaseName = GetStyleName$()
CreateStyle(NewName, BaseName, 0)
```

```
ModifySelect(NewName)
ModifyAlignment(AlignLeft, 0, 0, 0, 0, 0)
ModifyBreaks(4, 0, 0)
ModifyEffects("<·10>", SpaceIndent, ".20", 0, 0, 0, 0)
ModifySpacing(2, 0, 0, 0, 90, 0, 100)
ModifyTable(3, 2, ".", ",", "$", (8 + 16 + 128 + 256))
ModifyLines(1, 2, 180, 0, 0, 0, 0, 65535)
ModifyFont("TimesNewRomanPS", (20 * 20), 255, 1)
SetStyle(NewName)
ModifyReflow()
TYPE("This[Enter]is what the new[Enter]style looks[Enter]Like...[Enter]")
SetStyle(BaseName)
END FUNCTION
```

See also:

CreateStyle DefineStyle ModifyAlignment ModifyBreaks ModifyEffects ModifyFont ModifyLines ModifyReflow ModifySelect ModifyStyle ModifyTable

This function displays the Modify Style dialog box. Choosing this function is equivalent to choosing Style/Modify Style, but it cannot automatically modify a paragraph style. To allow the macro to modify paragraph styles directly, use the other modify paragraph style functions.

Syntax

ModifyStyle()

Return Value

1 (TRUE) if the paragraph style was modified.

0 (UserCancel) if the user canceled the function.

-2 (GeneralFailure) if the paragraph style could not be modified.

Example

```
FUNCTION Example()
ModifyStyle()
END FUNCTION
```

See also:

CreateStyle DefineStyle GetStyleName$ ModifyAlignment ModifyBreaks ModifyEffects ModifyFont ModifyLines ModifyReflow ModifySelect ModifySpacing ModifyTable

This function modifies paragraph style table format. Choosing this function is equivalent to choosing Style/Modify Style/Table format.

ModifySelect should be called before this function to define the style to modify.

Before the new table format options take effect for the selected paragraph style, the ModifyReflow function must be called in the macro.

Syntax

ModifyTable(CellFormat, Decimals, DecimalPoint, Separator, CurrencySymbol, Options)

CellFormat determines the cell format and can be one of the following:

GeneralFormat (1) - Show decimals only if required

FixedFormat (2) - Show a fixed number of decimals

CurrencyFormat (3) - Show the currency symbol

PercentFormat (4) - Display the number with a percent sign

Decimals is the number of decimal places to display for numbers.

DecimalPoint is the character to use for the decimal point.

Separator is the character to use as the thousands separator.

CurrencySymbol is the character to use for the currency symbol.

Options is a flag parameter with table format options and can be one of the following:

ThousandsSep (8) - Use thousands separator character. The ThousandsSeparator option should be set if thousands separators are to display and print. Either LeadingNegative, TrailingNegative, or ParenthesesNegative should be chosen to determine how to display negative numbers. The RedNegative option can be used to display and print negative numbers in red. Either the CurrencyLead or CurrencyTrail option should be used to indicate the position of the currency symbol.

LeadingNegative (16) - Use leading hyphen to indicate negative numbers

TrailingNegative (32) - Use trailing hyphen to indicate negative numbers

ParenthesesNegative (64) - Use parentheses to indicate negative numbers

RedNegative (128) - Display/print negative numbers in red

CurrencyLead (256) - Place currency symbol ahead of number

CurrencyTrail (0) - Place currency symbol after number

To display the Modify Style dialog box and allow the user to select the options for modifying paragraph styles: **ModifyStyle**

Return Value

1 (TRUE) if the table options were set.

-2 (GeneralFailure) if the options were not set.

Example

```
FUNCTION Example()
NewName = Query$("What do you want to name the new style?", "TestStyle")
BaseName = GetStyleName$()
CreateStyle(NewName, BaseName, 0)
ModifySelect(NewName)
ModifyAlignment(AlignLeft, 0, 0, 0, 0, 0)
ModifyBreaks(4, 0, 0)
ModifyEffects("<·10>", SpaceIndent, ".20", 0, 0, 0,0)
ModifySpacing(2, 0, 0, 0, 90, 0, 100)
ModifyTable(3, 2, ".", ",", "$", (8 + 16 + 128 + 256))
ModifyLines(1, 2, 180, 0, 0, 0, 0, 65535)
ModifyFont("TimesNewRomanPS", (20 * 20), 255, 1)
```

```
SetStyle(NewName)
ModifyReflow()
TYPE("This[Enter]is what the new[Enter]style looks[Enter]Like...[Enter]")
SetStyle(BaseName)
END FUNCTION
```

See also:

CreateStyle DefineStyle ModifyAlignment ModifyBreaks ModifyEffects ModifyFont ModifyLines ModifyReflow ModifySelect ModifySpacing ModifyStyle

This function applies changes to the page layout after the function ModLayoutInit has been run. Choosing this function is equivalent to accepting changes entered by choosing Page/Modify Page Layout.

Do not call this function without calling ModLayoutInit first.

Syntax

ModLayoutFinish()

Return Value

1 (TRUE) if the changes were accepted.

-2 if the changes were not accepted.

Example

```
FUNCTION Example()
ModLayoutInit(512)
ModLayoutLeftFooter(1440, 180, 1440, 720, 1, 1, 1, 0, 1, 0)
ModLayoutLeftHeader(1440, 180, 1440, 720, 1, 1, 1, 0, 1, 0)
ModLayoutLeftLines(255, 7, 1, 9, 1)
ModLayoutLeftPage(1440, 1440, 1440, 1440, 1, 1, 1, 0, 1, 0)
ModLayoutPageSize((11 * 1440), (8.5 * 1440), 1, 1)
ModLayoutRightHeader(1440, 180, 1440, 720, 1, 1, 1, 0, 1, 0)
ModLayoutRightFooter(1440, 180, 1440, 720, 1, 1, 1, 0, 1, 0)
ModLayoutRightLines(255, 7, 1, 9, 1)
ModLayoutRightPage(1440, 1440, 1440, 1440, 1, 1, 1, 0, 1, 0)
ModLayoutFinish()
END FUNCTION
```

See also:

FrameLayout ModLayoutFinish ModLayoutInit ModLayoutLeftFooter ModLayoutLeftHeader ModLayoutLeftLines ModLayoutLeftPage ModLayoutPageSize ModLayoutRightFooter ModLayoutRightHeader ModLayoutRightLines ModLayoutRightPage ModifyStyle

This function prepares Ami Pro to accept page layout changes. Choosing this function is equivalent to initializing changes made when choosing Page/Modify Page Layout.

This function must be called prior to any modify page layout functions.

Syntax

ModLayoutInit(Type)

Type is the type of page setup and can be one or both of the following:

(512) - All pages

(1024) - Mirror image

Return Value

1 (TRUE) if the initialization for changes was successful.

-2 (GeneralFailure) if the initialization failed.

-6 (NoMemory) if the function failed because of insufficient memory.

Example

```
FUNCTION Example()
ModLayoutInit(512)
ModLayoutLeftFooter(1440, 180, 1440, 720, 1, 1, 1, 0, 1, 0)
ModLayoutLeftHeader(1440, 180, 1440, 720, 1, 1, 1, 0, 1, 0)
ModLayoutLeftLines(255, 7, 1, 9, 1)
ModLayoutLeftPage(1440, 1440, 1440, 1440, 1, 1, 1, 0, 1, 0)
ModLayoutPageSize((11 * 1440), (8.5 * 1440), 1, 1)
ModLayoutRightHeader(1440, 180, 1440, 720, 1, 1, 1, 0, 1, 0)
ModLayoutRightFooter(1440, 180, 1440, 720, 1, 1, 1, 0, 1, 0)
ModLayoutRightLines(255, 7, 1, 9, 1)
ModLayoutRightPage(1440, 1440, 1440, 1440, 1, 1, 1, 0, 1, 0)
ModLayoutFinish()
END FUNCTION
```

See also:

FrameLayout ModLayoutFinish ModLayoutInit ModLayoutLeftFooter ModLayoutLeftHeader ModLayoutLeftLines ModLayoutLeftPage ModLayoutPageSize ModLayoutRightFooter ModLayoutRightHeader ModLayoutRightLines ModLayoutRightPage ModifyStyle

This function modifies the left footer information for the current page layout if the current page layout is not for all pages. Choosing this function is equivalent to choosing Page/Modify Page Layout/Footer and choosing Left Pages.

The ModLayoutFinish function must be called after this function to accept the modifications.

Do not call this function without first calling the ModLayoutInit function.

Syntax

ModLayoutLeftFooter(LeftMargin, TopMargin, RightMargin, BottomMargin, Units, Flag, NumCols, Cols, NumTabs[, Type, Offset])

LeftMargin is the distance from the left edge of the paper.

TopMargin is the distance from the top edge of the paper.

RightMargin is the distance from the right edge of the paper.

BottomMargin is the distance from the bottom edge of the paper.

The margin parameters are in twips and represent how much space to leave on that edge of the paper.

Units is the type of measurement and can be one of the following:

Inches (1) - units set to inches
CM (2) - units set to centimeters
Picas (3) - units set to picas
Points (4) - units set to points

The amount of indention must be given in twips (1 inch = 1440 twips). Multiply the desired number of inches by 1440 to determine the value in twips.

Flag is an options parameter and can have one or more of the following values:

(1) - Balance (if multi-column)
(2) - Gutterline (if multi-column)
(4) - Borderline

NumCols is the number of columns for the page.

Cols are pairs of numbers that represent the twip offset to the left and right margin for this column.

NumTabs is how many tabs follow.

You must enter a Type and Offset for each tab.

Type is the type of tab. The Offset parameter must also be used for each tab.

TabLeft (1) - Left tab
TabCenter (2) - Center tab
TabRight (3) - Right tab
TabNumeric (4) - Numeric tab

To display leaders preceding the tab, add one of the following values to the type of tab:

TabHyph (16384) - Displays hyphens
TabDot (-32768) - Displays dots
TabLine (-16384) - Displays a hyphen

Offset is the distance of the tab from the left margin and must be given in twips (1 inch = 1440 twips). The Type parameter must also be used for each tab.

Return Value

1 (TRUE) if the left footer was modified.

-2 (GeneralFailure) if the left footer was not modified.

Example

```
FUNCTION Example()
ModLayoutInit(512)
ModLayoutLeftFooter(1440, 180, 1440, 720, 1, 1, 1, 0, 1, 0)
ModLayoutLeftHeader(1440, 180, 1440, 720, 1, 1, 1, 0, 1, 0)
ModLayoutLeftLines(255, 7, 1, 9, 1)
ModLayoutLeftPage(1440, 1440, 1440, 1440, 1, 1, 1, 0, 1, 0)
ModLayoutPageSize((11 * 1440), (8.5 * 1440), 1, 1)
ModLayoutRightHeader(1440, 180, 1440, 720, 1, 1, 1, 0, 1, 0)
ModLayoutRightFooter(1440, 180, 1440, 720, 1, 1, 1, 0, 1, 0)
ModLayoutRightLines(255, 7, 1, 9, 1)
ModLayoutRightPage(1440, 1440, 1440, 1440, 1, 1, 1, 0, 1, 0)
ModLayoutFinish()
END FUNCTION
```

See also:

FrameLayout ModLayoutFinish ModLayoutInit ModLayoutLeftFooter ModLayoutLeftHeader ModLayoutLeftLines ModLayoutLeftPage ModLayoutPageSize ModLayoutRightFooter ModLayoutRightHeader ModLayoutRightLines ModLayoutRightPage ModifyStyle

This function modifies the left header information for the current page layout if the page setting is not for all pages. Choosing this function is equivalent to choosing Page/Modify Page Layout/Header and choosing Left Pages.

The ModLayoutFinish function must be called after this function to accept the modifications.

Do not call this function without first calling the ModLayoutInit function.

Syntax

ModLayoutLeftHeader(LeftMargin, TopMargin, RightMargin, BottomMargin, Units, Flag, NumCols, Cols, NumTabs[, Type, Offset])

LeftMargin is the distance from the left edge of the paper.

TopMargin is the distance from the top edge of the paper.

RightMargin is the distance from the right edge of the paper.

BottomMargin is the distance from the bottom edge of the paper.

The margin parameters are in twips (1 inch = 1440 twips) and represent how much space to leave on that edge of the paper.

Units is the type of measurement and can be one of the following:

Inches (1) - units set to inches
CM (2) - units set to centimeters
Picas (3) - units set to picas
Points (4) - units set to points

The amount of indention must be given in twips (1 inch = 1440 twips). Multiply the desired number of inches by 1440 to determine the value in twips.

Flag is an options parameter and can have one or more of the following values:

(1) - Balance (if multi-column)
(2) - Gutterline (if multi-column)
(4) - Borderline

NumCols is the number of columns for the page.

Cols are pairs of numbers that represent the twip offset to the left and right margin for this column.

NumTabs is how many tabs follow.

You must enter a Type and Offset for each tab.

Type is the type of tab. The Offset parameter must also be used for each tab.

TabLeft (1) - Left tab
TabCenter (2) - Center tab
TabRight (3) - Right tab
TabNumeric (4) - Numeric tab

To display leaders preceding the tab, add one of the following values to the type of tab:

TabHyph (16384) - Displays hyphens
TabDot (-32768) - Displays dots
TabLine (-16384) - Displays a hyphen

Offset is the distance of the tab from the left margin and must be given in twips (1 inch = 1440 twips). The Type parameter must also be used for each tab.

Return Value

1 (TRUE) if the left header was modified.

-2 (GeneralFailure) if the left header was not modified.

Example

```
FUNCTION Example()
ModLayoutInit(512)
ModLayoutLeftFooter(1440, 180, 1440, 720, 1, 1, 1, 0, 1, 0)
ModLayoutLeftHeader(1440, 180, 1440, 720, 1, 1, 1, 0, 1, 0)
ModLayoutLeftLines(255, 7, 1, 9, 1)
ModLayoutLeftPage(1440, 1440, 1440, 1440, 1, 1, 1, 0, 1, 0)
ModLayoutPageSize((11 * 1440), (8.5 * 1440), 1, 1)
ModLayoutRightHeader(1440, 180, 1440, 720, 1, 1, 1, 0, 1, 0)
ModLayoutRightFooter(1440, 180, 1440, 720, 1, 1, 1, 0, 1, 0)
ModLayoutRightLines(255, 7, 1, 9, 1)
ModLayoutRightPage(1440, 1440, 1440, 1440, 1, 1, 1, 0, 1, 0)
ModLayoutFinish()
END FUNCTION
```

See also:

FrameLayout ModLayoutFinish ModLayoutInit ModLayoutLeftFooter ModLayoutLeftHeader ModLayoutLeftLines ModLayoutLeftPage ModLayoutPageSize ModLayoutRightFooter ModLayoutRightHeader ModLayoutRightLines ModLayoutRightPage ModifyStyle

This function modifies the lines settings for the left page of the current page layout if the current page layout setting is not for all pages. Choosing this function is equivalent to choosing Page/Modify Page Layout/Lines and then choosing Left Pages.

The ModLayoutFinish function must be called after this function to accept the modifications.

Do not call this function without first calling the ModLayoutInit function.

Syntax

ModLayoutLeftLines(GutterShade, GutterStyle, BorderSides, BorderStyle, BorderSpace)

GutterShade is the color of the lines and can be one of the following:

White (16777215) - White
Cyan (16776960) - Light blue
Yellow (65535) - Yellow
Magenta (16711935) - Purple
Green (65280) - Green
Red (255) - Red
Blue (16711680) - Blue
Black (0) - Black
DarkGray (12566463) - 90% gray scale
MediumGray (8355711) - 50% gray scale
LightGray (4144959) - 20% gray scale
VeryLightGray (1644825) - 10% gray scale

GutterStyle defines the width of the line and can be one of the following:

Hairline (1) - Hairline
OnePoint (2) - One point wide
TwoPoint (3) - Two points wide
ThreePoint (4) - Three points wide
FourPoint (5) - Four points wide
FivePoint (6) - Five points wide
SixPoint (7) - Six points wide
DoubleOnePoint (8) - Parallel one point lines
DoubleTwoPoint (9) - Parallel two point lines
ThreeLines (10) - Hairline above and below a two point line
HairBelow (11) - Hairline below a three point line
HairAbove (12) - Hairline above a three point line

BorderSides is one of the following:

(1) - All sides
(2) - Left
(4) - Right
(8) - Top
(16) - Bottom

To set a combination of Left, Right, Top, or Bottom, add them together.

BorderStyle is the width of the border from a range of 1 to 12. See GutterStyle for the values.

BorderSpace is the position of the border with the following values:

(1) - Middle
(2) - Inside

(3) - Outside
(4) - Close inside
(5) - Close to outside

Return Value

1 (TRUE) if the left page line settings were modified.

-2 (GeneralFailure) if the left page line settings were not modified.

Example

```
FUNCTION Example()
ModLayoutInit(512)
ModLayoutLeftFooter(1440, 180, 1440, 720, 1, 1, 1, 0, 1, 0)
ModLayoutLeftHeader(1440, 180, 1440, 720, 1, 1, 1, 0, 1, 0)
ModLayoutLeftLines(255, 7, 1, 9, 1)
ModLayoutLeftPage(1440, 1440, 1440, 1440, 1, 1, 1, 0, 1, 0)
ModLayoutPageSize((11 * 1440), (8.5 * 1440), 1, 1)
ModLayoutRightHeader(1440, 180, 1440, 720, 1, 1, 1, 0, 1, 0)
ModLayoutRightFooter(1440, 180, 1440, 720, 1, 1, 1, 0, 1, 0)
ModLayoutRightLines(255, 7, 1, 9, 1)
ModLayoutRightPage(1440, 1440, 1440, 1440, 1, 1, 1, 0, 1, 0)
ModLayoutFinish()
END FUNCTION
```

See also:

FrameLayout ModLayoutFinish ModLayoutInit ModLayoutLeftFooter ModLayoutLeftHeader ModLayoutLeftLines ModLayoutLeftPage ModLayoutPageSize ModLayoutRightFooter ModLayoutRightHeader ModLayoutRightLines ModLayoutRightPage ModifyStyle

This function modifies the left page information for the current page layout if the page setting is not for all pages. Choosing this function is equivalent to choosing Page/Modify Page Layout/Left Pages and then choosing Left Pages.

The ModLayoutFinish function must be called after this function to accept the modifications.

Do not call this function without first calling the ModLayoutInit function.

Syntax

ModLayoutLeftPage(LeftMargin, TopMargin, RightMargin, BottomMargin, Units, Flag, NumCols, Cols, NumTabs[, Type, Offset])

LeftMargin is the distance from the left edge of the paper.

TopMargin is the distance from the top edge of the paper.

RightMargin is the distance from the right edge of the paper.

BottomMargin is the distance from the bottom edge of the paper.

The margin parameters are in twips (1 inch = 1440 twips) and represent how much space to leave on that edge of the paper.

Units is the type of measurement and can be one of the following:

Inches (1) - units set to inches
CM (2) - units set to centimeters
Picas (3) - units set to picas
Points (4) - units set to points

The amount of indention must be given in twips (1 inch = 1440 twips). Multiply the desired number of inches by 1440 to determine the value in twips.

Flag is an options parameter and can have one or more of the following values:

(1) - Balance (if multi-column)
(2) - Gutterline (if multi-column)
(4) - Borderline

NumCols is the number of columns for the page.

Cols are pairs of numbers that represent the twip offset to the left and right margin for this column.

NumTabs is how many tabs follow.

You must enter a Type and Offset for each tab.

Type is the type of tab. The Offset parameter must also be used for each tab.

TabLeft (1) - Left tab
TabCenter (2) - Center tab
TabRight (3) - Right tab
TabNumeric (4) - Numeric tab

To display leaders preceding the tab, add one of the following values to the type of tab:

TabHyph (16384) - Displays hyphens
TabDot (-32768) - Displays dots
TabLine (-16384) - Displays a hyphen

Offset is the distance of the tab from the left margin and must be given in twips (1 inch = 1440 twips). The Type parameter must also be used for each tab.

Return Value

1 (TRUE) if the left page information was modified.

-2 (GeneralFailure) if the left page information was not modified.

Example

```
FUNCTION Example()
ModLayoutInit(512)
ModLayoutLeftFooter(1440, 180, 1440, 720, 1, 1, 1, 0, 1, 0)
ModLayoutLeftHeader(1440, 180, 1440, 720, 1, 1, 1, 0, 1, 0)
ModLayoutLeftLines(255, 7, 1, 9, 1)
ModLayoutLeftPage(1440, 1440, 1440, 1440, 1, 1, 1, 0, 1, 0)
ModLayoutPageSize((11 * 1440), (8.5 * 1440), 1, 1)
ModLayoutRightHeader(1440, 180, 1440, 720, 1, 1, 1, 0, 1, 0)
ModLayoutRightFooter(1440, 180, 1440, 720, 1, 1, 1, 0, 1, 0)
ModLayoutRightLines(255, 7, 1, 9, 1)
ModLayoutRightPage(1440, 1440, 1440, 1440, 1, 1, 1, 0, 1, 0)
ModLayoutFinish()
END FUNCTION
```

See also:

FrameLayout ModLayoutFinish ModLayoutInit ModLayoutLeftFooter ModLayoutLeftHeader ModLayoutLeftLines ModLayoutLeftPage ModLayoutPageSize ModLayoutRightFooter ModLayoutRightHeader ModLayoutRightLines ModLayoutRightPage ModifyStyle

This function modifies the page size information for the current page layout. Choosing this function is equivalent to choosing Page/Modify Page Layout/Page settings.

The ModLayoutFinish function must be called after this function to accept the modifications.

Do not call this function without first calling the ModLayoutInit function.

Syntax

ModLayoutPageSize(Length, Width, Units, PaperType)

Length is the length of the page in twips (1 inch = 1440 twips), if the paper type is custom**.**

Width is the width of the page in twips (1 inch = 1440 twips), if the paper type is custom.

Units is the type of measurement and can be one of the following:

Inches (1) - units set to inches
CM (2) - units set to centimeters
Picas (3) - units set to picas
Points (4) - units set to points

The amount of indention must be given in twips (1 inch = 1440 twips). Multiply the desired number of inches by 1440 to determine the value in twips.

PaperType is one of the pre-defined paper types and must be one of the following:

(1) - Letter
(2) - Legal
(3) - A3
(4) - A4
(5) - A5
(6) - B5
(7) - Custom

Return Value

1 (TRUE) if the page size information was modified.

-2 (GeneralFailure) if the page size information was not modified.

Example

```
FUNCTION Example()
ModLayoutInit(512)
ModLayoutLeftFooter(1440, 180, 1440, 720, 1, 1, 1, 0, 1, 0)
ModLayoutLeftHeader(1440, 180, 1440, 720, 1, 1, 1, 0, 1, 0)
ModLayoutLeftLines(255, 7, 1, 9, 1)
ModLayoutLeftPage(1440, 1440, 1440, 1440, 1, 1, 1, 0, 1, 0)
ModLayoutPageSize((11 * 1440), (8.5 * 1440), 1, 1)
ModLayoutRightHeader(1440, 180, 1440, 720, 1, 1, 1, 0, 1, 0)
ModLayoutRightFooter(1440, 180, 1440, 720, 1, 1, 1, 0, 1, 0)
ModLayoutRightLines(255, 7, 1, 9, 1)
ModLayoutRightPage(1440, 1440, 1440, 1440, 1, 1, 1, 0, 1, 0)
ModLayoutFinish()
END FUNCTION
```

See also:

FrameLayout ModLayoutFinish ModLayoutInit ModLayoutLeftFooter ModLayoutLeftHeader ModLayoutLeftLines ModLayoutLeftPage ModLayoutPageSize ModLayoutRightFooter ModLayoutRightHeader ModLayoutRightLines ModLayoutRightPage ModifyStyle

This function modifies the footer information for the current page layout if the page setting is for all or right pages. Choosing this function is equivalent to choosing Page/Modify Page Layout/Footer and then choosing All Pages or Right Pages.

The ModLayoutFinish function must be called after this function to accept the modifications.

Do not call this function without first calling the ModLayoutInit function.

Syntax

ModLayoutRightFooter(LeftMargin, TopMargin, RightMargin, BottomMargin, Units, Flag, NumCols, Cols, NumTabs[, Type, Offset])

LeftMargin is the distance from the left edge of the paper.

TopMargin is the distance from the top edge of the paper.

RightMargin is the distance from the right edge of the paper.

BottomMargin is the distance from the bottom edge of the paper.

The margin parameters are in twips (1 inch = 1440 twips) and represent how much space to leave on that edge of the paper.

Units is the type of measurement and can be one of the following:

Inches (1) - units set to inches
CM (2) - units set to centimeters
Picas (3) - units set to picas
Points (4) - units set to points

The amount of indention must be given in twips (1 inch = 1440 twips). Multiply the desired number of inches by 1440 to determine the value in twips.

Flag is an options parameter and can have one or more of the following values:

(1) - Balance (if multi-column)
(2) - Gutterline (if multi-column)
(4) - Borderline

NumCols is the number of columns for the page.

Cols are pairs of numbers that represent the twip offset to the left and right margin for this column.

NumTabs is how many tabs follow.

You must enter a Type and Offset for each tab.

Type is the type of tab. The Offset parameter must also be used for each tab.

TabLeft (1) - Left tab
TabCenter (2) - Center tab
TabRight (3) - Right tab
TabNumeric (4) - Numeric tab

To display leaders preceding the tab, add one of the following values to the type of tab:

TabHyph (16384) - Displays hyphens
TabDot (-32768) - Displays dots
TabLine (-16384) - Displays a hyphen

Offset is the distance of the tab from the left margin and must be given in twips (1 inch = 1440 twips). The Type parameter must also be used for each tab.

Return Value

1 (TRUE) if the right footer was modified.

-2 (GeneralFailure) if the right footer was not modified.

Example

```
FUNCTION Example()
ModLayoutInit(512)
ModLayoutLeftFooter(1440, 180, 1440, 720, 1, 1, 1, 0, 1, 0)
ModLayoutLeftHeader(1440, 180, 1440, 720, 1, 1, 1, 0, 1, 0)
ModLayoutLeftLines(255, 7, 1, 9, 1)
ModLayoutLeftPage(1440, 1440, 1440, 1440, 1, 1, 1, 0, 1, 0)
ModLayoutPageSize((11 * 1440), (8.5 * 1440), 1, 1)
ModLayoutRightHeader(1440, 180, 1440, 720, 1, 1, 1, 0, 1, 0)
ModLayoutRightFooter(1440, 180, 1440, 720, 1, 1, 1, 0, 1, 0)
ModLayoutRightLines(255, 7, 1, 9, 1)
ModLayoutRightPage(1440, 1440, 1440, 1440, 1, 1, 1, 0, 1, 0)
ModLayoutFinish()
END FUNCTION
```

See also:

FrameLayout ModLayoutFinish ModLayoutInit ModLayoutLeftFooter ModLayoutLeftHeader ModLayoutLeftLines ModLayoutLeftPage ModLayoutPageSize ModLayoutRightFooter ModLayoutRightHeader ModLayoutRightLines ModLayoutRightPage ModifyStyle

This function modifies the header information for the current page layout if the page setting is for all or right pages. Choosing this function is equivalent to choosing Page/Modify Page Layout/Header and then choosing All Pages or Right pages.

The ModLayoutFinish function must be called after this function to accept the modifications.

Do not call this function without first calling the ModLayoutInit function.

Syntax

ModLayoutRightHeader(LeftMargin, TopMargin, RightMargin, BottomMargin, Units, Flag, NumCols, Cols, NumTabs, Tabs)

LeftMargin is the distance from the left edge of the paper.

TopMargin is the distance from the top edge of the paper.

RightMargin is the distance from the right edge of the paper.

BottomMargin is the distance from the bottom edge of the paper.

The margin parameters are in twips (1 inch = 1440 twips) and represent how much space to leave on that edge of the paper.

Units is the type of measurement and can be one of the following:

Inches (1) - units set to inches
CM (2) - units set to centimeters
Picas (3) - units set to picas
Points (4) - units set to points

The amount of indention must be given in twips (1 inch = 1440 twips). Multiply the desired number of inches by 1440 to determine the value in twips.

Flag is an options parameter and can have one or more of the following values:

(1) - Balance (if multi-column)
(2) - Gutterline (if multi-column)
(4) - Borderline

NumCols is the number of columns for the page.

Cols are pairs of numbers which represent the twip offset to the left and right margin for this column.

NumTabs is how many tabs follow.

You must enter a Type and Offset for each tab.

Type is the type of tab. The Offset parameter must also be used for each tab.

TabLeft (1) - Left tab
TabCenter (2) - Center tab
TabRight (3) - Right tab
TabNumeric (4) - Numeric tab

To display leaders preceding the tab, add one of the following values to the type of tab:

TabHyph (16384) - Displays hyphens
TabDot (-32768) - Displays dots
TabLine (-16384) - Displays a hyphen

Offset is the distance of the tab from the left margin and must be given in twips (1 inch = 1440 twips). The Type parameter must also be used for each tab.

Return Value

1 (TRUE) if the right header was modified.

-2 (GeneralFailure) if the right header was not modified.

Example

```
FUNCTION Example()
ModLayoutInit(512)
ModLayoutLeftFooter(1440, 180, 1440, 720, 1, 1, 1, 0, 1, 0)
ModLayoutLeftHeader(1440, 180, 1440, 720, 1, 1, 1, 0, 1, 0)
ModLayoutLeftLines(255, 7, 1, 9, 1)
ModLayoutLeftPage(1440, 1440, 1440, 1440, 1, 1, 1, 0, 1, 0)
ModLayoutPageSize((11 * 1440), (8.5 * 1440), 1, 1)
ModLayoutRightHeader(1440, 180, 1440, 720, 1, 1, 1, 0, 1, 0)
ModLayoutRightFooter(1440, 180, 1440, 720, 1, 1, 1, 0, 1, 0)
ModLayoutRightLines(255, 7, 1, 9, 1)
ModLayoutRightPage(1440, 1440, 1440, 1440, 1, 1, 1, 0, 1, 0)
ModLayoutFinish()
END FUNCTION
```

See also:

FrameLayout ModLayoutFinish ModLayoutInit ModLayoutLeftFooter ModLayoutLeftHeader ModLayoutLeftLines ModLayoutLeftPage ModLayoutPageSize ModLayoutRightFooter ModLayoutRightHeader ModLayoutRightLines ModLayoutRightPage ModifyStyle

This function modifies the lines settings for the current page layout if the page setting is for all or right pages. Choosing this function is equivalent to choosing Page/Modify Page Layout/Lines and then choosing All Pages or Right Pages.

The ModLayoutFinish function must be called after this function to accept the modifications.

Do not call this function without first calling the ModLayoutInit function.

Syntax

ModLayoutRightLines(GutterShade, GutterStyle, BorderSides, BorderStyle, BorderSpace)

GutterShade is the color of the lines and can be one of the following:

- White (16777215) - White
- Cyan (16776960) - Light blue
- Yellow (65535) - Yellow
- Magenta (16711935) - Purple
- Green (65280) - Green
- Red (255) - Red
- Blue (16711680) - Blue
- Black (0) - Black
- DarkGray (12566463) - 90% gray scale
- MediumGray (8355711) - 50% gray scale
- LightGray (4144959) - 20% gray scale
- VeryLightGray (1644825) - 10% gray scale

GutterStyle defines the width of the line and can be one of the following:

- Hairline (1) - Hairline
- OnePoint (2) - One point wide
- TwoPoint (3) - Two points wide
- ThreePoint (4) - Three points wide
- FourPoint (5) - Four points wide
- FivePoint (6) - Five points wide
- SixPoint (7) - Six points wide
- DoubleOnePoint (8) - Parallel one point lines
- DoubleTwoPoint (9) - Parallel two point lines
- ThreeLines (10) - Hairline above and below a two point line
- HairBelow (11) - Hairline below a three point line
- HairAbove (12) - Hairline above a three point line

BorderSides is one of the following:

- (1) - All sides, or a combination of:
- (2) - Left
- (4) - Right
- (8) - Top
- (16) - Bottom

To set a combination of Left, Right, Top, or Bottom, add them together.

BorderStyle is the width of the border from a range of 1 to 12. See GutterStyle for the values.

BorderSpace is the position of the border with the following values:

- (1) - Middle
- (2) - Inside

(3) - Outside
(4) - Close inside
(5) - Close to outside

Return Value

1 (TRUE) if the right page line settings were modified.

-2 (GeneralFailure) if the right page line settings were not modified.

Example

```
FUNCTION Example()
ModLayoutInit(512)
ModLayoutLeftFooter(1440, 180, 1440, 720, 1, 1, 1, 0, 1, 0)
ModLayoutLeftHeader(1440, 180, 1440, 720, 1, 1, 1, 0, 1, 0)
ModLayoutLeftLines(255, 7, 1, 9, 1)
ModLayoutLeftPage(1440, 1440, 1440, 1440, 1, 1, 1, 0, 1, 0)
ModLayoutPageSize((11 * 1440), (8.5 * 1440), 1, 1)
ModLayoutRightHeader(1440, 180, 1440, 720, 1, 1, 1, 0, 1, 0)
ModLayoutRightFooter(1440, 180, 1440, 720, 1, 1, 1, 0, 1, 0)
ModLayoutRightLines(255, 7, 1, 9, 1)
ModLayoutRightPage(1440, 1440, 1440, 1440, 1, 1, 1, 0, 1, 0)
ModLayoutFinish()
END FUNCTION
```

See also:

FrameLayout ModLayoutFinish ModLayoutInit ModLayoutLeftFooter ModLayoutLeftHeader ModLayoutLeftLines ModLayoutLeftPage ModLayoutPageSize ModLayoutRightFooter ModLayoutRightHeader ModLayoutRightLines ModLayoutRightPage ModifyStyle

This function modifies the page information for the current page layout if the page setting is for all or right pages. Choosing this function is equivalent to choosing Page/Modify Page Layout/Right Pages and then choosing All Pages or Right Pages.

The ModLayoutFinish function must be called after this function to accept the modifications.

Do not call this function without first calling the ModLayoutInit function.

Syntax

ModLayoutRightPage(LeftMargin, TopMargin, RightMargin, BottomMargin, Units, FaceFlag, NumCols, Cols, NumTabs[, Type, Offset])

LeftMargin is the distance from the left edge of the paper.

TopMargin is the distance from the top edge of the paper.

RightMargin is the distance from the right edge of the paper.

BottomMargin is the distance from the bottom edge of the paper.

The margin parameters are in twips (1 inch = 1440 twips) and represent how much space to leave on that edge of the paper.

Units is the type of measurement and can be one of the following:

Inches (1) - units set to inches
CM (2) - units set to centimeters
Picas (3) - units set to picas
Points (4) - units set to points

The amount of indention must be given in twips (1 inch = 1440 twips). Multiply the desired number of inches by 1440 to determine the value in twips.

FaceFlag is an options parameter and can have one or more of the following values:

(1) - Balance (if multi-column)
(2) - Gutterline (if multi-column)
(4) - Borderline

NumCols is the number of columns for the page.

Cols are pairs of numbers that represent the twip offset to the left and right margin for this column.

NumTabs is how many tabs follow.

You must enter a Type and Offset for each tab.

Type is the type of tab. The Offset parameter must also be used for each tab.

TabLeft (1) - Left tab
TabCenter (2) - Center tab
TabRight (3) - Right tab
TabNumeric (4) - Numeric tab

To display leaders preceding the tab, add one of the following values to the type of tab:

TabHyph (16384) - Displays hyphens
TabDot (-32768) - Displays dots
TabLine (-16384) - Displays a hyphen

Offset is the distance of the tab from the left margin and must be given in twips (1 inch = 1440 twips). The Type parameter must also be used for each tab.

Return Value

1 (TRUE) if the right page information was modified.

-2 (GeneralFailure) if the right page information was not modified.

Example

```
FUNCTION Example()
ModLayoutInit(512)
ModLayoutLeftFooter(1440, 180, 1440, 720, 1, 1, 1, 0, 1, 0)
ModLayoutLeftHeader(1440, 180, 1440, 720, 1, 1, 1, 0, 1, 0)
ModLayoutLeftLines(255, 7, 1, 9, 1)
ModLayoutLeftPage(1440, 1440, 1440, 1440, 1, 1, 1, 0, 1, 0)
ModLayoutPageSize((11 * 1440), (8.5 * 1440), 1, 1)
ModLayoutRightHeader(1440, 180, 1440, 720, 1, 1, 1, 0, 1, 0)
ModLayoutRightFooter(1440, 180, 1440, 720, 1, 1, 1, 0, 1, 0)
ModLayoutRightLines(255, 7, 1, 9, 1)
ModLayoutRightPage(1440, 1440, 1440, 1440, 1, 1, 1, 0, 1, 0)
ModLayoutFinish()
END FUNCTION
```

See also:

FrameLayout ModLayoutFinish ModLayoutInit ModLayoutLeftFooter ModLayoutLeftHeader ModLayoutLeftLines ModLayoutLeftPage ModLayoutPageSize ModLayoutRightFooter ModLayoutRightHeader ModLayoutRightLines ModLayoutRightPage ModifyStyle

This function sets a macro function to be called if the user clicks the mouse while the current macro is running. This function is passed the window handle (HWND), a flag, and x and y coordinates. HWND is the handle of the selected window, and flag is a combination of the following bits:

(1) - Left mouse button is down.
(2) - Right mouse button is down.
(4) - Shift key is down.
(8) - Control key is down.

X and Y are relative to upper left corner of HWND and are in device units.

Syntax

MouseInterrupt(Function)

Function is the function that is run when a mouse button is pressed. This parameter may contain the macro filename or the function within that file to call.

Return Value

The previously set MouseInterrupt function.

Example

```
FUNCTION Example()
MouseInterrupt("MouseInt")   ' Set them
TYPE("Click the mouse")
FOR i = 1 to 5  ' Kill some time.
    FOR j = 1 to 100
    NEXT
    TYPE("{i}.")
NEXT
END FUNCTION

' The Interrupt function: just echo the input parameters

FUNCTION MouseInt(hwnd, flag, x, y)
Message("hwnd {hwnd}, flag {flag}, {x}, {y}")
END FUNCTION
```

See Also:

KeyInterrupt DlgKeyInterrupt

This function promotes the current paragraph to a higher outline level in Outline Mode or moves cells in a table left one column. In a table, you can move two or more adjacent selected cells or the column in which the cursor is located to the left one column. Keyboard commands must be used to record this function.

Syntax

MoveLeftOrPromote()

Return Value

1 (TRUE) if the cells move or the outline paragraph style is promoted.

0 (NoAction) if no action is taken.

Example

```
FUNCTION Example()
MoveLeftOrPromote()
END FUNCTION
```

See also:

MoveParagraphDown MoveParagraphUp MoveRightOrDemote

This function exchanges the current paragraph with the next paragraph or moves cells in a table down one row. In a table, you can move two or more adjacent selected cells or the column in which the cursor is located to the left one column. Keyboard commands must be used to record this function.

Syntax

MoveParagraphDown()

Return Value

1 (TRUE) if the cells move or the current paragraph style is exchanged with the next paragraph.

0 (NoAction) if no action is taken.

Example

```
FUNCTION Example()
MoveParagraphDown()
END FUNCTION
```

See also:

MoveLeftOrPromote MoveParagraphUp MoveRightOrDemote

This function exchanges the current paragraph with the previous paragraph or moves cells in a table up one row. In a table, you can move two or more adjacent selected cells or the entire row in which the cursor is located up one row. Keyboard commands must be used to record this function.

Syntax

MoveParagraphUp()

Return Value

1 (TRUE) if the cells move or the current paragraph style is exchanged with the previous paragraph.

0 (NoAction) if no action is taken.

Example

```
FUNCTION Example()
MoveParagraphUp()
END FUNCTION
```

See also:

MoveLeftOrPromote MoveParagraphDown MoveRightOrDemote

This function demotes the current paragraph to a lower outline level in Outline Mode or moves cells in a table right one column. In a table, you can move two or more adjacent selected cells or the entire column in which the cursor is located to the right one column. Keyboard commands must be used to record this function.

Syntax

MoveRightOrDemote()

Return Value

1 (TRUE) if the cells move or the outline paragraph style is demoted.

0 (NoAction) if no action is taken.

Example

```
FUNCTION Example()
MoveRightOrDemote()
END FUNCTION
```

See also:

MoveLeftOrPromote MoveParagraphDown MoveParagraphUp

This function displays a Windows Message box with the specified title, prompt, icon, and push buttons. It waits for the user to select a push button and then returns the ID of the button the user pressed.

Modified in 3.0 There is a new parameter, Title, in Ami Pro release 3.0.

Syntax

MultiDecide(Prompt, Type[, Title])

prompt is a string used as a prompt to the user. It can be a maximum of 80 characters.

type is a number that defines the push buttons offered to the user and the icon displayed in the dialog box. It can be one of the following button types and one of the icon types added together.

The choices for buttons are:

OKButton (0) - OK button.
OKCancelButton (1) - OK and Cancel buttons
AbortIgnoreRetryButton (2) - Abort, Ignore and Retry buttons
YesNoCancelButton (3) - Yes, No, and Cancel buttons
YesNoButton (4) - Yes and No buttons
RetryCancelButton (5) - Retry and Cancel buttons

The choices for icons are:

HandIcon (16) - The hand or stop sign
QuestionIcon (32) - Question mark
ExclamationIcon (48) - Exclamation point
AsteriskIcon (64) - Asterisk

Title is the title for the message box. The default title is "Ami Pro Macro".

Return Value

This function returns a number representing the button pressed:

OK (1) - OK button
Cancel (2) - Cancel button
Abort (3) - Abort button
Retry (4) - Retry button
Ignore (5) - Ignore button
Yes (6) - Yes button
No (7) - No button

Example

```
FUNCTION Example()
Answer = MultiDecide("Format your hard drive now?", (YesNoButton + HandIcon), "Just Kidding")
IF Answer = 6
    message("Wow.  You're pretty trusting.")
ELSEIF Answer = 7
    message("Rats.")
ENDIF
END FUNCTION
```

See also:

Decide DialogBox Message Query$ UserControl

This function displays the New dialog box. Choosing this function is equivalent to choosing File/New.

Modified in 3.0 There are two new values, 512 and 1024, in Ami Pro release 3.0.

Syntax

New(Style, WithContents, Options)

Style is the style sheet file to use with the new file.

WithContents determines if the new file is created with the paragraph style sheet contents. It can be one of the following:

NoContents (0) - Do not bring in contents of paragraph style sheet

WithContents (1) - Bring in contents with paragraph style sheet

Options determines if the new file is placed in a new Multiple Document Interface (MDI) window or replaces the currently active document. It can be one of the following values:

(0) - Open another window for the new file

(128) - Close the current document

(512) - Do not run the macro associated with the style sheet, if one exists

(1024) - Show the style sheets by description rather than file name when the New dialog box displays

If the ReplaceCurrent option is used and a document is already open, then the new file replaces the current document in the selected window. If the ReplaceCurrent option is set, then any documents opened thereafter have this option set. To display the New dialog box and allow the user to choose the style sheet file and with contents options: **New**

Return Value

1 (TRUE) if the new file was created.

0 (UserCancel) if the user canceled the function.

-2 (GeneralFailure) if the file was not created.

Example

```
FUNCTION Example()
New("_MACRO.STY", 1, 0)
END FUNCTION
```

See also:

FileOpen GetOpenFileName$

This function creates a new window with the same name as the active window. If the active window contains a saved Ami Pro file, the new window contains a read-only copy of that file. Choosing this function is equivalent to choosing Window/New Window.

Syntax

NewWindow()

Return Value

1 (TRUE) if the new window was created.

-2 (GeneralFailure) if the new window was not created.

Example

```
FUNCTION Example()
thisfile = GetOpenFileName$()
Message("Opening another window for {thisfile}.")
NewWindow()
END FUNCTION
```

See also:

CascadeWindow NextWindow TileWindow SelectWindow

This function changes the active document window to the next document window. Choosing this function is equivalent to pressing **CTRL+TAB**.

Syntax

NextWindow()

Return Value

This function does not return a value.

Example

```
FUNCTION Example()
NextWindow()
END FUNCTION
```

See also:

CascadeWindow NewWindow TileWindow SelectWindow

This function sets the no hyphenation attribute for selected text or for all following text if no text is selected. It acts as a toggle, turning off the attribute if it is currently on or turning on the attribute if it is currently off. Choosing this function is equivalent to choosing Edit/Mark Text/No Hyphenation.

Syntax

NoHyphenation()

Return Value

1 (TRUE) if the change of hyphenation was successful.

-2 (GeneralFailure) if the change of hyphenation was not successful.

Example

```
FUNCTION Example()
NoHyphenation()
END FUNCTION
```

See also:

ProtectedText

This function removes attributes for selected text or for all following text if no text is selected. Choosing this function is equivalent to choosing Text/Normal.

Syntax

NormalText()

Return Value

1 (TRUE) if the attributes were removed from the text.

-6 (NoMemory) if the function failed because of insufficient memory.

Example

```
FUNCTION Example()
String = "This is a line of text."
Bold(1)
Italic(1)
TYPE("{String}")
NormalText()
TYPE("{String}")
END FUNCTION
```

See also:

Bold Italic Underline WordUnderline

This function opens a note located at the insertion point and allows the user to edit it. It is the equivalent of double-clicking on a note. This function does not automatically insert or delete notes.

To set the user initials for notes and the default note color, refer to the UserSetup function.

Syntax

Notes()

Return Value

1 (TRUE) if the notes function was completed.

-2 (GeneralFailure) if the notes function was not completed.

-6 (NoMemory) if the function failed because of insufficient memory.

Example

```
FUNCTION Example()
Notes()
END FUNCTION
```

See also:

InsertNote UserSetUp

This function returns the number of seconds that have elapsed since midnight on January 1, 1970.

Syntax

Now()

Return Value

A number corresponding to the number of seconds that have elapsed since midnight on January 1, 1970.

Example

```
FUNCTION Example()
Born = Query$("What is your Birthday (MM/DD/YYYY)?")
Date = FormatDate$(Now(), "h")
Time = FormatTime$(Now(), 6)
Days = DateDiff(Born, Date)
TextDate = FormatDate$(Now(), "d")
Message("It is now {Time} on {TextDate}. You are {Days} days old.")
END FUNCTION
```

See also:

InsertDate GetTime FormatDate$ FormatTime$

This function is used to dimension arrays for the NWGetContainerNames function.

Syntax

NWGetContainerCount(Object)

Object is the NewWave name for a file and its application. It can be up to 32 characters in length.

Return Value

The number of containers.

See also:

NWGetContainerNames

This function loads the passed arrays with container names and references.

Syntax

NWGetContainerNames(Object, &Names, &Ref)

Object is the NewWave name for a file and its application. It can be up to 32 characters in length.

&Names is the container names for the object.

&Ref is the reference number receiving the parent's reference number.

Return Value

The container names in the names array and their reference numbers in the Ref array.

See also:

NWGetContainerCount

This function returns the current container name and reference.

Syntax

NWGetCurrentContainer(&Object, &Ref)

&Object is the object receiving the parent's name.

&Ref is the reference number receiving the parent's reference number.

Return Value

The current container name and reference.

See also:

CreateANew ImportText IsNewWave ListObjects NWGetContainerCount NWGetContainerNames NWGetCurrentObject$ NWGetObjectCount NWGetObjectNames NWGetParent NWReferenceToFile$ ObjectAttributes OpenObject SaveAsMaster SaveAsObject Share ShowLinks

This function returns the current object name.

Syntax

NWGetCurrentObject$()

Return Value

The current object name.

See also:

CreateANew ImportText IsNewWave ListObjects NWGetContainerCount NWGetContainerNames NWGetCurrentContainer NWGetObjectCount NWGetObjectNames NWGetParent NWReferenceToFile$ ObjectAttributes OpenObject SaveAsMaster SaveAsObject Share ShowLinks

This function refers to a container and returns the number of objects in it. It is normally used to allocate arrays for NWGetObjectNames.

Syntax

NWGetObjectCount(Object)

Object is the NewWave name for a file and its application. It can be up to 32 characters in length.

Return Value

The number of objects in the container.

See also:

NWGetObjectNames CreateANew ImportText IsNewWave ListObjects NWGetContainerCount NWGetContainerNames NWGetCurrentContainer NWGetCurrentObject$ NWGetParent NWReferenceToFile$ ObjectAttributes OpenObject SaveAsMaster SaveAsObject Share ShowLinks

This function loads the two arrays with the list of object names and reference numbers.

Syntax

NWGetObjectNames(Object, &ObjectArray, &RefArray)

Object is the NewWave name for a file and its application. It can be up to 32 characters in length.

&ObjectArray is the object names.

&RefArray is reference numbers.

Return Value

The number of objects in the container.

See also:

CreateANew ImportText IsNewWave ListObjects NWGetContainerCount NWGetContainerNames NWGetCurrentContainer NWGetCurrentObject$ NWGetObjectCount NWGetParent NWReferenceToFile$ ObjectAttributes OpenObject SaveAsMaster SaveAsObject Share ShowLinks

This function returns the parent's name and reference number.

Syntax

NWGetParent(Ref, &Object, &Ref)

Ref is the reference number of an object.

&Obj is the object receiving the parent's name.

&Ref is the reference number receiving the parent's reference number.

Return Value

The parent's name and reference.

See also:

CreateANew ImportText IsNewWave ListObjects NWGetContainerCount NWGetContainerNames NWGetCurrentContainer NWGetCurrentObject$ NWGetObjectCount NWGetObjectNames NWReferenceToFile$ ObjectAttributes OpenObject SaveAsMaster SaveAsObject Share ShowLinks

This function returns the actual DOS file name from the reference number.

Syntax

NWReferenceToFile$(Ref)

Ref is the reference number of an object.

Return Value

The DOS filename for the object.

See also:

CreateANew ImportText IsNewWave ListObjects NWGetContainerCount NWGetContainerNames NWGetCurrentContainer NWGetCurrentObject$ NWGetObjectCount NWGetObjectNames NWGetParent ObjectAttributes OpenObject SaveAsMaster SaveAsObject Share ShowLinks

This function changes the attributes for an object. Choosing this function is equivalent to choosing either Action/Doc Info/Attributes (for the open Ami Pro object) or Objects/Attributes of Object (for the selected object inside an Ami Pro object).

Syntax

ObjectAttributes(Which, Options, ObjectTitle, Comments)

Which is one of the following values:

(0) - The open Ami Pro object

(1) - The selected object in an Ami Pro frame

Options is one of the following values:

(1) - Turn on autoshare

(2) - Turn off autoshare

(4) - Change the title of the object

(8) - Change the comments

(16) - Change autoshare (use if turn autoshare is on or off)

Values may be combined.

ObjectTitle is the title of the object.

Comments is the comments string.

To display the Attributes dialog box and allow the user to set the parameters: **ObjectAttributes**

Return Value

This function does not return a value.

Example

```
FUNCTION Example()
ObjectAttributes(0 4 "memo2" "") 'change the object name to memo2
END FUNCTION
```

See also:

CreateANew ImportText IsNewWave ListObjects NWGetContainerCount NWGetContainerNames NWGetCurrentContainer NWGetCurrentObject$ NWGetObjectCount NWGetObjectNames NWGetParent NWReferenceToFile$ OpenObject SaveAsMaster SaveAsObject Share ShowLinks

This function sets an accelerator key when you specify an Ami Pro function or a macro name. Any accelerator key currently set to the specified keystroke is reassigned to the new function or macro. Choosing this function is equivalent to choosing Tools/Macros/Edit and then entering keys in the Playback shortcut keys text box. To reassign the keystroke to its original state, make the second parameter a 0 and the third parameter the null string ("").

Syntax

OnKey(Key, Function, MacroName)

Key is the keystroke, enclosed in square brackets, to use as an accelerator key, in square brackets.

Function is the Ami Pro function this keystroke should execute. If a macro is to be run, this parameter should be the null string(""). Do not include parentheses around the function.

MacroName is the name of the macro to be run. If an Ami Pro function is to be executed, this parameter should be the null string ("").

Return Value

1 (TRUE) if the accelerator key was assigned.

0 (UserCancel) if the user canceled the function.

-2 (GeneralFailure) if the accelerator key could not be assigned.

Example

```
FUNCTION Example()
OnKey([ctrlshifts], SaveAs, "")
END FUNCTION
```

See also:

ChangeShortcutKey

This function sets a permanent callback function that is called whenever focus changes from one Multiple Document Interface (MDI) document to another.

Syntax

OnMDIActivate(MacroName[!Function])

MacroName is the name of the macro to run when the focus changes from one Ami Pro document to another using the MDI interface. This parameter may contain the macro filename and the **Function** to call. At a minimum, this parameter must contain the macro file name.

Use the null string("") as the MacroName to disable this function.

Return Value

The previously set OnMDIActivate callback function.

Example

```
FUNCTION Example()
Mac = GetRunningMacroFile$()
OnMDIActivate("{Mac}!Changed")   ' Note: filename is required here
END FUNCTION

FUNCTION Changed()
Message(GetOpenFileName$())   ' Just show the new filename
StatusBarMsg("")
END FUNCTION
```

See also:

NextWindow RunLater

This function opens a merge data file. It does not edit the data file automatically.

Syntax

OpenDataFile(FileName, App)

FileName is the name of the document to open. If the file to be opened is not in the current directory or document directory, the full path must be used.

App is the Ami Pro file type. To open an Ami Pro file, use the null string ("").

Return Value

This function returns 1.

Example

```
FUNCTION Example()
OpenDataFile("senators.sam","")
END FUNCTION
```

See also:

CreateDataFile MergeToFile OpenMergeFile

This function opens a merge document file. The user can insert additional field names into the document, continue the merge, or select a data file.

Syntax

OpenMergeFile(FileName, Options, App)

FileName is the name of the document to open. If the file to be opened is not in the current directory or document directory, the full path must be used. To open the 'Untitled' file, use the null string ("") as the file name.

Options is a number corresponding to the options for the file you open. To use both options, add them together.

(1) - Required

(128) - Close the current file (must combine with 1)

App is the Ami Pro file type. To open an Ami Pro file, use the null string ("").

Return Value

This function returns 1.

Example

```
FUNCTION Example()
OpenMergeFile("Merge.sam", 1, "")
END FUNCTION
```

See also:

CreateDataFile MergeToFile OpenDataFile

This function opens a NewWave object in the selected frame. Choosing this function is equivalent to choosing Objects/Open.

A frame containing a NewWave object must be selected.

Syntax

OpenObject()

Return Value

1 (TRUE) if the object was opened.

-2 (GeneralFailure) if the object could not be opened.

Example

```
FUNCTION Example()
TYPE ("[CtrlHome]") 'go to start of doc
GoToCmd(4 2 1) 'go to first frame
OpenObject()
END FUNCTION
```

See also:

CreateANew ImportText IsNewWave ListObjects NWGetContainerCount NWGetContainerNames NWGetCurrentContainer NWGetCurrentObject$ NWGetObjectCount NWGetObjectNames NWGetParent NWReferenceToFile$ ObjectAttributes SaveAsMaster SaveAsObject Share ShowLinks

These functions open one of the last five previously opened Ami Pro files. The function OpenPreviousFile1 is the most recently opened file and OpenPreviousFile5 is the oldest file. Choosing these functions are equivalent to choosing File/#, where # is the number to the left of the file name to open at the bottom of the File menu.

The number of recently open files to display may be set using the UserSetup function.

Syntax

OpenPreviousFile1()

OpenPreviousFile2()

OpenPreviousFile3()

OpenPreviousFile4()

OpenPreviousFile5()

Return Value

0 (FALSE) if the user cancels the function or if no action is taken.

1 (TRUE) if the file is opened.

-2 (GeneralFailure) if the file is not opened.

Example

```
FUNCTION Example()
OpenPreviousFile1()
END FUNCTION
```

See also:

UserSetup

This function changes the number of displayed outline levels. The number of levels set are the number of levels printed if the document is printed. Choosing this function is equivalent to clicking the icon for the number of outline levels while in Outline Mode.

You must be in Outline Mode to use this function.

Syntax

OutlineLevels(Levels)

Levels is the number of outline levels to display.

Return Value

This function returns 1.

Example

```
FUNCTION Example()
Levels = Query$("Levels?")
OutlineLevels(Levels)
END FUNCTION
```

See also:

OutlineStyle

This function changes the current Multiple Document Interface (MDI) document in Ami Pro from Draft Mode or Layout Mode to Outline Mode. Choosing this function is equivalent to choosing View/Outline Mode.

Syntax

OutlineMode()

Return Value

1 (TRUE) if the view mode was changed.

0 (NoAction) if no action was taken because Ami Pro is already in the mode selected.

Example

```
FUNCTION Example()
OutlineMode()
END FUNCTION
```

See also:

DraftMode LayoutMode EnlargedView FacingView FullPageView GetMode GetViewLevel StandardView OutlineMode

This function assigns outline styles and options to paragraph styles. Choosing this function is equivalent to choosing Style/Outline Styles.

Modified in 3.0 There are two new values, #6 and #7, for BulletText in Ami Pro release 3.0.

Syntax

OutlineStyle(Count[, Style, Level, Options, BulletText])

Count is the number of levels to which you assign styles.

The remaining parameters depend on Count. An entire set of the optional parameters must be present for the number of levels defined by the Count parameter.

Style is the paragraph to which you want to assign a style.

Level is the outline level in which the style is assigned.

Options is the reset options for each outline level. It is one of the following values:

(0) - Do not reset
(2) - Reset after a higher level
(4) - Reset after an intervening style
(8) - Turn on cumulative numbering

The Reset after a higher level and Reset after an intervening style options cannot be used together. However, the cumulative numbering value can be combined with either of the other values.

BulletText is the leading text for the paragraph. The BulletText parameter specifies the text, bullets, or numbers which must precede each paragraph using this paragraph style. The text must be typed as it would appear in the edit box in the Style/Modify Style/Bullets & numbers dialog box. Bullets can be inserted by typing the following text where the bullet should be placed:

<·1> - Small Round Bullet
<·2> - Large Round Bullet
<·3> - Small Square Bullet
<·4> - Large Square Bullet
<·5> - Large Outline Square Bullet
<·6> - Small Diamond Bullet
<·7> - Large Diamond Bullet
<·8> - Small Open Circle Bullet
<·9> - Large Open Circle Bullet
<·10> - Check Mark
<·11> - Tack
<·12> - Square shadow below bullet
<·13> - Square shadow above bullet
<·14> - Check box
<·15> - Square with X bullet
<·16> - Rounded arrowhead top shaded
<·17> - Rounded arrowhead bottom shaded

To type the dot in front of the bullet number, turn on Num Lock, hold the ALT key, and type 0183, and release the ALT key. Numbers can be inserted by typing the following text where the number should be inserted:

<#1> - 1 2 3 4 5...
<#2> - A B C D E...
<#3> - a b c d e...
<#4> - I II III IV V...

<#5> - i ii iii iv v...
<#6> - * ** *** ****...
<#7> - (dagger characters)...

Return Value

1 (TRUE) if the outline style is created.

0 (UserCancel) if the user canceled the function.

Example

```
FUNCTION Example()
OutlineStyle(1, "Body Text", 1, 4, "<·17>")
END FUNCTION
```

See also:

ModifyStyle

This function inserts or removes a page break at the insertion point. Choosing this function is equivalent to choosing Page/Breaks.

Syntax

PageBreak(Function)

Function is the page break function desired. The Function parameter defines the page break function to use, according to the following list:

InsBreak (1) - Insert page break
DelBreak (2) - Delete page break
CenterBreak (3) - Insert vertically centered page break
InsColBreak (4) - Insert column break
DelColBreak (5) - Delete column break

To display the Page Breaks dialog box and allow the user to choose the page break options: **PageBreak**

Return Value

1 (TRUE) if the page break was inserted or removed.

0 (UserCancel) if the user canceled the function.

-2 (GeneralFailure) if the break was not inserted or removed.

Example

```
FUNCTION Example()
PageBreak(1)
END FUNCTION
```

See also:

InsertLayout

This function moves the insertion point down one page. Choosing this function is equivalent to clicking the page down icon on the status bar.

Syntax

PageDown()

Return Value

This function returns 1.

Example

```
FUNCTION Example()
PageDown()
END FUNCTION
```

See also:

PageUp TYPE

This function is equivalent to choosing Page/Page Numbering.

Syntax

PageNumber(Text, StartOn, StartNum, Style)

Text is the leading text to use for the page number. If no leading text is used, this parameter should be the null string ("").

StartOn is the page number to start the numbering on.

StartNum is the number to use for the first numbered page.

Style is the numbering style to use for the number and can be one of the following:

(1) - 1, 2, 3, 4, 5
(I) - I, II, III, IV, V, VI
(i) - i, ii, iii, iv, v, vi
(A) - A, B, C, D, E, F
(a) - a, b, c, d, e, f

To display the Page Numbering dialog box and allow the user to choose the page number: **PageNumber**

Return Value

1 (TRUE) if the page number was inserted.

0 (UserCancel) if the user canceled the function.

-2 (GeneralFailure) if the number was not inserted.

Example

```
FUNCTION Example()
PageNumber("Page", 1, 1, 1)
END FUNCTION
```

See also:

LineNumber

This function moves the insertion point up one page. Choosing this function is equivalent to clicking the page up icon on the status bar.

Syntax

PageUp()

Return Value

This function returns 1.

Example

```
FUNCTION Example()
PageUp()
END FUNCTION
```

See also:

PageDown TYPE

This function pastes the contents of the clipboard at the location of the insertion point. Choosing this function is equivalent to choosing Edit/Paste.

Syntax

Paste()

Return Value

1 (TRUE) if the text was pasted.

-2 (GeneralFailure) if the text could not be pasted or if there was no text to paste.

Example

```
FUNCTION Example()
Stuff = Query$("Enter what you want put on the clipboard:")
ClipboardWrite(Stuff, 1)
CALL Example2()
END FUNCTION

FUNCTION Example2()
Paste()
TYPE("[Enter]Should look exactly like the line below:[Enter]")
TYPE(ClipboardRead(1))
END FUNCTION
```

See also:

Copy Cut CurChar$ CurWord$ CurShade$ ClipboardWrite ClipboardRead

This function allows the macro to pause for a set period of time before continuing.

Syntax

Pause(Time)

Time is the amount of time to pause, in tenths of seconds.

Return Value

This function does not return a value.

Example

```
FUNCTION Example()
TYPE("Hello...")
PAUSE(030)
TYPE("There!")
END FUNCTION
```

See also:

HourGlass RunLater GetTime

This function returns the logical page number in a master document, given the physical page number.

Syntax

PhysicalToLogical(Page)

Page is the page number that appears when the file is opened for viewing.

Return Value

The logical page number (the number that actually prints in a master document print).

0 (UserCancel) if the user canceled the function.

Example

```
FUNCTION Example()
MacFile = GetRunningMacroFile$()
Count = GetBookMarkCount()
IF Count > 0
    DIM Bookmarks(Count)
    GetBookMarkNames(&Bookmarks)
    DeleteMenu(1, "&Bookmarks")
    AddMenu(1, "&Bookmarks")
    FOR I = 1 to Count
        ThisBookmark = Bookmarks(I)
        AddMenuItem(1, "&Bookmarks", ThisBookmark, "{MacFile}!Example2({ThisBookmark})",
ThisBookmark)
    NEXT
ELSE
    Message("No bookmarks in this document!")
ENDIF
END FUNCTION

FUNCTION Example2(Bkmk)
MarkBookMark(Bkmk, FindBookmark)
PhysicalPage = GetBookMarkPage(Bkmk)
LogicalPage = PhysicalToLogical(GetBookMarkPage(Bkmk))
Message("{Bkmk} is on physical page {PhysicalPage} and logical page {LogicalPage}.")
END FUNCTION
```

See also:

GetBookMarkPage GetFmtPageStr$

This function displays the Print Envelope dialog box. This function does not automatically print the envelope. Choosing this function is equivalent to choosing File/Print Envelope.

Syntax

PrintEnvelope()

Return Value

This function returns 1.

Example

```
FUNCTION Example()
PrintEnvelope()
END FUNCTION
```

See also:

FilePrint PrintSetup

This function sets print options to use for print and merge. If this function is not used before printing and merging, the default options are used for printing the document. Choosing this function is equivalent to choosing File/Print/Options.

Modified in 3.0 The Flag parameter has a new value, 4096, which prints the current page.

Syntax

PrintOptions(Flag, FirstBin, RestBin)

Flag is a number that defines the value of other print options and may be one or more of the following:

PrintAll (1) - Prints all pages of document, ignoring StartPage and EndPage.

Reverse (2) - Prints the document in reverse order.

Collate (4) - Collate multiple copy output.

CropMarks (16) - Print the document with crop marks.

PrtDocInfo (32) - Print the document description along with the document.

PrePrinted (64) - Print the document on preprinted forms. Do not print protected text.

PrintUpdateFields (128) - Updates power fields before printing.

PrintNotes (256) - Prints the document with notes.

PrintEven (512) - Prints only the even pages of the document.

PrintOdd (1024) - Prints only the odd pages of the document.

PrintBoth (1536) - Prints both even and odd pages of the document.

PrintNoPictures (2048) - Prints the document without pictures. To print with pictures do not use this value.

4096 - Prints only the current page.

The desired options should be added together to make up the value of the flag parameter. The options for setting the sheet feeder bin cannot be set with this command. To set the bins, use the PrintOptions function. If the PrintOptions function is not used, Ami Pro uses the bins specified in the Control Panel.

FirstBin is a number representing the sheet feeder bin to use for the first page of the document.

RestBin is a number representing the sheet feeder bin to use for the remaining pages of the document.

To display the Merge dialog box and allow the user to set print options for merge: **Merge**

To display the Print dialog box and allow the user to set print options for print: **Print**

Return Value

1 (TRUE) if the print options were set.

0 (UserCancel) if the user canceled the function.

-2 (GeneralFailure) if the print options were not set.

Example

```
FUNCTION Example()
'print even pages from bin 1
PrintOptions(512, 1, 1)
FilePrint(1, 1, 9999, 1)
END FUNCTION
```

See also:

FilePrint Merge MergeMacro MergeToFile

This function selects a new printer and printer port for the current document. Choosing this function is equivalent to choosing File/Printer Setup.

Modified in 3.0 There is a new parameter, Port, in Ami Pro 3.0.

Syntax

PrintSetup(Printer, Port)

Printer is the name of the printer that must be used for this document.

Port is the name of the port to which the printer is attached.

To display the Printer Setup dialog box and allow the user to select the name of the printer to use or to allow the printer to be configured: **PrintSetup**

Return Value

1 (TRUE) if the named printer was successfully selected for the document.

0 (UserCancel) if the user canceled the function.

-2 (GeneralFailure) if the printer name was incorrect.

Example

```
FUNCTION Example()
PrintSetup("Apple LaserWriter II NTX", "LPT1:")
Message("Printer changed to LaserWriter on LPT1")
END FUNCTION
```

See also:

ControlPanel FilePrint

This function sets protection of the selected table cells. It acts as a toggle, turning off the protection if it is currently on or turning on the protection if it is currently off. Choosing this function is equivalent to choosing Table/Protect Cells.

You must be in a table to use this function.

Syntax

ProtectCells()

Return Value

1 (TRUE) if the cells were protected or unprotected.

-2 (GeneralFailure) if the cells could not be protected or unprotected or if the program was not in tables mode.

-6 (NoMemory) if the function failed because of insufficient memory.

Example

```
FUNCTION Example()
ProtectCells()
END FUNCTION
```

See also:

TableLayout Tables ConnectCells ProtectedText

This function sets protection for selected text or for all following text if no text is selected. It acts as a toggle, turning off the protection if it is on or turning on the protection if it is off. Choosing this function is equivalent to choosing Edit/Mark Text/Protected Text.

Syntax

ProtectedText()

Return Value

1 (TRUE) if the text was protected.

-2 (GeneralFailure) if the text was not protected or unprotected.

Example

```
FUNCTION Example()
Message("Protecting selected text")
ProtectedText()
END FUNCTION
```

See also:

ConnectCells ProtectCells

This function displays a Windows Message box with a specified prompt and an edit box for the user to enter string data to be sent back to the macro.

Syntax

Query$(Prompt[, Text])

Prompt is a string passed as a prompt to the user. It can be up to 80 characters long.

Text is an optional parameter that displays in the edit box as a default.

Return Value

The string typed by the user.

Null string ("") if the user does not type anything.

If the user chooses Cancel instead of OK, the macro executes the ONCANCEL routine (if one has been defined).

Example

```
FUNCTION Example()
name = Query$("What is your name?", "Cassie")
Message("Hi, {name}")
END FUNCTION
```

See also:

Decide DialogBox Message MultiDecide UserControl

This function sums a column and places the results in the current cell. Choosing this function is equivalent to choosing Table/Quick Add/Column.

Syntax

QuickAddCol()

Return Value

1 (TRUE) if the column was added.

-6 (NoMemory) if the function failed because of insufficient memory.

Example

```
FUNCTION Example()
Message("Adding the current column of numbers")
QuickAddCol()
END FUNCTION
```

See also:

SetFormula EditFormula QuickAddRow

This function sums a row and places the results in the current cell. Choosing this function is equivalent to choosing Table/Quick Add/Row.

Syntax

QuickAddRow()

Return Value

1 (TRUE) if the row was added.

-6 (NoMemory) if the function failed because of insufficient memory.

Example

```
FUNCTION Example()
Message("Adding the current row of numbers")
QuickAddRow()
END FUNCTION
```

See also:

SetFormula EditFormula QuickAddCol

This function allows you to view any messages that have been received. Choosing this function is equivalent to clicking the mail button on the status bar (to the left of the Insert/Type/Rev button). If the mail server is running, the server is activated so you can view the mail that has been received.

This function is only available if a mail server compatible with Ami Pro, such as Lotus Notes or cc:Mail, is installed.

Syntax

ReadMail()

Return Value

1 (TRUE) if the mail server was activated to view the mail.

0 (NoAction) if the mail server was not running or if there is no mail.

Example

```
FUNCTION Example()
result = ReadMail()
IF result = 0
    Message("No mail.")
ENDIF
END FUNCTION
```

See also:

SendMail

This function closes the data file that was opened by the RecOpen function. This data file is in memory.

Syntax

RecClose(Handle)

Handle is the file ID handle returned by the RecOpen function.

Return Value

1 (TRUE) if the data file was closed.

-2 (GeneralFailure) if the data file could not be closed.

Example

```
FUNCTION Example()
DIM field(12)
handle = RecOpen("DATAFILE.SAM", "")
Message(handle)
IF handle THEN
    numfields = RecFieldCount(handle)
    Message("There are {numfields} fields:")
    IF numfields>12 THEN
        Message("Warning - Cannot display all fields")
        numfields=12
    ENDIF
    FOR I = 1 to numfields
        name=(RecFieldName$(handle, I))
        Message("Field name = {name}")
        field(i)=name
    NEXT
    WHILE RecNextRec(handle)=0
        FOR i=1 TO numfields
            name=(RecGetField(handle, field(i)))
            Message("Field value = {name}")
        NEXT
    WEND
    RecClose(handle)
ELSE
    Message("Could not open document")
ENDIF
END FUNCTION
```

See also:

RecFieldCount RecGetField RecNextRec RecOpen RecFieldName$

This function counts the number of fields in the specified data file. This data file is in memory.

Syntax

RecFieldCount(Handle)

Handle is the file ID handle returned by the RecOpen function.

Return Value

The number of fields in the data file.

-2 (GeneralFailure) if the number of fields could not be determined.

Example

```
FUNCTION Example()
DIM field(12)
handle = RecOpen("DATAFILE.SAM", "")
Message(handle)
IF handle THEN
    numfields = RecFieldCount(handle)
    Message("There are {numfields} fields:")
    IF numfields>12 THEN
        Message("Warning - Cannot display all fields")
        numfields=12
    ENDIF
    FOR I = 1 to numfields
        name=(RecFieldName$(handle, I))
        Message("Field name = {name}")
        field(i)=name
    NEXT
    WHILE RecNextRec(handle)=0
        FOR i=1 TO numfields
            name=(RecGetField(handle, field(i)))
            Message("Field value = {name}")
        NEXT
    WEND
    RecClose(handle)
ELSE
    Message("Could not open document")
ENDIF
END FUNCTION
```

See also:

RecGetField RecNextRec RecOpen RecFieldName$ RecClose

This function retrieves the name of the field for the specified field number. The name is retrieved from the data file in memory.

Syntax

RecFieldName$(Handle, FieldNumber)

Handle is the file ID handle returned by the RecOpen function.

FieldNumber is the number of the field name to retrieve from the field names paragraph in the data file.

Return Value

A string containing the requested field name.

-2 (GeneralFailure) if the field name could not be extracted.

Example

```
FUNCTION Example()
DIM field(12)
handle = RecOpen("DATAFILE.SAM", "")
Message(handle)
IF handle THEN
    numfields = RecFieldCount(handle)
    Message("There are {numfields} fields:")
    IF numfields>12 THEN
        Message("Warning - Cannot display all fields")
        numfields=12
    ENDIF
    FOR I = 1 to numfields
        name=(RecFieldName$(handle, I))
        Message("Field name = {name}")
        field(i)=name
    NEXT
    WHILE RecNextRec(handle)=0
        FOR i=1 TO numfields
            name=(RecGetField(handle, field(i)))
            Message("Field value = {name}")
        NEXT
    WEND
    RecClose(handle)
ELSE
    Message("Could not open document")
ENDIF
END FUNCTION
```

See also:

RecFieldCount RecGetField RecNextRec RecOpen RecClose

This function retrieves the contents of the specified field in the current record. The name is retrieved from the data file in memory.

Syntax

RecGetField(Handle, "FieldName")

Handle is the file ID handle returned by the RecOpen function.

FieldName is the name of the field to retrieve from the current record in the data file.

Return Value

The contents of the specified field.

-2 (GeneralFailure) if the contents could not be retrieved.

Example

```
FUNCTION Example()
DIM field(12)
handle = RecOpen("DATAFILE.SAM", "")
Message(handle)
IF handle THEN
    numfields = RecFieldCount(handle)
    Message("There are {numfields} fields:")
    IF numfields>12 THEN
        Message("Warning - Cannot display all fields")
        numfields=12
    ENDIF
    FOR I = 1 to numfields
        name=(RecFieldName$(handle, I))
        Message("Field name = {name}")
        field(i)=name
    NEXT
    WHILE RecNextRec(handle)=0
        FOR i=1 TO numfields
            name=(RecGetField(handle, field(i)))
            Message("Field value = {name}")
        NEXT
    WEND
    RecClose(handle)
ELSE
    Message("Could not open document")
ENDIF
END FUNCTION
```

See also:

RecFieldCount RecNextRec RecOpen RecFieldName$ RecClose

This function advances the record pointer to the next record in the specified data file. This data file is in memory.

Syntax

RecNextRec(Handle)

Handle is the file ID handle returned by the RecOpen function.

Return Value

0 if the action was successful.

-1 if the end of the file was reached.

Example

```
FUNCTION Example()
DIM field(12)
handle = RecOpen("DATAFILE.SAM", "")
Message(handle)
IF handle THEN
    numfields = RecFieldCount(handle)
    Message("There are {numfields} fields:")
    IF numfields>12 THEN
        Message("Warning - Cannot display all fields")
        numfields=12
    ENDIF
    FOR I = 1 to numfields
        name=(RecFieldName$(handle, I))
        Message("Field name = {name}")
        field(i)=name
    NEXT
    WHILE RecNextRec(handle)=0
        FOR i=1 TO numfields
            name=(RecGetField(handle, field(i)))
            Message("Field value = {name}")
        NEXT
    WEND
    RecClose(handle)
ELSE
    Message("Could not open document")
ENDIF
END FUNCTION
```

See also:

RecFieldCount RecGetField RecOpen RecFieldName$ RecClose

This function allows you to open an existing data file into memory.

Syntax

RecOpen(FileName, Description[, Options])

FileName is the name of the data file to open.

Description is the description file. If the data file is an Ami Pro document, the description is the null string ("").

Options is the filter type when import filters are used to read the data file.

Return Value

A positive number if the record is opened.

0 (FALSE) if the record is not opened.

Example

```
FUNCTION Example()
DIM field(12)
handle = RecOpen("DATAFILE.SAM", "")
Message(handle)
IF handle THEN
    numfields = RecFieldCount(handle)
    Message("There are {numfields} fields:")
    IF numfields>12 THEN
        Message("Warning - Cannot display all fields")
        numfields=12
    ENDIF
    FOR I = 1 to numfields
        name=(RecFieldName$(handle, I))
        Message("Field name = {name}")
        field(i)=name
    NEXT
    WHILE RecNextRec(handle)=0
        FOR i=1 TO numfields
            name=(RecGetField(handle, field(i)))
            Message("Field value = {name}")
        NEXT
    WEND
    RecClose(handle)
ELSE
    Message("Could not open document")
ENDIF
END FUNCTION
```

See also:

RecFieldCount RecGetField RecNextRec RecFieldName$ RecClose

This function deletes the page layout for the current page. The page layout for the previous page is effective for the current page. If the page layout deleted was for the first page, the original (default) page layout is in effect. Choosing this function is equivalent to choosing Page/Insert Page Layout/Remove.

You must be in Layout Mode to use this function.

Syntax

RemoveLayout()

Return Value

1 (TRUE) if the page layout was deleted.

-2 if the page layout was not deleted.

0 (NoAction) if no action was taken.

-6 (NoMemory) if the function failed because of insufficient memory.

Example

```
FUNCTION Example()
choice = Decide("This will remove the current page layout. Continue?")
IF choice = 1
   RemoveLayout()
ENDIF
END FUNCTION
```

See also:

InsertLayout RevertLayout

This function renames the eight user defined document information fields. Choosing this function is equivalent to choosing File/Doc Info, selecting Other Fields, and then selecting Rename Fields.

Syntax

RenameDocInfoField(FieldNumber, Field)

FieldNumber is the number of the field to rename. The first field number is 0.

Field is the new name for the field.

To display the Document Info dialog box and allow the user to choose names for the fields: **DocInfo**

Return Value

1 (TRUE) if the new document info field name was saved.

-2 (GeneralFailure) if the name was not saved.

Example

```
FUNCTION Example()
FieldNum = Query$("What field do you want to rename?")
FieldName = Query$("What do you want to rename field #{FieldNum} to?")
RenameDocInfoField(FieldNum, FieldName)
END FUNCTION
```

See also:

DocInfo InsertDocInfo

This function gives a new name to an existing menu item. RenameMenuItem does not change the functionality of the menu item, but simply gives it a new name. The new name displays the next time the user displays the pulldown menu.

Syntax

RenameMenuItem(BarID, Menu[, CascadeMenu], OldName, NewName)

BarID is the identification number of the menu bar returned from the AddBar function. To use the default Ami Pro menu bar, use 1.

Menu is the name of the pull down menu that contains the item you wish to rename. This must match exactly the name of the pull down menu you want to modify, including ampersand (&) characters. An ampersand is placed before a character that has an underline.

CascadeMenu is the optional Cascade menu name that contains the item you wish to rename. This must match exactly the name of the cascade menu you want to modify, including ampersand (&) characters.

OldName is the current name of the menu item.

NewName is the new name to use for the menu item.

Return Value

1 (TRUE) if the item was successfully renamed.

0 (FALSE) if the item could not be found.

Example

```
FUNCTION Example()
RenameMenuItem(1, "&File", "E&xit", "&Quit")
END FUNCTION
```

See also:

AddBar AddMenu AddMenuItem AddMenuItemDDE ChangeMenuAction CheckMenuItem DeleteMenu DeleteMenuItem GrayMenuItem ShowBar

This function finds and optionally replaces text or paragraph styles in a document. Choosing this function is equivalent to choosing Edit/Find & Replace. If the macro replaces all the occurrences of a phrase or paragraph style, the macro works as if the user initiated Find & Replace and issued the Replace All command. If the ReplaceAll flag is not used, this function finds the first occurrence of the target phrase and continues with the macro. If the ReplaceAll flag is not used, this function does not perform a replacement.

The Replace function is not interactive. If an interactive Find & Replace is desired, the FindReplace function should be used. A macro must be edited to insert this non-recordable function.

Syntax

Replace(SourceAttr, DestAttr, Flag, FindString, ReplaceString)

SourceAttr is the attribute combination for the source string and can be one or more of the following:

NormalAttr (0) - Text without any attributes
BoldText (4) - Bold Text
ItalicText (8) - Italicized text
UnderlineText (16) - Underlined text
WordUnderlineText (32) - Word underlined text
SmallCaps (-32768) - Small caps text

Multiple attributes can be specified by adding the values for the attributes together.

DestAttr is the attribute combination for the destination string and can be one or more of the following:

NormalAttr (0) - Text without any attributes
BoldText (4) - Bold Text
ItalicText (8) - Italicized text
UnderlineText (16) - Underlined text
WordUnderlineText (32) - Word underlined text
SmallCaps (-32768) - Small caps text

Multiple attributes can be specified by adding the values for the attributes together.

Flag is a flag that defines the options for the Find & Replace and can be one or more of the following:

FindExact (1) - Find only text with same capitalization as typed in search string
FindStyles (2) - Find/replace paragraph styles instead of text
FindWholeWord (4) - Find only whole words
FindBackwards (8) - Find from the current cursor position to the beginning of the document
ReplaceExact (16) - Replace using capitalization specified for replace string
ReplaceExactAttr (32) - Replace using attributes specified, rather than as found
FindExactAttr (64) - Find only text with same attributes as specified
ReplaceAll (1024) - Replace all occurrences without stopping for confirmation
FindFromPage1 (8192) - Start search on page 1 of document
FindCurrentStream (16384) - Find only in the current text stream

Note: Setting this option turns off the option to search in other text streams, unlike the other flags, which turn on the option.

Multiple options can be specified by adding values together.

FindString is the string or paragraph style to search for.

ReplaceString is the string or paragraph style name to replace with. If the ReplaceAll flag is not used, this parameter is ignored.

To display the Find/Replace dialog box and allow the user to determine the parameters for the find and replace operation: **FindReplace**

Return Value

1 (TRUE) if the find/replace function was successful.

-7 (EndFile) if the end of the file was reached or if no match was found. Since a global replacement always reaches the end of the file, this function always returns EndFile if the ReplaceAll option is used.

-6 (NoMemory) if the function failed because of insufficient memory.

Example

```
FUNCTION Example()
Message("Replacing a's with x's")
Replace(0, 0, 0, "a", "x")
END FUNCTION
```

See also:

FindReplace GoToCmd GoToShade

This function restores the Ami Pro window to the size of the last non-full screen window. Choosing this function is equivalent to choosing System/Restore.

Syntax

Restore()

Return Value

This function does not return a value.

Example

```
FUNCTION Example()
Restore()
END FUNCTION
```

See also:

Maximize Minimize

This function cancels changes made to the document since it was last saved and displays the previously saved version of the document. Choosing this function is equivalent to choosing File/Revert to Saved.

Normally, the user is given a chance to abort the revert. If you want to avoid this message in your macro, use the Messages function to turn off messages before using this function.

Syntax

Revert()

Return Value

1 (TRUE) if the file successfully reverted to the previous version.

-2 (GeneralFailure) if the file could not be reverted.

Example

```
FUNCTION Example()
Revert()
END FUNCTION
```

See also:

Messages Save SaveAs

This function restores the original page layout to the layout for the current page. Choosing this function is equivalent to choosing Page/Insert Page Layout/Revert.

Syntax

RevertLayout()

Return Value

1 (TRUE) if the page layout was successfully reverted.

-6 (NoMemory) if the function failed because of insufficient memory.

Example

```
FUNCTION Example()
RevertLayout()
END FUNCTION
```

See also:

InsertLayout RemoveLayout

This function initiates Revision Marking Review. If any revisions are found, the Review Revision Marking dialog box is displayed and the user may decide whether to accept, cancel, or skip the revision. This function does not automatically accept or cancel a revision. Choosing this function is equivalent to choosing Tools/Revision Marking/Review Rev.

Syntax

ReviewRevisions()

Return Value

1 (TRUE) if the mode was changed.

0 (UserCancel) if the user canceled the function.

-2 (GeneralFailure) if the mode could not be changed.

Example

```
FUNCTION Example()
' Review revisions and turn off revision marking
RevisionMarking(2 0)
ReviewRevisions()
END FUNCTION
```

See also:

RevisionMarking RevisionMarkOpts

This function marks selected text as a revision insertion. Choosing this function is equivalent to choosing Edit/Mark Text/Revision Insertion.

Syntax

RevisionInsertion()

Return Value

1 (TRUE) if the revision was accepted.

0 (NoAction) if no action was taken.

Example

```
FUNCTION Example()
Message("Marking selected text for revision insertion.")
RevisionInsertion()
END FUNCTION
```

See also:

RevisionMarking ReviewRevisions

This function turns revision marking on or off and lets you determine what revision marking function to perform. Choosing this function is equivalent to choosing Tools/Revision Marking.

Syntax

RevisionMarking(Which, State)

Which defines which function to perform and can be one of the following:

MarkRevisions (1) - Changes the revision state only
ReviewRevisions (2) - Step through each revision mark from the beginning of the document
AcceptRevisions (3) - Accept all revisions in the document
CancelRevisions (4) - Cancel all revisions in the document

State is a flag which determines if revision marking is in effect and can be one of the following:

RevOn (1) - Turn on revision marking
RevOff (0) - Turn off revision marking

To display the Revision Marking dialog box and allow the user to select the revision marking function: **RevisionMarking**

Return Value

1 (TRUE) if the changes were accepted.

0 (UserCancel/FALSE) if the user canceled the function or if no action was taken.

Example

```
FUNCTION Example()
Message("Turn on revision marking.")
RevisionMarking(1, 1)
END FUNCTION
```

See also:

RevisionMarkOpts RevisionInsertion ReviewRevisions

This function is used to set the attributes and colors for marking insertions and deletions for revision marking. Options may also be changed for showing marks in margins. Choosing this function is equivalent to choosing Tools/Revision Marking/Options.

Syntax

RevisionMarkOpts(InsFlag, InsUseColor, InsColor, DelFlag, DelChar, DelUseColor, DelColor, MarksFlag, MarksChar, Position)

InsFlag defines which attributes to use for marking insertions and can be one of the following:

RevNoAttribute (0) - No attribute set for marking insertions
RevBold (1) - Use bold attribute for marking insertions
RevItalic (2) - Use italic attribute for marking insertions
RevbUnderline (3) - Use underline attribute for marking insertions
RevDblUnderline (4) - Use double underline attribute for marking insertions

InsUseColor indicates if color is to be used for marking insertions and can be one of the following:

ColorOn (1) - Use color
ColorOff (0) - Do not use color

InsColor is the color to use for marking insertions and can be one of the following:

White (16777215)
Cyan (16776960)
Yellow (65535)
Magenta (16711935)
Green (65280)
Red (255)
Blue (16711680)
Black (0)

DelFlag indicates which attributes to use for marking deletions and can be one of the following:

RevNoAttribute (0) - No attribute set for marking deletions
RevStrikeThru (1) - Strike through the delete characters
RevOverstrike (2) - Use the DelChar parameter to overstrike the deleted characters

DelChar is the overstrike character to use for marking deletions.

DelUseColor sets the flag if color is to be used for marking deletions and can be one of the following:

ColorOn (1) - Use color
ColorOff (0) - Do not use color

DelColor is the color to use for marking deletions and can be one of the following:

White (16777215)
Cyan (16776960)
Yellow (65535)
Magenta (16711935)
Green (65280)
Red (255)
Blue (16711680)
Black (0)

MarksFlag defines the kind of mark used to show marks in margins. It is one of the following values:

RevNoAttribute (0) - No attribute set for showing marks in margins
RevBars (1) - Use revision bar when showing marks in margins

RevChar (2) - Use the MarksChar to show marks in margins

MarksChar is the character used for showing marks in margins.

Position defines in which margin the marks indicator is displayed and can be one of the following:

RevMarkLeft (0) - Show marks in left margin

RevMarkRight (1) - Show marks in right margin

RevMarkRightLeft (2) - Show marks for even pages in the left margin and odd pages in the right margin

To display the revision marking options dialog box and allow the user to select the options: **RevisionMarkOpts**

Return Value

1 if the attributes are set.

0 (UserCancel) if the user canceled the function.

Example

```
FUNCTION Example()
RevisionMarkOpts(RevItalic, ColorOn, 16711680, RevStrikeThru, 0, ColorOn, 255, RevBars, 0,
RevMarkRightLeft)
RevisionMarking(1, 1)
END FUNCTION
```

See also:

RevisionMarking ReviewRevisions RevisionInsertion

This function retrieves the specified number of characters from the right end of the specified string.

Syntax

Right$(Text, Length)

Text is the string to parse.

Length is the number of characters from the right end of the string to parse.

Return Value

A string containing the specified characters parsed from the original string that may be the null string.

Example

```
FUNCTION Example()
DEFSTR Char;
OpenFile = GetOpenFileName$()
I = LEN(OpenFile)
WHILE "\" != Assign(&Char, MID$(OpenFile, I, 1))
    I = I - 1
WEND
FileName = Right$(OpenFile, (LEN(OpenFile) - I))
Message("The current file is {FileName}.")
END FUNCTION
```

See also:

Left$ MID$ Instr LEN

This function acts as a toggle to turn right alignment on or off for a paragraph of text. Choosing this function is equivalent to choosing Text/Alignment/Right.

Syntax

RightAlign()

Return Value

1 (TRUE) if the text was right aligned, or if alignment was removed.

-6 (NoMemory) if the function failed because of insufficient memory.

Example

```
FUNCTION Example()
String = "This is a line of Text. "
FOR i = 1 to 10
    Paste()
NEXT
LeftAlign()
END FUNCTION
```

See also:

Center Justify LeftAlign NormalText

This function scrolls the document to the right edge of the page without moving the insertion point. Choosing this function is equivalent to dragging the scroll box on the horizontal scroll bar to the extreme right.

Syntax

RightEdge()

Return Value

This function returns 0.

Example

```
FUNCTION Example()
RightEdge()
END FUNCTION
```

See also:

CharLeft CharRight EndOfFile LeftEdge LineDown LineUp ScreenDown ScreenLeft ScreenRight ScreenUp TopOfFile

This function rounds a number to an integer. Values with fractional parts greater than or equal to .5 are rounded up to the next highest integer value. Those with fractional parts less than .5 are rounded down to the next lowest integer.

Syntax

Round(Value)
Value is the number to be rounded.

Return Value

This function returns an integer.

Example

```
FUNCTION Example()
Number = Query$("What number do you want to round?")
Round(Number)
END FUNCTION
```

See also:

Mod IsNumeric

This function allows one macro to start another at a later time. While waiting for the called macro to start, control is turned back to the user.

Syntax

RunLater(MacroName[!Function[(parm1[, parm2...])]][, Hours:]Minutes[.Seconds])

MacroName is the macro to run at a later time.

Hours is the number of hours to wait before playing back the macro.

Minutes is the number of minutes to wait before playing back the macro.

Seconds is the number of seconds to wait before playing back the macro.

Return Value

This function does not return a value.

Example

```
FUNCTION Example()
MacFile = GetRunningMacroFile$()
RunLater("{MacFile}!Example2()", 0.10)
END FUNCTION

FUNCTION Example2()
Message("It is now 10 seconds after you pushed ""OK"" in the message box.")
END FUNCTION
```

See also:

GetTime Pause

This function saves the currently active document to disk. Choosing this function is equivalent to choosing File/Save. This function cannot be used to save an Untitled document. You should use the SaveAs function to save an Untitled document.

Syntax

Save()

Return Value

1 (TRUE) if the file was saved.

0 (UserCancel/FALSE) if the user canceled the function.

-2 (GeneralFailure) if the file was not saved, if the file was password protected and password verification failed, if the file was read-only, or if the file to be saved was untitled.

Example

```
FUNCTION Example()
result = Save()
IF result = 0
    Message("Use SaveAs")
ENDIF
END FUNCTION
```

See also:

SaveAs Revert

This function saves the document on the screen with a different name or file type. Choosing this function is equivalent to choosing File/Save As.

Syntax

SaveAs(FileName, Options, Description, App)

FileName is the name of the file to save the document as. This parameter may optionally include a path.

Options are options for saving the file if saving as an Ami Pro document. It is one of the following values:

0 - Ami Pro file or Non-ASCII file

4 - ASCII file

KeepFormat (32) - Keep format

Password (64) - Use password protection

SaveAs12 (128) - Save as Ami Pro release 1.2 format

Password and SaveAs12 values may not be used together. If saving as an Ami Pro Macro file type, use only the KeepFormat value.

Description is the document description of the new file.

App is the name of the filter that appears in the AMIPRO.INI file. Use the null string ("") to specify an Ami Pro file.

To display the Save As dialog box and allow the user to choose the name of the file: **SaveAs**

Return Value

1 (TRUE) if the file was saved.

0 (UserCancel) if the user canceled the function.

-2 (GeneralFailure) if the file was not saved.

Example

```
FUNCTION Example()
SaveAs("test.sam", 32, "Test file", "")
END FUNCTION
```

See also:

FileOpen Revert Save

This function saves the current open Ami Pro object as an Ami Pro master. Choosing this function is equivalent to choosing Action/Save As Master. After saving, the master is added to the list of Ami Pro masters in the Create A New dialog box.

Syntax

SaveAsMaster(MasterName, Options)

MasterName is the name of the new master and can be up to 32 characters long.

Options is one of the following values:

NoContents (0) - Do not save the object's contents in the master

WithContents (1) - Save the object's contents in the master

To display the Save As Master dialog box and allow the user to select the parameters: **SaveAsMaster**

Return Value

1 (TRUE) if the object was saved.

0 (UserCancel/NoAction) if the user canceled the function or if no action was taken.

-2 (GeneralFailure) if the object was not saved.

Example

```
FUNCTION Example()
SaveAsMaster("Mastername", 1)
END FUNCTION
```

See also:

CreateANew ImportText IsNewWave ListObjects NWGetContainerCount NWGetContainerNames NWGetCurrentContainer NWGetCurrentObject$ NWGetObjectCount NWGetObjectNames NWGetParent NWReferenceToFile$ ObjectAttributes OpenObject SaveAsObject Share ShowLinks

This function saves the current document's paragraph styles to a new paragraph style sheet. Choosing this function is equivalent to choosing Style/Save as a Style Sheet.

Modified in 3.0 This function contains a new parameter, Description, in Ami Pro release 3.0.

Syntax

SaveAsNewStyle(Name, WithContents, MacroName, RunMacro, Description)

Name is the name to name the new style sheet.

WithContents is a flag that determines whether to save the document's contents in the new paragraph style sheet and can be one of the following:

(1) - Save the contents of the file in the paragraph style sheet

(0) - Do not save the file's contents in the paragraph style sheet

MacroName is the name of the macro to run when a new file is opened with the selected paragraph style sheet.

RunMacro determines whether or not to run MacroName when the new file with the selected paragraph style sheet is opened. It is one of the following values:

(1) - Run the macro

(0) - Do not run the macro

Description is the style sheet description.

To display a dialog box to allow the user to select the name for the new paragraph style sheet: **SaveAsNewStyle**

Return Value

1 (TRUE) if the paragraph style sheet was saved.

0 (UserCancel) if the user canceled the function.

Example

```
FUNCTION Example()
SaveAsNewStyle("Example.sty", 1, "", "", "My example style sheet")
END FUNCTION
```

See also:

UseAnotherStyle

This function saves the current open Ami Pro object. Choosing this function is equivalent to choosing Action/Save As.

To save as a non-Ami Pro object, use the SaveAs function.

Syntax

SaveAsObject(ObjectName, Options, Path, Description)

ObjectName is the name of the new object saved.

Options is one of the following values:

(32) - Keep format
(64) - Use password protection
(128) - Save as Ami Pro release 1.2 format

Path is the location where the object is to be saved.

Description is the document description string.

To display the Save As dialog box and allow the user to select the parameters: **SaveAs**

Return Value

1 (TRUE) if the object was saved.

0 (UserCancel) if the user canceled the function.

-2 (GeneralFailure) if the object was not saved.

Example

```
FUNCTION Example()
SaveAsObject("Objectname" 32 "\" "")
END FUNCTION
```

See also:

CreateANew ImportText IsNewWave ListObjects NWGetContainerCount NWGetContainerNames NWGetCurrentContainer NWGetCurrentObject$ NWGetObjectCount NWGetObjectNames NWGetParent NWReferenceToFile$ ObjectAttributes OpenObject SaveAsMaster Share ShowLinks

This function scrolls the document down one screen without moving the insertion point. Choosing this function is equivalent to clicking above the scroll box on the vertical scroll bar.

Syntax

ScreenDown()

Return Value

1 (TRUE) if the document was able to scroll.

-2 (GeneralFailure) if the document could not be scrolled.

Example

```
FUNCTION Example()
ScreenDown()
END FUNCTION
```

See also:

CharLeft CharRight EndOfFile LeftEdge LineDown LineUp RightEdge ScreenLeft ScreenRight ScreenUp TopOfFile

This function scrolls the document left one screen without moving the insertion point. Choosing this function is equivalent to clicking to the right of the scroll box on the horizontal scroll bar.

Syntax

ScreenLeft()

Return Value

1 (TRUE) if the document was able to scroll.

-2 (GeneralFailure) if the document could not be scrolled.

Example

```
FUNCTION Example()
ScreenLeft()
END FUNCTION
```

See also:

CharLeft CharRight EndOfFile LeftEdge LineDown LineUp RightEdge ScreenDown ScreenRight ScreenUp TopOfFile

This function scrolls the document right one screen without moving the insertion point. Choosing this function is equivalent to clicking to the left of the scroll box on the horizontal scroll bar.

Syntax

ScreenRight()

Return Value

1 (TRUE) if the document was able to scroll.

-2 (GeneralFailure) if the document could not be scrolled.

Example

```
FUNCTION Example()
ScreenRight()
END FUNCTION
```

See also:

CharLeft CharRight EndOfFile LeftEdge LineDown LineUp RightEdge ScreenDown ScreenLeft ScreenUp TopOfFile

This function scrolls the document up one screen without moving the insertion point. Choosing this function is equivalent to clicking below the scroll box on the vertical scroll bar.

Syntax

ScreenUp()

Return Value

1 (TRUE) if the document was able to scroll.

-2 (GeneralFailure) if the document could not be scrolled.

Example

```
FUNCTION Example()
ScreenUp()
END FUNCTION
```

See also:

CharLeft CharRight EndOfFile LeftEdge LineDown LineUp RightEdge ScreenDown ScreenLeft ScreenRight TopOfFile

This function selects a column in a table. The column that contains the insertion point is selected. Choosing this function is equivalent to choosing Table/Select Column.

The insertion point must be in the table before performing this function.

Syntax

SelectColumn()

Return Value

1 (TRUE) if the column is selected.

0 (NoAction) if no action is taken. The table may not exist or may not be selected.

Example

```
FUNCTION Example()
SelectColumn()
END FUNCTION
```

See also:

SelectEntireTable SelectRow

This function selects the entire table. Choosing this function is equivalent to choosing Table/Select Entire Table.

The insertion point must be in the table before performing this function.

Syntax

SelectEntireTable()

Return Value

1 (TRUE) if the entire table is selected.

0 (NoAction) if no action is taken. The table may not exist or may not be selected.

Example

```
FUNCTION Example()
SelectEntireTable()
END FUNCTION
```

See also:

SelectColumn SelectRow DeleteEntireTable

This function allows you to select a frame by name. Use the MarkBookMark function to name the frames.

Syntax

SelectFrameByName(FrameName)

FrameName is the name of the frame you want to select.

Return Value

0 if the frame could not be found.

Example

```
FUNCTION Example()
myframe = Query$("Enter the name of the bookmark set to the frame you want to select:")
SelectFrameByName(myframe)
END FUNCTION
```

See also:

MarkBookMark GoToCmd

This function selects a row in a table. The row that contains the insertion point is selected. Choosing this function is equivalent to choosing Table/Select Row.

The insertion point must be in the table before performing this function.

Syntax

SelectRow()

Return Value

1 (TRUE) if the row is selected.

0 (NoAction) if no action is taken. The table may not exist or may not be selected.

Example

```
FUNCTION Example()
SelectRow()
END FUNCTION
```

See also:

SelectColumn SelectEntireTable

This function allows the user to select a new paragraph style. Choosing this function is equivalent to choosing Style/Select a Style. This function does not automatically select a new paragraph style.

To have the macro select a paragraph style for the user, use the SetStyle function.

Syntax

SelectStyle()

Return Value

1 (TRUE) if the new paragraph style was selected.

0 (UserCancel) if the user canceled the function.

-2 (GeneralFailure) if the paragraph style was not selected.

Example

```
FUNCTION Example()
SelectStyle()
END FUNCTION
```

See also:

ModifyStyle SetStyle ShowStylesBox ToggleStylesBox

This function allows you to give any Multiple Document Interface (MDI) document in Ami Pro the focus and make it the active document.

Syntax

SelectWindow(FileName)

FileName is the name of the document to get the focus. This name must match the name in the title bar of the document.

Return Value

0 if the document was not selected, or was not found.

Example

```
FUNCTION Example()
filename = Query$("Name the window to select (name must match exactly).")
SelectWindow(filename)
END FUNCTION
```

See also:

NewWindow NextWindow TileWindow CascadeWindow

This function sends the selected frame to the rear of a stack of frames. Choosing this function is equivalent to choosing Frame/Send to Back.

Syntax

SendFrameToBack()

Return Value

1 (TRUE) if the frame was sent to the back.

0 (NoAction) if no action was taken because the frame was already in the back.

Example

```
FUNCTION Example()
x = strfield$(CursorPosition$(), 1, ",")
y = strfield$(CursorPosition$(), 2, ",")
AddFrame(x, y, (x + 1440), (y - 1440))
MarkBookMark("Frame1", AddBookmark)
AddFrame((x + 360), (y - 360), (x + 1800), (y - 1800))
MarkBookMark("Frame2", AddBookmark)
SendFrameToBack()
MarkBookMark("Frame1", FindBookmark)
Message("The First frame is in front.")
MarkBookMark("Frame2", FindBookmark)
BringFrameToFront()
Message("The Second frame is in front.")
END FUNCTION
```

See also:

BringFrameToFront GoToCmd AddFrame AddFrameDLG

This function sends keystrokes to the active application.

Syntax

SendKeys(Keys)

Keys is a string that is typed into the document.

Keys can be any of the keyboard characters. Any attributes or variables within the string are not typed into the document.

To insert a left curly brace, type two curly braces ({{).

To insert a left square brace, type two square braces ([[).

To insert a double quote mark, type two double quote marks ("").

Text can contain a variable name. If a variable name is used, enclose it in curly braces ({}) so that it can be recognized as a variable name. Text can also contain an insertion point movement or function key. To type a key, surround its name with square braces. An insertion point movement or function key can include the words CTRL, SHIFT or ALT to indicate a shifted state. The following key names can be used:

[Home] - Home Key
[End] - End Key
[PgUp] - Page Up Key
[PgDn] - Page Down Key
[Ins] - Insert Key
[Del] - Delete Key
[Backspace] - Backspace Key
[Enter] - Enter or Return Key
[Tab] - Tab Key
[ESC] - Escape Key
[Up] - Up Arrow Key
[Down] - Down Arrow Key
[Left] - Left Arrow Key
[Right] - Right Arrow Key
[F1] - [F12] - Function Keys F1 through F12

Return Value

This function does not return a value.

Example

```
FUNCTION Example()
Exec("NOTEPAD.EXE", 1)
SendKeys("[altf]o")
END FUNCTION
```

See also:

TYPE Exec ActivateApp

This functions allows you to send a mail message using a mail server. Currently, Ami Pro supports cc:Mail and Lotus Notes. If the text has been selected in the current document, the text is copied to the clipboard and then sent as a note. Choosing this function is equivalent to choosing File/Send Mail.

You can only use one mail server. This function is only available if a mail server compatible with Ami Pro is installed.

Syntax

SendMail(Attach)

Attach is used to save and attach the active document to the note. It is one of the following values:

Attach (1) - Save and attach the active document to the note

No (0) - Do not attach the active document to the note

To display the Send Mail dialog box: **SendMail**

Return Value

1 (TRUE) if the mail was sent.

0 (UserCancel/FALSE) if the user canceled the function or if no action was taken.

-2 (GeneralFailure) if the mail server was not found or if it could not be activated.

Example

```
FUNCTION Example()
Message("Saving and sending this doc")
Save()
SendMail(1)
END FUNCTION
```

See also:

ReadMail

This function sets the default path for backup files. Choosing this function is equivalent to choosing Tools/User Setup/Paths.

Syntax

SetBackPath(NewPath)

NewPath is the desired new backup path.

Return Value

1 (TRUE) if the path was successfully set.

-2 (BadPath) if the path could not be set because of an invalid directory name.

Example

```
FUNCTION Example()
Backup = Query$("Please enter the desired backup path:" GetBackPath$())
Docs = Query$("Please enter the desired document path:" GetDocPath$())
Styles = Query$("Please enter the desired styles path:" GetStylePath$())
SetDocPath(Docs)
SetStylePath(Styles)
SetBackPath(Backup)
END FUNCTION
```

See also:

GetBackPath$ GetDocPath$ GetStylePath$ SetDocPath SetStylePath

This function sets the data and description file for the InsertMerge function.

Syntax

SetDataFile(RecFile, RecFileFlag, DescFile, DescFileFlag)

RecFile is the record file to use when performing a merge. If the RecFile is a NewWave object, the parameter must be the full path to the object name.

RecFileFlag is the flag to identify whether the DataFile is a NewWave object or Ami Pro file. If it is a NewWave object, the parameter must be set to 1. If it is an Ami Pro file, the RecFileFlag parameter should be set to 0.

DescFile is the description file to use when the record file is not an Ami Pro file. If no description file is needed, insert an empty string (""). If the DescFile is a NewWave object, the parameter must be the full path to the object name.

DescFileFlag is the flag to identify whether the DescFile is a NewWave object or Ami Pro file. If it is a NewWave object, the parameter must be set to 1. If it is an Ami Pro file, the DescFileFlag parameter should be set to 0.

Return Value

1 (TRUE) if the record and description files are set.

-3 if invalid input.

0 (NoAction) if no action was taken.

Example

```
FUNCTION Example()
SetDataFile ("Testrel.sam", 0, "", 0)
InsertMerge()
END FUNCTION
```

See also:

InsertMerge

This function sets editing defaults for using Ami Pro. Choosing this function is equivalent to choosing Tools/User Setup/Options.

Modified in 3.0 There are several new values for the Options parameter in Ami Pro release 3.0. Those values are 4, 8, 16, 32, 64, 128, and 256.

Syntax

SetDefOptions(Options, HotZone)

Options is a flag containing default options and can be one or more of the following:

DoKerning (1) - Use pair kerning

DoWindows (2) - Control widows and orphans in the document

(4) - Print in background (speed option)

(8) - Flow in background (speed option)

(16) - Save for fast display (graphic display speed option - saves often)

(32) - Save while open (graphic display speed option - saves on open)

(64) - Conserve disk space (graphic display speed option - no auto save)

(128) - Hyphenate last word in paragraph

(256) - Hyphenate last word in column/page

To set more than one option, add the options together.

HotZone is the number of characters to use for the hyphenation hot zone. You can specify any number between 2 and 9.

To display the Defaults dialog box and allow the user to select his editing defaults: **UserSetup**

Return Value

1 (TRUE) if the default settings were successfully set.

0 (UserCancel) if the user canceled the function.

Example

```
FUNCTION Example()
SetDefOptions(0, 5)
END FUNCTION
```

See also:

ViewPreferences SetBackPath SetDocPath SetStylePath UserSetup LoadOptions

This function displays the Default Paths dialog box. It does not automatically set the default paths. Choosing this function is equivalent to choosing Tools/User Setup/Paths.

Syntax

SetDefPaths()

Return Value

This function returns 0.

Example

```
FUNCTION Example()
SetDefPaths()
END FUNCTION
```

See also:

UserSetup SetDocPath SetStylePath SetBackPath SetMacroPath

This function is used to flag a control in the next displayed dialog box as a callback. If the contents of the control are modified once the dialog box is displayed, the specified macro is called.

A callback is a macro function that is initiated by a displayed dialog box and executes without closing that box.

Syntax

SetDlgCallBack(ID, MacroName!Function)

ID is the ID of the control that you want to monitor.

Macroname is the filename of the macro file containing the function to be executed. If the macro file does not exist in the current macros directory, the full path must be used.

Function is the name of the function to be executed if the specified object is modified. MacroName and Function are separated by an exclamation mark.

You may not send any parameters or even use parenthesis with this parameter. The specified function is called with three parameters: the windows handle for the dialog box, the ID specified, and the contents of the specified object.

Return Value

This function does not return a value.

Example

```
FUNCTION Example()
MacFile = GetRunningMacroFile$()
SetDlgCallBack(50, "{MacFile}!Message1")
Box = DialogBox(".", "ExampleBox")
IF Box = 0
    EXIT FUNCTION
ENDIF
TYPE(GetDialogField$(8002))
END FUNCTION

FUNCTION Message1(hdlg, id, text)
Name = GetDlgItemText(hdlg, 8000)
Message("The contents of the first box are {Name}.")
Message("We will now fill the second box with the inverse of {Name}.")
FOR I = len(Name) to 1 step -1
    Name2 = strcat$(Name2, MID$(Name, I, 1))
NEXT
SetDlgItemText(hdlg, 8002, Name2)
END FUNCTION

DIALOG ExampleBox
-2134376448 8 106 38 160 54 "" "" "Sample Dialog Box"
FONT 6 "Helv"
50 6 62 12 8000 1350631552 "edit" "" 0
4 6 44 10 1000 1342177280 "static" "Your Name:" 0
50 20 62 12 1003 1342177287 "static" "" 0
4 22 44 10 1002 1342177280 "static" "Reversed:" 0
52 22 58 8 8002 1342177280 "static" "" 0
116 4 40 14 1 1342242817 "button" "OK" 0
116 20 40 14 2 1342242816 "button" "Cancel" 0
100 36 56 14 50 1342242816 "button" "&Run Example..." 0
END DIALOG
```

See also:

This function sets the contents of a specified control in a callback, while the dialog box remains displayed.

Syntax

SetDlgItemText(Handle, ID, Text)

Handle is the windows handle to the dialog box to be modified and it was passed to the callback function.

ID is the control ID in which to set the contents.

Text is the string data (if modifying an edit, combo, or list box, or static text), or a Boolean TRUE or FALSE value (if modifying a radio button or check box) to set the contents of the control to.

Return Value

1 (TRUE) if the function was successful.

0 (FALSE) if the function failed.

Example

```
FUNCTION Example()
MacFile = GetRunningMacroFile$()
SetDlgCallBack(50, "{MacFile}!Message1")
Box = DialogBox(".", "ExampleBox")
IF Box = -1
    Message("Could not find dialog box!")
    EXIT FUNCTION
ELSEIF Box = 0
    EXIT FUNCTION
ENDIF
TYPE(GetDialogField$(8002))
END FUNCTION

FUNCTION Message1(hdlg, id, text)
Name = GetDlgItemText(hdlg, 8000)
Message("The contents of the first box are {Name}.")
Message("We will now fill the second box with the inverse of {Name}.")
FOR I = len(Name) to 1 step -1
    Name2 = strcat$(Name2, MID$(Name, I, 1))
NEXT
SetDlgItemText(hdlg, 8002, Name2)
END FUNCTION

DIALOG ExampleBox
-2134376448 8 106 38 160 54 "" "" "Sample Dialog Box"
FONT 6 "Helv"
50 6 62 12 8000 1350631552 "edit" "" 0
4 6 44 10 1000 1342177280 "static" "Your Name:" 0
50 20 62 12 1003 1342177287 "static" "" 0
4 22 44 10 1002 1342177280 "static" "Reversed:" 0
52 22 58 8 8002 1342177280 "static" "" 0
116 4 40 14 1 1342242817 "button" "OK" 0
116 20 40 14 2 1342242816 "button" "Cancel" 0
100 36 56 14 50 1342242816 "button" "&Run Example..." 0
END DIALOG
```

See also:

GetDLGItem GetDLGItemText DialogBox FillEdit FillList GetDialogField$

This function sets the default path for document storage. Choosing this function is equivalent to choosing Tools/User Setup/Paths.

Syntax

SetDocPath(NewPath)

NewPath is the desired new default document path.

Return Value

1 (TRUE) if the path was successfully set.

-2 (GeneralFailure) if the path could not be set.

Example

```
FUNCTION Example()
Backup = Query$("Please enter the desired backup path:" GetBackPath$())
Docs = Query$("Please enter the desired document path:" GetDocPath$())
Styles = Query$("Please enter the desired styles path:" GetStylePath$())
SetDocPath(Docs)
SetStylePath(Styles)
SetBackPath(Backup)
END FUNCTION
```

See also:

ViewPreferences GetBackPath$ GetDocPath$ GetStylePath$ SetBackPath SetStylePath UserSetup SetDefPaths

This function creates or updates a document variable in the current document. A document variable is used to store information in the document and may be retrieved by using the GetDocVar function.

Syntax

SetDocVar(Name, Text)

Name is the name for the document variable by which it may be accessed.

Text is the information that is stored in the document variable.

The two strings combined must not exceed 250 bytes in length.

Return Value

This function returns 1.

Example

```
FUNCTION Example()
SetDocVar("answer 1", "George Washington")
END FUNCTION
```

See also:

GetDocVar

This function inserts a formula into the current table cell. Choosing this function is equivalent to choosing Table/Edit Formula.

Syntax

SetFormula(Formula)

Formula is the formula that is typed into the table cell.
To allow the user to type the formula for the cell: **EditFormula**

Return Value

1 (TRUE) if the formula was inserted.

-6 (NoMemory) if the function failed because of insufficient memory.

Example

```
FUNCTION Example()
radius = Query$("What is the radius?")
SetFormula(3.1415926 * radius * radius)
END FUNCTION
```

See also:

EditFormula

This function sets the defaults for the frame. Choosing this function is equivalent to choosing Frame/Modify Frame Layout/Make Default.

Do not call this function without having called the FrameModInit function previously.

The FrameModFinish function must be called after this function to accept the modifications.

Syntax

SetFrameDefaults(BorderWhere, ThickType, PosType, Units, ShadeType, BackType, DefType, LeftMargin, BottomMargin, TopMargin, RightMargin, ShadowLeft, ShadowTop, ShadowRight, ShadowBottom, ShadowColor, Rounded)

BorderWhere is the lines around a frame. It is one of the following values:

(1) - All
(2) - Left
(4) - Right
(8) - Top
(16) - Bottom

ThickType is the thickness of the border. It is one of the following values:

Hairline (1) - Hairline
OnePoint (2) - One point rule
TwoPoint (3) - Two point rule
ThreePoint (4) - Three point rule
FourPoint (5) - Four point rule
FivePoint (6) - Five point rule
SixPoint (7) - Six point rule
DoubleOnePoint (8) - Parallel one point rule
DoubleTwoPoint (9) - Parallel two point rule
ThreeLines (10) - Hairline above and below a two point rule
HairBelow (11) - Hairline below a three point rule
HairAbove (12) - Hairline above a three point rule

PosType is the position of the border around a frame. It is one of the following values:

(1) - Middle
(2) - Inside
(3) - Outside
(5) - Close to outside

You can only choose one value for the PosType parameter.

Units is the type of measurement and can be one of the following:

Inches (1) - units set to inches
CM (2) - units set to centimeters
Picas (3) - units set to picas
Points (4) - units set to points

ShadeType is the line color.

BackType is the background color.

DefType is a setting based on: how the text should wrap around a frame, whether a frame is transparent or opaque, has square or round corners, where it is placed on a page, and whether a macro is assigned to a frame. It is one of the following values:

Opaque (64) - Hide text or picture behind frame

Wraparound (128) - Display text above, below, to the left, or to the right of the frame

RepeatFrame (256) - Repeat frame on multiple pages. To repeat on all pages, do not use in combination with RepeatEven or RepeatOdd.

TextFrame (512) - Always use in combination with other values. It is a required value.

RepeatOdd (8192) - Repeat frame on odd pages. Use with the RepeatFrame value.

Borders (65536) - Use if frame has borders

NoWrapBeside (131072) - Display text above and below frame, but not to the left or right of the frame

AnchorFrame (524288) - Used to anchor frame in its current position or to a carriage return. You cannot use any repeat values with this value.

RepeatEven (4194304) - Repeat frame on even pages. Use with the RepeatFrame value.

RunMacro (134217728) - Run a macro each time the frame is selected

You can add the values together to get the DefType parameter. The syntax is "N &0x300c0", where "N" is the DefType value.

LeftMargin is the desired left margin of the frame.

TopMargin is the desired top margin of the frame.

RightMargin is the desired right margin of the frame.

BottomMargin is the desired bottom margin of the frame.

The value for all parameters except Units should be given in twips (1 inch = 1440 twips). Multiply the desired number of inches by 1440 to determine the value in twips.

Rounded is the amount that the corners are rounded, in percent. (100% = circle).

ShadowColor is the value assigned to the colors. It is one of the following values:

Red - 255

Orange - 33279

Yellow - 65535

Green - 65280

Cyan - 16776960

MedBlue - 16744448

Blue - 16727905

Purple - 16711809

Magenta - 16711935

Pink - 8388863

White - 16777215

Black - 0

The following four parameters determine the distance of the shadow from a specific side of the frame. They are either zero or positive integers. Multiply the desired distance in inches by 1440 to determine the value in twips. They are one of the following values or may be a custom value:

None (0) - No shadow

Shallow (57) - Shallow shadow

Normal (100) - Normal shadow

Deep (172) - Deep shadow

ShadowLeft is the distance that the shadow is offset from the left side of the frame in twips (1 inch = 1440 twips).

ShadowTop is the distance that the shadow is offset from the top of the frame in twips (1 inch = 1440 twips).

ShadowRight is the distance that the shadow is offset from the right side of the frame in twips (1 inch = 1440 twips).

ShadowBottom is the distance that the shadow is offset from the bottom of the frame in twips (1 inch = 1440 twips).

Return Value

This function returns 1.

Example

```
FUNCTION Example()
SetFrameDefaults(0 1 1 1 0 16777215 0 0 0 0 0 0 0 0 0 0 0)
END FUNCTION
```

See also:

FrameModInit FrameModBorders FrameModLines FrameModType FrameModFinish

This function assigns a value stored in a local array variable element to a global array variable element.

Syntax

SetGlobalArray(Name, Index, Value)

Name is the name or the ID number of the global array to set.

Index is the element of the array in which to place the value.

Value is the value that should be stored in the array element.

Return Value

1 (TRUE) if the value was successfully set.

0 (FALSE) if the value could not be set. If you attempt to set a global variable that does not exist, a run-time error results.

Example

```
FUNCTION Example()
AllocGlobalVar("Names", 5)'Allocate space for a 5 element global variable
AllocGlobalVar("Numbers", 5)'Allocate space for a 5 element global variable
AllocGlobalVar("YourName", 1)'Allocate space for a single element global variable.
FOR I = 1 to 5'Do the following 5 times.
    SetGlobalArray("Names", I, Query$("Enter Name Number {I}"))
    'Fill a global array with the return from QUERY
    SetGlobalArray("Numbers", I, (100/I))'Fill a global array with a number
NEXT
NewName = Query$("What is your name?")'Query the user for his/her name
SetGlobalVar("YourName", NewName)'Set a global variable to that value
CALL Example2()'Call the following function
END FUNCTION

FUNCTION Example2()
Name = GetGlobalVar$("YourName")'Get the value of the global variable
Message("Your name is {Name}.")'Message that value in a box.
FOR I = 1 to 5'Do the following 5 times.
    TheirName = GetGlobalArray$("Names", I)'Get the value of the current element from the array
    TheirNumber = GetGlobalArray$("Numbers", I)'Get the value of the current element from the
array
    TYPE("Name #{I} is {TheirName}, and the number is {TheirNumber}.[Enter]")
    'Type the values to the screen.
NEXT
FreeGlobalVar("Names")'Clear the space for the first global array
FreeGlobalVar("Numbers")'Clear the space for the second global array
FreeGlobalVar("YourName")'Clear the space for the global variable
END FUNCTION
```

See also:

AllocGlobalVar FreeGlobalVar GetGlobalVar$ GetGlobalArray$ SetGlobalVar Global Variables Variables GetGlobalVarCount GetGlobalVarNames

This function assigns the value stored in a single element local variable to a global variable.

Syntax

SetGlobalVar(Name, Value)

Name is the name or the ID number of the global variable to set.

Value is the value to store into the global variable.

Return Value

1 (TRUE) if the value was successfully set.

0 (FALSE) if the value could not be set. If you attempt to set a global variable that does not exist, a run-time error results.

Example

```
FUNCTION Example()
Again:
Count = GetGlobalVarCount()
IF Count < 1
    Count = 1
ELSE
    DIM Globals(Count)
    GetGlobalVarNames(&Globals)
    FillEdit(9000, &Globals)
ENDIF
Box = DialogBox(".", "ExampleBox")
IF Box = 0
    EXIT FUNCTION
ELSEIF Box = 3 AND GetDialogField$(9000) != ""
    FreeGlobalVar(GetDialogField$(9000))
ELSEIF Box = 4 AND GetDialogField$(9000) != ""
    Message(GetGlobalVar$(GetDialogField$(9000)))
ELSEIF Box = 5
    AllocGlobalVar(Query$("Name for new global variable:"), Query$("Number of elements?"))
ELSEIF Box = 6 AND GetDialogField$(9000) != ""
    SetGlobalVar(GetDialogField$(9000), Query$("What do you want in it?"))
ENDIF
GoTo Again
END FUNCTION

DIALOG ExampleBox
-2134376448 7 98 20 160 102 "" "" "Global Variables"
FONT 6 "Helv"
116 4 40 14 2 1342242817 "button" "Done" 0
116 20 40 14 3 1342242816 "button" "Free" 0
116 36 40 14 4 1342242816 "button" "Show..." 0
116 52 40 14 5 1342242816 "button" "Allocate..." 0
116 68 40 14 6 1342242816 "button" "Set..." 0
6 14 102 82 9000 1352728579 "listbox" "" 0
8 4 98 8 1000 1342177280 "static" "Currently Allocated Globals:" 0
END DIALOG
```

See also:

AllocGlobalVar FreeGlobalVar GetGlobalVar$ GetGlobalArray$ SetGlobalArray Global Variables Variables GetGlobalVarCount GetGlobalVarNames

This function sets the default path for the icon files. Choosing this function is equivalent to choosing Tools/User Setup/Paths.

Syntax

SetIconPath(NewPath)

NewPath is the desired new path for the icon files.

Return Value

1 (TRUE) if the path was successfully set.

-2 (GeneralFailure) if the path could not be set.

Example

```
FUNCTION Example()
SetIconPath("c:\amipro\icons")
END FUNCTION
```

See also:

SetBackPath SetDocPath SetMacroPath UserSetup SetStylePath SetDefPaths

This function sets the icon size of the icon set. Choosing this function is equivalent to choosing Tools/SmartIcons/Icon Size. A macro must be edited to insert this non-recordable function.

Syntax

SetIconSize(Size)

Size is one of the following values:

(1) - Small (EGA)
(2) - Medium (VGA)
(3) - Large (Super VGA)

Return Value

1 (TRUE) if the icon size changes.

0 (NoAction) if no action is taken.

Example

```
FUNCTION Example()
SetIconSize(2)
END FUNCTION
```

See also:

ChangeIcons IconBottom IconFloating IconRight IconTop IconCustomize ShowIconBar HideIconBar

This function sets the file name used to generate an index. It records the index file name for use in the Tools/TOC, Index dialog box. It also records the index file name for use in the File/Master Document/Options dialog box. Choosing this function is equivalent to choosing File/Master Document/Options.

This function should be used prior to the Generate function when generating an index.

Syntax

SetIndexFile(Separators, IndexFile)

Separators indicates whether or not alphabetic separators are used in the index. It is one of the following values:

Yes (1) - Include alphabetic separators.

No (2) - Do not include alphabetic separators.

IndexFile is the file name for the index file. For NewWave, the parameter must be the full path to the object name.

To allow the user to view the TOC, Index dialog box and enter the index file name: **Generate**

To allow the user to view the Master Documents Options dialog box and enter the index file name: **MasterDocOpts**

Return Value

This function returns 1.

Example

```
FUNCTION Example()
docpath=GetDocPath$()
' do not include alpha separators
SetIndexFile (2, "{docpath}test.sam")
END FUNCTION
```

See also:

Generate SetTOCFile

This function sets the default path for macro storage. Choosing this function is equivalent to choosing Tools/User Setup/Paths.

Syntax

SetMacroPath(NewPath)

NewPath is the desired new default path for macros.

Return Value

1 (TRUE) if the macro path was set.

-2 (GeneralFailure) if the macro path could not be set.

Example

```
FUNCTION Example()
SetMacroPath("C:\Amipro\Macros")
END FUNCTION
```

See also:

SetStylePath SetDocPath SetBackPath SetIconPath UserSetup SetDefPaths

This function determines which files comprise a master document. It makes the current file a master document. Choosing this function is equivalent to choosing File/Master Document and including the files for the master document.

Syntax

SetMasterFiles(Size, Count, File1[, File2...])

Size is the amount of buffer space allocated to hold the files. It is determined by adding up all of the characters in each filename, plus the number of files, plus one.

Count is the number of files in the master document.

File1 is the name of the first file in the list of master documents. If the file does not exist in the current documents directory, the full path to the file must be used. For NewWave, the parameter must be the full path to the object name.

File2 and any other files are treated the same as file1, above.

To display the Master Document dialog box and allow the user to specify the files: **MasterDoc**

Return Value

1 (TRUE) if the master files list was set.

-2 (GeneralFailure) if the master files list could not be set.

Example

```
FUNCTION Example()
size = 1 + len("TEST.SAM") + 1
SetMasterFiles(size, 1, "TEST.SAM")
END FUNCTION
```

See also:

GetMasterFiles GetMasterFilesCount

This function assigns the named paragraph style to the current paragraph. Choosing this function is equivalent to selecting a paragraph style name from the status bar or the styles box.

Syntax

SetStyle(Style)

Style is the name of a paragraph style in the current paragraph style sheet or document.

To display the Styles Box and allow the user to select the name of the paragraph style to be used: **SelectStyle**

Return Value

This function returns 1.

Example

```
FUNCTION Example()
curstyle = GetStyleName$()
newstyle = "Number List"
SetStyle(newstyle)
TYPE("This is what the {newstyle} style looks like.[Enter]")
SetStyle(curstyle)
END FUNCTION
```

See also:

GetStyleName$ ModifyStyle SelectStyle ShowStylesBox ToggleStylesBox

This function sets the default path for style sheet storage. Choosing this function is equivalent to choosing Tools/User Setup/Paths.

Syntax

SetStylePath(NewPath)

NewPath is the desired new path for paragraph style sheets.

Return Value

1 (TRUE) if the path was successfully set.

-2 (GeneralFailure) if the path could not be set.

Example

```
FUNCTION Example()
Backup = Query$("Please enter the desired backup path:" GetBackPath$())
Docs = Query$("Please enter the desired document path:" GetDocPath$())
Styles = Query$("Please enter the desired styles path:" GetStylePath$())
SetDocPath(Docs)
SetStylePath(Styles)
SetBackPath(Backup)
END FUNCTION
```

See also:

GetBackPath$ GetDocPath$ GetStylePath$ SetBackPath SetDocPath UserSetup SetDefPaths SetIconPath

This function sets the destination file for a table of contents that was generated from the current file. It records the information for the TOC file name in the Tools/TOC, Index dialog box . It also records the information for the File/Master Document/Options dialog box. Choosing this function is equivalent to choosing File/Master Document/Options.

This function should be used prior to the Generate function when creating a table of contents.

This function does not check the path to see if it is valid.

Syntax

SetTOCFile(TOCFile)

TOCFile is the filename for the generated table of contents. For NewWave, the parameter must be the full path to the object name. To allow the user to view the TOC, Index dialog box and enter the TOC file name: **Generate.** To allow the user to view the Master Documents Options dialog box and enter the TOC file name: **MasterDocOpts**

Return Value

This function returns 1.

Example

```
FUNCTION Example()
SetTOCFile("TestToc.sam")
END FUNCTION
```

See also:

Generate MasterDocOpts

This function sets the options for a generated table of contents for the current file. Choosing this function is equivalent to choosing File/Master Doc/Option/TOC Options from the menu. It is also equivalent to choosing Tools/TOC, Index/TOC Options from the menu.

This function replaces the TOCOptions function that is available in Ami Pro 3.0.

Syntax

SetTOCOpts(Style1, Sep1, Style2, Sep2, Style3, Sep3, TOCOpts)

Style1 is the paragraph style for level 1 TOC entries.

Sep1 is the character to separate level 1 TOC entries from the page numbers for the pages where the entries are located.

Style2 is the paragraph style for level 2 TOC entries.

Sep2 is the character to separate level 2 TOC entries from the page numbers for the pages where the entries are located.

Style3 is the paragraph style for level 3 TOC entries.

Sep3 is the character to separate level 3 TOC entries from the page numbers for the pages where the entries are located.

The separator character may be a period, hyphen, underline, or space.

TOCOpts is used to set options for using page numbers and aligning the page numbers for each TOC level. It is one of the following values:

Key1Page (1) - Include page numbers for level 1 TOC entries
Key1Align (2) - Include page numbers for level 1 TOC entries at the right margin.
Key1Page (4) - Include page numbers for level 2 TOC entries
Key1Align (8) - Include page numbers for level 2 TOC entries at the right margin.
Key1Page (16) - Include page numbers for level 3 TOC entries
Key1Align (32) - Include page numbers for level 3 TOC entries at the right margin.

Values may be added together.

To display the TOC Options dialog box and allow the user to select the parameters: **SetTOCOpts**

Return Value

1 (TRUE) if the options were set.

0 (UserCancel) if the user canceled the function.

Example

```
FUNCTION Example()
SetTOCOpts("Header1", ".", "Header2", ".", "SubHead", ".", 1)
END FUNCTION
```

See also:

Generate, SetTOCFile

This function shares the NewWave object in the selected frame. Choosing this function is equivalent to choosing Edit/Share.

A frame containing the NewWave object must be selected. The function shares the NewWave object in the selected frame.

Syntax

Share()

Return Value

1 (TRUE) if the object was shared.

-2 (GeneralFailure) if the object was not shared.

Example

```
FUNCTION Example()
TYPE ("[CtrlHome]") 'go to top of doc
GoToCmd(4 2 1) 'select first frame
Share()
END FUNCTION
```

See also:

CreateANew ImportText IsNewWave ListObjects NWGetContainerCount NWGetContainerNames NWGetCurrentContainer NWGetCurrentObject$ NWGetObjectCount NWGetObjectNames NWGetParent NWReferenceToFile$ ObjectAttributes OpenObject SaveAsMaster SaveAsObject ShowLinks

This function displays a menu bar and its menus.

Syntax

ShowBar(BarID)

BarID is the identification number of the menu bar returned from the AddBar function. To use the default Ami Pro menu bar, use 1.

Return Value

1 (TRUE) if the menu bar was successfully displayed.

0 (FALSE) if the bar could not be shown, or if an invalid BarID was used.

Example

```
FUNCTION Example()
StatusBarMsg("Adding menu bar and items....")'Notify the user what we are doing
MacFile = GetRunningMacroFile$()'Get the name of this macro file.
BarID = AddBar()'Add a new menu bar.
IF BarID > 0'If it was added,
    AddMenu(BarID, "&File")'Add a File menu to it.
    AddMenu(BarID, "&Macros")'Add a Macros menu to it
    AddMenuItem(BarID, "&File", "&New", New, "Begin a new document")
    'Add an item to the file menu
    AddMenuItem(BarID, "&File", "&Long Menus", "{MacFile}!Back()", "Restore Ami Pro Menu Bar.")
    'Add an item to the file menu.
    AddMenuItem(BarID, "&Macros", "&Edit", MacroEdit, "Edit a macro file.")
    'Add an item to the macros menu.
    AddCascadeMenu(BarID,  "&Macros", "&Run")'Add a cascade menu to the macros menu
    AddCascadeMenuItem(BarID, "&Macros", "&Run", "&Run current", "RUNCURR.SMM", "Run the
displayed macro")
    'Add an item to the cascade menu on macros.
    AddCascadeMenuItem(BarID, "&Macros", "&Run", "&Run another", MacroPlay, "Run another macro")
    'Add an item to the cascade menu on macros.
    AddMenuItem(BarID, "&Macros", "&Options", MacroOptions, "Choose Macro options.")
    'Add an item to the macros menu.
    ShowBar(BarID)'Show the newly created menu bar.
ENDIF
StatusBarMsg("")'Restore the status bar.
END FUNCTION

FUNCTION Back()
ShowBar(1)'Restore the original Ami Pro menu bar.
END FUNCTION
```

See also:

AddBar AddMenu AddMenuItem AddMenuItemDDE ChangeMenuAction CheckMenuItem DeleteMenu DeleteMenuItem GrayMenuItem RenameMenuItem

This function displays the set of SmartIcons in its default location on the screen. Choosing this function is equivalent to choosing View/Show SmartIcons.

Syntax

ShowIconBar()

Return Value

1 (TRUE) if the icon bar is displayed.

0 (NoAction) if no action was taken because the icon bar was already displayed.

Example

```
FUNCTION Example()
ReturnValue = HideIconBar()
IF ReturnValue != 1
    IF Decide("Icon bar already hidden! Would you like it displayed?")
        ShowIconBar()
    ENDIF
ENDIF
END FUNCTION
```

See also:

ViewPreferences HideIconBar ShowTabRuler ToggleIconBar IconBottom IconCustomize IconFloating IconLeft IconRight IconTop ToggleIconBar

This function opens a container that has a linked copy of the selected object. Choosing this function is equivalent to choosing Objects/Show Links.

An Ami Pro object must be open and a NewWave object must be selected.

Syntax

ShowLinks(Path)

Path is the full path to the NewWave container.

To display the Show Links dialog box and allow the user to select the parameters: **ShowLinks**

Return Value

1 (TRUE) if the container was opened.

0 (UserCancel/FALSE) if the user canceled the function.

-2 (GeneralFailure) if the container could not be opened.

Example

```
FUNCTION Example()
GoToCmd(4 2 1) 'select frame containing object
ShowLinks("Memo Folder") 'opens the Memo Folder
ShowLinks("January Folder\Memo Folder2") 'opens the  memo  folder which is inside the January
Folder
END FUNCTION
```

See also:

CreateANew ImportText IsNewWave ListObjects NWGetContainerCount NWGetContainerNames NWGetCurrentContainer NWGetCurrentObject$ NWGetObjectCount NWGetObjectNames NWGetParent NWReferenceToFile$ ObjectAttributes OpenObject SaveAsMaster SaveAsObject Share

This function displays the Styles Box. Choosing this function is equivalent to choosing View/Show Styles Box.

Syntax

ShowStylesBox()

Return Value

1 (TRUE) if the Styles box is displayed.

0 (NoAction) if no action was taken because the Styles box was already displayed.

Example

```
FUNCTION Example()
ReturnValue = HideStylesBox()
IF ReturnValue != 1
    IF Decide("Styles Box already hidden! Would you like it displayed?")
        ShowStylesBox()
    ENDIF
ENDIF
END FUNCTION
```

See also:

ViewPreferences HideStylesBox ShowIconBar ShowTabRuler ToggleStylesBox

This function displays the tab ruler at the top of the screen. Choosing this function is equivalent to choosing View/Show Ruler.

Syntax

ShowTabRuler()

Return Value

1 (TRUE) if the tab ruler is displayed.

0 (NoAction) if no action was taken because the tab ruler was already displayed.

Example

```
FUNCTION Example()
ShowTabRuler()
END FUNCTION
```

See also:

ViewPreferences HideTabRuler ShowIconBar ToggleTabRuler

This function allows the macro programmer to debug macros by stepping through each macro statement one line at a time.

SingleStep uses two modes. In the first of these, if you have the macro to be debugged open in a window and a SingleStep statement in it, the line currently executing is highlighted in the macro file and a modeless dialog box displays. If the macro to be debugged is not open, a dialog box displays the currently running statement.

If the macro is in a Multiple Document Interface (MDI) window you can:

Resume - Removes the dialog box and continues to the next break point, continues to the next SingleStep(1), or goes to the end of the macro.

Single Step - Executes the current instruction then pauses again.

Step Through - If a CALL statement is made, the macro from the CALL statement executes to completion and control returns to the original macro.

Variables - This brings up a modal dialog box showing the list of local variables currently set in the currently running macro. You may change the contents of a variable here at run time. If the variable is an array, the number of elements in the array display and may not be changed.

Break Points - This brings up a modal dialog box showing the list of currently active breakpoints. The Break Points list box lists the filename, an exclamation point, the function name, a period, and then the line number within that function.

Set BP - This sets a break point on the current line. This assumes, of course, that you are still in the macro window or in another window that contains a running macro.

Break points cannot be set on empty lines or commented lines or lines with ELSE, WEND, or ENDIF. Break points may only be placed on a line with an executable code statement and it must be after a SingleStep() function in the macro.

Cancel - This cancels the macro. If your macro has an ONCANCEL statement, it is executed.

If your macro modifies open documents, make sure these documents are not the currently executing macros. Note that when the macro first pauses, the line about to be executed is selected and can easily be deleted.

Syntax

SingleStep(State)

State determines whether single step mode is off (0) or on (1).

Return Value

This function does not return a value.

Example

```
FUNCTION Example()
SingleStep(1)
FOR I = 1 to 10
    TYPE("Hello...")
NEXT
END FUNCTION
```

See also:

AnswerMsgBox Messages UserControl IgnoreKeyboard DebuggingMacros KeyInterrupt

This function sets the column and row width and height, along with the gutter width and height. Choosing this function is equivalent to choosing Table/Column/Row Size. If the automatic row height option is set for the table, the row height option is ignored.

Syntax

SizeColumnRow(ColWidth, ColGutter, RowHeight, RowGutter)

ColWidth is the width of the column in twips.

ColGutter is the width of the column gutter in twips.

RowHeight is the height of the row in twips.

RowGutter is the height of the row gutter in twips.

Multiply the desired number of inches by 1440 to determine the value in twips. (1 inch = 1440 twips).

To display a dialog box to allow the user to size columns and rows: **SizeColumnRow**

Return Value

1 (TRUE) if the rows and columns were sized.

0 (UserCancel) if the user canceled the function.

-2 (GeneralFailure) if the rows and columns were not sized.

Example

```
FUNCTION Example()
SizeColumnRow(720, 0, 180, 0)
END FUNCTION
```

See also:

DeleteColumnRow InsertColumnRow

This function sets the small capitalization for selected text or for all following text if no text is selected. It acts as a toggle, turning off small caps if it is currently on, and turning on small caps if it is currently off. Choosing this function is equivalent to choosing Text/Caps/SmallCaps.

Syntax

SmallCaps()

Return Value

1 (TRUE) if the attribute was set.

-6 (NoMemory) if the function failed because of insufficient memory.

Example

```
FUNCTION Example()
WHILE "" = CurShade$()
    UserControl("Select the text to modify, then choose Resume...")
WEND
SmallCaps()
END FUNCTION
```

See also:

InitialCaps LowerCase UooerCase

This function sorts the selected text or the entire document. Choosing this function is equivalent to choosing Tools/Sort.

Syntax

Sort(Key1, Key2, Key3, NumParas, Delimiter, Options)

Key1 is the field number of the first key.

Key2 is the field number of the second key.

Key3 is the field number of the third key.

Key1, Key2, and Key3 are the field numbers of the keys to use for the sort. If fewer than three keys are used, the unused keys must have values, but they are ignored. The NumParas parameter is the number of paragraphs (or rows, if sorting a table) that should be considered a single record.

NumParas is the number of paragraphs (or rows, if sorting a table) that should be considered a single record.

Delimiter is the field delimiter. This parameter determines the field delimiter, if it is not a TAB symbol. If the delimiter should be a TAB, this option should be set in the options, and the value of the delimiter is ignored. If a quote mark is the delimiter, its ANSI value, 34, should be used (for example, CHR$(34)).

Options is a flag containing the sort options and may be one or more of the following:

Ascending (1) - Sort in ascending order. If this option is not set, the sort is in descending order.

KeyOneAlpha (2) - Sort key 1 is alphanumeric. If this option is not set, key 1 is assumed to be numeric.

KeyTwoAlpha (4) - Sort key 2 is alphanumeric. If this option is not set, key 1 is assumed to be numeric.

KeyThreeAlpha (8) - Sort key 3 is alphanumeric. If this option is not set, key 1 is assumed to be numeric.

KeyOneFirst (16) - Sort using the first word of key 1

KeyOneSecond (32) - Sort using the second word of key 1

KeyTwoFirst (64) - Sort using the first word of key 2

KeyTwoSecond (128) - Sort using the second word of key 2

KeyThreeFirst (256) - Sort using the first word of key 3

KeyThreeSecond (512) - Sort using the second word of key 3

The KeyXFirst or KeyXSecond parameters should be used to only sort on the first or second words of the key field. If these parameters are not used, the data is sorted based on the entire contents of the field.

UseKeyTwo (1024) - Use values for key 2, if not set, parameters for key 2 are ignored

UseKeyThree (2048) - Use values for key 3, if not set, parameters for key 3 are ignored

AnsiSort (4096) - Sorts using ANSI sorting sequence. If not set, sorts using IBM PC sort sequence.

This option should always be set for Ami Pro files.

TabDelimited (8192) - Sort fields are tab delimited

TableSort (16384) - Sort is in a table

These options should be added together. Options for keys 2 and 3 do not need to be set if they are not going to be used. The ANSISort parameter should always be set for Ami Pro files. If the fields are tab delimited, the TabDelimited parameter should be set. If it is set, the Delimiter parameter is ignored.

To display the Sort dialog box and allow the user to determine sort options: **Sort**

Return Value

1 (TRUE) if the sort was successful.

0 (UserCancel) if the user canceled the function.

-6 (NoMemory) if the function failed because of insufficient memory.

Example

```
FUNCTION Example()
IF "" = CurShade$()
    UserControl("Shade on the area to sort, or none for the entire stream, then choose
Resume...")
ENDIF
Sort(1, 2, 3, 1, "", 1)
END FUNCTION
```

See also:

Merge

This function sets line spacing for the current paragraph or selected paragraphs. Choosing this function is equivalent to choosing Text/Spacing.

Syntax

Spacing(Amount)

Amount is the new spacing to set. The Amount parameter determines fixed line spacing according to the following list:

SpaceStyle (0) - Use the line spacing specified in the paragraph style

SpaceSingle (-1) - Use single spacing

SpaceOneHalf (-2) - Use 1 1/2 line spacing

SpaceDouble (-3) - Use double spacing

To set custom line spacing, the Amount parameter should be a positive integer representing line spacing in twips. To determine twips from points, multiply the point size by 20.

To display the Spacing dialog box and allow the user to determine the new spacing: **Spacing**

Return Value

1 (TRUE) if the line spacing was successfully changed.

0 (UserCancel) if the user canceled the function.

-6 (NoMemory) if the function failed because of insufficient memory.

Example

```
FUNCTION Example()
Spacing(-2)
END FUNCTION
```

See also:

FontChange NormalText Center Justify LeftAlign RightAlign

This function sets special effects for selected text or for all following text if no text is selected. Choosing this function is equivalent to choosing Text/Special Effects.

Syntax

SpecialEffects(Which, OverStrikeChar)

Which is a flag representing the special effects to use and can be one of the following:

Superscript (64) - Superscript text
Subscript (128) - Subscript text
DoubleUnderline (256) - Double underline text
Strikethrough (512) - Strikethrough text
Overstrike (1024) - Overstrike text with character given in OverStrike parameter

OverStrikeChar is the character used to overstrike the text. The Overstrike value in the Which parameter must also be set.

More than one special effect can be chosen by adding the values for the desired effects together.
To display the Special Effects dialog box and allow the user to determine special effects settings: **SpecialEffects**

Return Value

1 (TRUE) if the special effects were applied.

0 (UserCancel) if the user canceled the function.

-6 (NoMemory) if the function failed because of insufficient memory.

Example

```
FUNCTION Example()
SpecialEffects(Superscript + DoubleUnderline, "")
END FUNCTION
```

See also:

NormalText Underline WordUnderline

This function allows you to check the spelling of words or documents. Choosing this function is equivalent to choosing Tools/Spell Check.

Syntax

SpellCheck(Options)

Options is a flag representing the options to use for the spell check, and can be one or more of the following:

SpellBegin (1) - Starts the spell checking from the beginning of the document.

CurrentStream (2) - Spell checks only the current stream. Do not set this flag if you want all streams to be spell checked.

NoDoubleWord (4) - Ignores repeated words.

IgnoreNums (8) - Ignores words containing numbers.

IgnoreInitCaps (16) - Checks the spelling of words that do not begin with a capital letter.

TurboCheck (32) - Speeds up the spell checking by only checking the documents that have been flagged as changed since the last spell check.

SpellAlt (64) - Checks for alternate spelling entered by the user.

To display the Spell Check dialog box and allow the user to select the parameters: **SpellCheck**

Return Value

1 (TRUE) if the spell check was done.

0 (UserCancel) if the user canceled the function.

-2 (GeneralFailure) if the spell check could not be done.

Example

```
FUNCTION Example()
SpellCheck(1)
END FUNCTION
```

See also:

Thesaurus

This function changes the current view level to standard view. Choosing this function is equivalent to choosing View/Standard.

Syntax

StandardView()

Return Value

This function returns 1.

Example

```
FUNCTION Example()
StandardView()
END FUNCTION
```

See also:

EnlargedView FacingView FullPageView GetViewLevel LayoutMode CustomView

This function displays a message in the Ami Pro status bar. The macro does not pause. The message can be reset by calling this function with the null string ("") as its parameter.

Syntax

StatusBarMsg(Message)

Message is the string of text to be displayed in the program's status bar. If the null string ("") is sent, the default Ami Pro status bar is restored.

Return Value

1 (TRUE) if the message was displayed.

-2 (GeneralFailure) if the message could not be displayed.

Example

```
FUNCTION Example()
IgnoreKeyboard(1)
HourGlass(1)
StatusBarMsg("Importing graphic file...")
ImportPicture("BMP", "C:\AMIPRO\ICONS\123W.BMP", ".BMP", 0)
StatusBarMsg("")
HourGlass(0)
IgnoreKeyboard(0)
END FUNCTION
```

See also:

HourGlass IgnoreKeyboard Messages AnswerMsgBox KeyInterrupt

This function concatenates strings together to make a longer string. Any number of strings can be appended together at a time.

Syntax

strcat$(Text1, Text2[, Text3...])

Text1 is a string to which Text2 is appended.

Text2 is a string which is appended to Text1.

Text3 is a string which is appended to Text1 and Text2.

Return Value

This function returns the new string.

Example

```
FUNCTION Example()
MacFile = GetRunningMacroFile$()
SetDlgCallBack(50, "{MacFile}!Message1")
Box = DialogBox(".", "ExampleBox")
IF Box = -1
    Message("Could not find dialog box!")
    EXIT FUNCTION
ELSEIF Box = 0
    EXIT FUNCTION
ENDIF
TYPE(GetDialogField$(8002))
END FUNCTION

FUNCTION Message1(hdlg, id, text)
Name = GetDlgItemText(hdlg, 8000)
Message("The contents of the first box are {Name}.")
Message("We will now fill the second box with the inverse of {Name}.")
FOR I = len(Name) to 1 step -1
    Name2 = strcat$(Name2, MID$(Name, I, 1))
NEXT
SetDlgItemText(hdlg, 8002, Name2)
END FUNCTION

DIALOG ExampleBox
-2134376448 8 106 38 160 54 "" "" "Sample Dialog Box"
FONT 6 "Helv"
50 6 62 12 8000 1350631552 "edit" "" 0
4 6 44 10 1000 1342177280 "static" "Your Name:" 0
50 20 62 12 1003 1342177287 "static" "" 0
4 22 44 10 1002 1342177280 "static" "Reversed:" 0
52 22 58 8 8002 1342177280 "static" "" 0
116 4 40 14 1 1342242817 "button" "OK" 0
116 20 40 14 2 1342242816 "button" "Cancel" 0
100 36 56 14 50 1342242816 "button" "&Run Example..." 0
END DIALOG
```

See also:

ASC strchr LCASE$ UCASE$ strfield$ MID$ LEN FormatNum$

This function searches the specified string for a character. It returns the location of the character within the string.

Syntax

strchr(Offset, Text, Char)

Offset is the location in the string to begin searching. Offset is one based. To search from the beginning of the string, use an offset of 1.

Text is a string that is searched for a specific character.

Char is the first character to be searched for.

Return Value

The number of characters into the string at which the desired character was found.

-1 if the desired character is not found in the string, the function returns.

Example

```
FUNCTION Example()
String = Query$("Enter the string:")
Offset = 0
Char = Query$("What char to search for?")
Position = Strchr(Offset, String, Char)
Message(Position)
END FUNCTION
```

See also:

ASC CHR$ strcat$ LCASE$ UCASE$ strfield$ MID$ LEN FormatNum$ Instr Left$ Right$

This function is used to extract individual fields from a field delimited string. The function assumes a record format where variable length fields are separated from each other with a unique character.

Modified in 3.0 In Ami Pro release 2.0, the function only returns the desired field or a null value.

Syntax

strfield$(Text, FieldNumber, Separator)

Text is the string from which the field is extracted.

FieldNumber is the number of the field to be extracted (1 = first field).

Separator is the character which separates the fields.

Return Value

This function returns the extracted field.

-1 if there are not enough fields in the string.

The null string ("") if the desired field is empty.

Example

```
FUNCTION Example()
LayoutMode()
IF not IsFrameSelected()
    Pos = CursorPosition$()
    x = strfield$(Pos, 1, ",")
    y = strfield$(Pos, 2, ",")
    AddFrame(x, y, (x + 1440), (y - 1440))
ENDIF
DrawingMode()
END FUNCTION
```

See also:

ASC CHR$ strcat$ strchr LCASE$ UCASE$ MID$ LEN FormatNum$

This function is used to manage paragraph styles. Choosing this function is equivalent to choosing Style/Style Management.

The StyleManageInit function must be used prior to this function. The StyleManageFinish function must be used immediately after this function.

Syntax

StyleManageAction(Style, Action, FuncNo)

Style is the name of the managed paragraph style.

Action is used to select the desired action to perform on the paragraph style.

It is one of the following values:

FuncNo (1) - Assigns a function key to a paragraph style.

DocToSty (2) - Moves a paragraph style from the document to the style sheet.

StyToDoc (4) - Moves a paragraph style from the style sheet to the document.

Remove (8) - Removes a paragraph style. You can remove any paragraph style from the document style list and any style except Body Text from the style sheet style list.

Revert (16) - Undo changes made to a paragraph style. The style reverts to the formatting information stored in the style sheet.

StyList (32) - The paragraph style being affected in the style sheet. Use this value if removing, reverting, or changing the function number of a style in the style sheet.

FuncNo is the function key assigned to the paragraph style in the StyleName parameter. The range for this number is 2 – 16.

Return Value

This function returns 1.

Example

```
FUNCTION Example()
StyleManageInit()
action = StyList + FuncNo
' assign body text F3 as its function key
StyleManageAction("Body Text", action,  3)
StyleManageFinish()
END FUNCTION
```

See also:

StyleManagement StyleManageInit StyleManageFinish

This function must be used after the StyleManageAction function, which manages styles. Choosing this function is equivalent to accepting changes entered by choosing Style/Style Management.

This function does not automatically manage styles.

Syntax

StyleManageFinish()

To display the StyleManagement dialog box and allow the user to manage paragraph styles:
StyleManagement

Return Value

This function returns 1.

Example

```
FUNCTION Example()
StyleManageInit()
action = StyList + FuncNo
' assign body text F3 as its function key
StyleManageAction("Body Text", action,  3)
StyleManageFinish()
END FUNCTION
```

See also:

StyleManagement StyleManageInit StyleManageFinish

This function is used to prepare Ami Pro for style management changes. Choosing this function is equivalent to initializing changes made when choosing Style/Style Management.

It must be used prior to the StyleManageAction function.

Syntax

StyleManageInit()

Return Value

This function returns 1.

Example

```
FUNCTION Example()
StyleManageInit()
action = StyList + FuncNo
' assign body text F3 as its function key
StyleManageAction("Body Text", action,  3)
StyleManageFinish()
END FUNCTION
```

See also:

StyleManagement StyleManageAction StyleManageFinish

This function allows the user to view the Style Management dialog box and manage paragraph styles. Choosing this function is equivalent to choosing Style/Style Management.

This function does not manage paragraph styles automatically.

Syntax

StyleManagement()

Return Value

1 (TRUE) if the changes were accepted.

0 (UserCancel) if the user canceled the function.

Example

```
FUNCTION Example()
StyleManagement()
END FUNCTION
```

See also:

ModifyStyle SaveAsNewStyle UseAnotherStyle StyleManageInit StyleManageAction StyleManageFinish

This function returns the starting and ending rows if Ami Pro is in a table. If a single cell or no cell is selected, the starting row equals the ending row. If you are not in a table, the passed variables are not changed.

Syntax

TableGetRange(&StartRow, &StartCol, &EndRow, &EndCol)

&StartRow contains the row the insertion point is in.

&StartCol contains the column the insertion point is in.

&EndRow contains the last selected row.

&EndCol contains the last selected column.

If a range is selected in the table, StartRow and StartCol are different from EndRow and EndCol.

If you are not in the tables mode, StartRow, StartCol, EndRow, and EndCol are not changed.

Return Value

1 (TRUE) if you are in tables mode, the variables are updated.

0 (FALSE) if you are not in tables mode.

Example

```
FUNCTION Example()
DEFSTR StartRow, StartCol, EndRow, EndCol;
Tables(1, TRUE, 4, 10)
'TableLayout(2, TRUE, 1440, 0, 0, 0, TRUE, TRUE, FivePoint, TRUE)
WHILE StartRow != 2
    TYPE("[Right]")
    TableGetRange(&StartRow, &StartCol, &EndRow, &EndCol)
WEND
TYPE("[SHIFTRight][SHIFTRight]")
ConnectCells()
TableLines(AllSides, 0, 0, OnePoint, 0)
END FUNCTION
```

See also:

Tables TableLayout

This function sets the layout of an existing table. Choosing this function is equivalent to choosing Table/Modify Table Layout.

Modified in 3.0 There is a new parameter, DisableMouse, in Ami Pro release 3.0.

Syntax

TableLayout(ChangeLayout, AutoHeight, ColWidth, ColGutter, RowHeight, RowGutter, Center, LineAround, LineStyle, HonorProtect, SpanPages, DisableMouse)

ChangeLayout is a flag indicating that this is a layout change. The ChangeLayout parameter should always be set to 2.

AutoHeight is a flag indicating whether the automatic row height option is enabled. The AutoHeight parameter should be set to 1 (On) if the height of rows should be allowed to grow automatically, or 0 (Off) if the row height should be fixed.

ColWidth is the width of the columns in twips.

ColGutter is the width of the column gutters in twips.

RowHeight is the height of the rows in twips.

If the automatic row height option is set, the option for row height is ignored.

RowGutter is the height of the row gutters in twips.

ColWidth, ColGutter, RowHeight and RowGutter should be set to the appropriate width for columns, rows, and gutters.

Multiply the desired number of inches by 1440 to determine the value in twips (1 inch = 1440 twips).

Center is a flag indicating whether a page table should be centered between the margins and can be one of the following:

1 (On) if the table should be centered on the page.

0 (Off) if the table is not to be centered on the page.

If the table is in a frame, this parameter is ignored, but should still be set.

LineAround is a flag indicating whether there should be a line around the table, and may be one of the following:

1 (On) if a line should be drawn around the table.

0 (Off) if no line should be drawn around the table.

LineStyle is the paragraph style to use for the line around the table and may be one of the following:

Hairline (1) - Hairline

OnePoint (2) - One point rule

TwoPoint (3) - Two point rule

ThreePoint (4) - Three point rule

FourPoint (5) - Four point rule

FivePoint (6) - Five point rule

SixPoint (7) - Six point rule

DoubleOnePoint (8) - Parallel one point rules

DoubleTwoPoint (9) - Parallel two point rules

ThreeLines (10) - Hairline above and below a two point rule

HairBelow (11) - Hairline below a three point rule

HairAbove (12) - Hairline above a three point rule

If no line should be drawn around the table, this parameter must be set, but the value of the parameter is ignored.

HonorProtect is a flag indicating whether protection should be honored in the table and may be one of

the following:

1 (On) if cells marked as protected should not be editable.

0 (Off) if protected cells can be edited.

SpanPages is a flag indicating whether data in a cell should continue to the next page without moving the entire row to the next page. The AutoHeight parameter must also be set to 1 if SpanPages is set to 1.

1 (On) - If rows should span pages

0 (Off) - If rows should not span pages

DisableMouse is a flag indicating whether the mouse can change the size of rows and columns.

1 (Yes) if the mouse should be disabled

0 (No) if the mouse should be enabled

To display the Table Layout dialog box and allow the user to set the table layout: **TableLayout**

Return Value

1 (TRUE) if the table layout was changed.

0 (UserCancel) if the user canceled the function.

-2 (GeneralFailure) if the layout could not be changed.

-6 (NoMemory) if the function failed because of insufficient memory.

Example

```
FUNCTION Example()
DEFSTR StartRow, StartCol, EndRow, EndCol;
Tables(1, TRUE, 4, 10)
'TableLayout(2, TRUE, 1440, 0, 0, 0, TRUE, TRUE, FivePoint, TRUE)
WHILE StartRow != 2
    TYPE("[Right]")
    TableGetRange(&StartRow, &StartCol, &EndRow, &EndCol)
WEND
TYPE("[SHIFTRight][SHIFTRight]")
ConnectCells()
TableLines(AllSides, 0, 0, OnePoint, 0)
END FUNCTION
```

See also:

ProtectCells Tables TableLines

This function sets the line and color options for a cell or group of cells. Choosing this function is equivalent to choosing Table/Lines & colors.

Syntax

TableLines(LineOn, LineOff, NoChange, LineStyle, ShadeType)

LineOn is a flag indicating which sides of the cell should have the new lines/colors and may be one or more of the following:

AllSides (1) - All sides of the cell
Outline (2) - The outline of the cell
LeftSide (4) - The left side of the cell
RightSide (8) - The right side of the cell
TopSide (16) - The top side of the cell
BottomSide (32) - The bottom side of the cell
CellShade (64) - The shading of the cell

Multiple sides of the cell can be specified by adding the individual values together.

LineOff is a flag indicating which sides of the cell should not have lines or colors and may be one or more of the following:

AllSides (1) - All sides of the cell
Outline (2) - The outline of the cell
LeftSide (4) - The left side of the cell
RightSide (8) - The right side of the cell
TopSide (16) - The top side of the cell
BottomSide (32) - The bottom side of the cell
CellShade (64) - The shading of the cell

Multiple sides of the cell can be specified by adding the individual values together.

NoChange is a flag indicating which sides of the cell should remain the same and may be one or more of the following:

None (0) - No sides of the cell
AllSides (1) - All sides of the cell
Outline (2) - The outline of the cell
LeftSide (4) - The left side of the cell
RightSide (8) - The right side of the cell
TopSide (16) - The top side of the cell
BottomSide (32) - The bottom side of the cell
CellShade (64) - The shading of the cell

Multiple sides of the cell can be specified by adding the individual values together.

The LineOn, LineOff, and NoChange parameters are equivalent to choosing check box states for each of the sides of the cell. Using the menu function, a checked box indicates that the new lines/colors should be applied to the element indicated by the box. An unchecked box indicates that no lines/colors should be applied to an element. A grayed box indicates that lines or colors are already applied to the element and that they should not be changed from what they already are.

LineStyle is a number representing the desired line paragraph style and may be one of the following:

Hairline (1) - Hairline
OnePoint (2) - One point rule
TwoPoint (3) - Two point rule
ThreePoint (4) - Three point rule

FourPoint (5) - Four point rule
FivePoint (6) - Five point rule
SixPoint (7) - Six point rule
DoubleOnePoint (8) - Parallel one point rules
DoubleTwoPoint (9) - Parallel two point rules
ThreeLines (10) - Hairline above and below a two point rule
HairBelow (11) - Hairline below a three point rule
HairAbove (12) - Hairline above a three point rule

ShadeType is a number representing the desired shading type and may be one of the following:

BlackShade (1) - Black background
SeventyFiveShade (2) - 75% black shading
FiftyShade (3) - 50% black shading
TwentyFiveShade (4) - 25% black shading
TenShade (5) - 10% black shading
BlueShade (6) - Blue background
RedShade (7) - Red background
MagentaShade (8) - Purple background
GreenShade (9) - Green background
YellowShade (10) - Yellow background
CyanShade (11) - Light blue background
WhiteShade (12) - White background

If no shading should be applied to the cell(s), this parameter should be set but the value of the parameter is ignored.

To display the Lines dialog box and allow the user to set the table line options: **TableLines**

Return Value

1 (TRUE) if the cell lines and colors were changed.

0 (UserCancel) if the user canceled the function.

-2 (GeneralFailure) if the lines could not be changed.

-6 (NoMemory) if the function failed because of insufficient memory.

Example

```
FUNCTION Example()
DEFSTR StartRow, StartCol, EndRow, EndCol;
Tables(1, TRUE, 4, 10)
'TableLayout(2, TRUE, 1440, 0, 0, 0, TRUE, TRUE, FivePoint, TRUE)
WHILE StartRow != 2
    TYPE("[Right]")
    TableGetRange(&StartRow, &StartCol, &EndRow, &EndCol)
WEND
TYPE("[SHIFTRight][SHIFTRight]")
ConnectCells()
TableLines(AllSides, 0, 0, OnePoint, 0)
END FUNCTION
```

See also:

TableLayout Tables

This function creates a table with the specified parameters. Choosing this function is equivalent to choosing Tools/Tables.

Syntax

Tables(Which, AutoHeight[, ColWidth, GutWidth, RowHeight, GutHeight, Center, LineAround, LineStyle, HonorProtect, SpanPages], NumCols, NumRows)

Which is a flag indicating what to do with this function and can be one of the following:

(1) - Creates only the table

(3) - Creates the table and includes the table layout information

AutoHeight is set if the height of the rows in the new table should be auto height. This function can be a 1 (On), or a 0 (Off).

ColWidth is the width of the columns to create, in twips.

GutWidth is the width of the column gutters to create, in twips.

RowHeight is the height of the rows to create, in twips.

This parameter is ignored if the AutoHeight parameter is set to 1.

GutHeight is the height of the row gutters to create, in twips.

Center is set if you want the table to be centered between the margins and may be a 1 (On) or a 0 (Off).

If this table is being placed in a frame, this parameter is ignored.

LineAround is used if you want a line to be placed around the table and may be a 1 (On) or a 0 (Off).

LineStyle is the style of the line to be placed around the table if the LineAround parameter is set to 1.

HonorProtect is set if you want honor protection set inside the table. It can be a 1 (On) or a 0 (Off).

SpanPages is set if you want rows to span pages (On) or if you want rows to be retained separately over page breaks (Off).

NumCols is the number of columns to place in this table.

NumRows is the number of rows to place in this table.

Return Value

1 (TRUE) if the table was created.

0 (UserCancel) if the user canceled the function.

-2 (GeneralFailure) if the table could not be created.

Example

```
FUNCTION Example()
DEFSTR StartRow, StartCol, EndRow, EndCol;
Tables(1, TRUE, 4, 10)
'TableLayout(2, TRUE, 1440, 0, 0, 0, TRUE, TRUE, FivePoint, TRUE)
WHILE StartRow != 2
    TYPE("[Right]")
    TableGetRange(&StartRow, &StartCol, &EndRow, &EndCol)
WEND
TYPE("[SHIFTRight][SHIFTRight]")
ConnectCells()
TableLines(AllSides, 0, 0, OnePoint, 0)
END FUNCTION
```

See also:

AddFrame TableLayout TableLines

This function adds, deletes, or modifies tabs, and changes the number of columns, indentions, and margins. Choosing this function is equivalent to specifying tab functions on the tab ruler.

The amount for indentions must be given in twips (1 inch = 1440 twips). Multiply the desired number of inches by 1440 to determine the value in twips.

Syntax

TabRuler(Command[, Parameters])

Command is the type of tab function to process and can be one of the following:

ChangeMargins (1) - Change the margins
ChangeAllIndent (2) - Change all the indents
ChangeFirstIndent (3) - Change the first indent
ChangeSecIndent (4) - Change the second indent
ChangeRightIndent (5) - Change the right indent
ChangeTabs (6) - Change the tabs
ChangeCols (7) - Change the number of columns

TabRuler(ChangeMargins, 1, Column, Offset1, Offset 2)

Column is the number of the column to modify in the horizontal ruler. To modify the vertical ruler set column to zero.

Offset1 is the amount to offset from the left side, in twips, for the horizontal ruler. It is the amount to offset from the top, in twips, for the vertical ruler. The offset on the horizontal ruler is the left margin and the offset on the vertical ruler is the top margin.

Offset2 is the amount to offset from the left side, in twips, for the horizontal ruler. It is the amount to offset from the top, in twips, for the vertical ruler. The offset on the horizontal ruler is the right margin and the offset on the vertical ruler is the bottom margin.

TabRuler(ChangeAllIndent, Offset)

Offset is the amount to set indention from the left margin, in twips.

TabRuler(ChangeFirstIndent, Offset)

TabRuler(ChangeSecIndent, Offset)

Offset is the amount to set indention from the left margin plus the Indent All indention, in twips.

TabRuler(ChangeRightIndent, Offset)

Offset is the amount to set indention from the right margin, in twips.

TabRuler(ChangeTabs, TabNumber, Action[, Offset, Type])

TabNumber is a number indicating which tab to add, delete, or modify. The TabNumber begins at 0.

Action specifies what action to perform on the selected tab.

(-1) - Delete tab
(-2) - Add tab
(-3) - Delete all tabs
(Positive number) - the new offset for a tab, in twips, from the left margin

If adding a tab or deleting a single tab, the **Offset** and **Type** parameters are required.

Offset is the offset for each tab, in twips, from the left margin.

Type is type for each tab and can be one of the following:

TabLeft (1) - Left tab
TabCenter (2) - Center tab
TabRight (3) - Right tab
TabNumeric (4) - Numeric Tab

TabRuler(ChangeCols, NumCols)

This function changes the number of total columns for the ruler.

NumCols is the total number of columns.

Return Value

If Command is ChangeMargins, 1 (TRUE) if successful.

0 (NoAction) if no margins are changed.

If Command is ChangeAllIndent, ChangeFirstIndent, ChangeSecIndent, or ChangeRightIndent, 1 (TRUE) if successful.

-6 (InsufficientMemory) if not enough memory.

If Command is ChangeTabs, 0 (NoAction) if the incorrect tab number is used.

1 (TRUE) if successful.

-6 (InsufficientMemory) if not enough memory.

If Command is ChangeCols, 1.

Example

```
FUNCTION Example()
' Delete all the tabs
TabRuler(ChangeTabs, 0, -3)
'Add tab 1 to be center tab at 2.5 inches
TabRuler(ChangeTabs, 0, -2, 2160, 2)
' Add tab 2 to be left tab at 3.5 inches
TabRuler(ChangeTabs, 1, -2, 3600, 1)
' Add tab 3 to be right tab at 5 inches
TabRuler(ChangeTabs, 2, -2, 5760, 3)
' Add tab 4 to be numeric tab at 6 inches
TabRuler(ChangeTabs, 3, -2, 7200, 4)
' Change the second tab to offset of 4 inches
TabRuler(ChangeTabs, 1, 4320)
' Change the top margin to start at 3 inches
' and bottom margin to start at 9 inches
TabRuler(ChangeMargins, 1, 0, 4320, 12960)
' Change the left margin to .5 inches
' and the right margin to start at 6 inches
TabRuler(ChangeMargins, 1, 1, 720, 8640)
' Change all indents .5 inches
TabRuler(ChangeAllIndent, 720)
' Change first indent to 1 inch
TabRuler(ChangeFirstIndent, 1440)
' Change second indent to .75 inches
TabRuler(ChangeSecIndent, 1080)
' Change right indent to 1 inch
TabRuler(ChangeRightIndent, 1440)
' Change the number of columns to 2
TabRuler(ChangeCols, 2)
END FUNCTION
```

See also:

ModifyStyle ModifyLayout

This function inserts a tab ruler. Choosing this function is equivalent to choosing Page/Ruler/Insert.

Syntax

TabRulerInsert()

Return Value

1 (TRUE) if the tab ruler was inserted.

If this function is called when Page/Ruler/Insert is grayed, the macro terminates and displays an error message.

Example

```
FUNCTION Example()
TabRulerInsert()
END FUNCTION
```

See also:

ModifyLayout FloatingHeader TabRulerRemove

This function removes a tab ruler. Choosing this function is equivalent to choosing Page/Ruler/Remove.

Syntax

TabRulerRemove()

Return Value

1 (TRUE) if the tab ruler was removed.

If this function is called when Page/Ruler/Remove is grayed, the macro terminates and displays an error message.

Example

```
FUNCTION Example()
TabRulerRemove()
END FUNCTION
```

See also:

ModifyLayout FloatingHeader TabRulerInsert

This function displays the Thesaurus dialog box and searches for the selected word in the Thesaurus. Choosing this function is equivalent to choosing Tools/Thesaurus. This function does not automatically use the thesaurus functions.

Syntax

Thesaurus()

Return Value

This function returns 0.

Example

```
FUNCTION Example()
Thesaurus()
END FUNCTION
```

See also:

SpellCheck

This function allows you to tile Multiple Document Interface (MDI) windows without overlapping them. Choosing this function is equivalent to choosing Window/Tile.

Syntax

TileWindow()

Return Value

This function does not return a value.

Example

```
FUNCTION Example()
Text = UCASE$(Left$(Query$("What action to take (Tile, Cascade, New, Select) on MDI Windows?"),
1))
SWITCH Text
CASE "T"
TileWindow()
CASE "C"
CascadeWindow()
CASE "N"
NewWindow()
CASE "S"
SelectWindow(Query$("Name of window to select (Name must match EXACTLY)?"))
default
Message("Only the proposed options are available.")
ENDSWITCH
END FUNCTION
```

See also:

CascadeWindow NewWindow NextWindow SelectWindow

This function sets the options for a generated table of contents for the current document. Choosing this function is equivalent to choosing File/Master Document/Options/TOC Options. It is also equivalent to choosing Tools/TOC, Index/TOC Options.

This function replaces the SetTOCOpts function that is available in Ami Pro 2.0.

Syntax

TOCOptions(Style, Separator, Flag[, Style, Separator, Flag...])

Style is the name of the paragraph style.

Separator is the character that separates the text from the page number. The separator that can be used is dependent on the Flag parameter and can be one of the following:

("") - None
(".") - Dot
("-") - Dash
("_") - Underline
(",") - Comma

Flag can be either page numbers, or right alignment and page numbers.

(1) - Use page numbers
(2) - Use right alignment (must combine with Use page numbers)

The rules for the Flag parameter:

If both page numbers and right align are used, the separator can be none, dot, dash or underline.
If only page numbers are used, the separator can be only none or comma.
If right align is used, it must be combined with the value for Use page numbers.
If page numbers are not used, the separator must be none.

To display the TOC Options dialog box: **TOCOptions**

Return Value

This function returns 1.

Example

```
FUNCTION Example()
TOCOptions("Title" "." 3 "Subhead" ", " 1 )
END FUNCTION
```

See also:

Generate SetTOCFile

This functions turns on Clean Screen mode if it is not already on. If Clean Screen mode is on, then it turns off. Choosing this function is equivalent to choosing View/Show/Hide Clean Screen.

Syntax

ToggleCleanScreen()

Return Value

This function returns 1.

Example

```
FUNCTION Example()
CleanScreenOptions(112)    ' Display the scroll bars and return icon
ToggleCleanScreen()
END FUNCTION
```

See also:

CleanScreenOptions

This function shows or hides the set of SmartIcons. If the set is displayed, it is removed. If the palette is not displayed, it is displayed. Choosing this function is equivalent to choosing View/Show/Hide SmartIcons.

Syntax

ToggleIconBar()

Return Value

1 (TRUE) if the icon bar toggled.

0 (NoAction) if no action was taken.

Example

```
FUNCTION Example()
ToggleIconBar()
END FUNCTION
```

See also:

ViewPreferences HideIconBar ShowIconBar ToggleStylesBox ToggleTabRuler IconBottom IconCustomize IconFloating IconLeft IconRight IconTop ShowIconBar

This function shows or hides the Styles Box. If the Styles Box is displayed, it is removed. If the Styles Box is not displayed, it is displayed. Choosing this function is equivalent to choosing View/Show/Hide Styles Box.

Syntax

ToggleStylesBox()

Return Value

This function returns 1.

Example

```
FUNCTION Example()
ToggleStylesBox()
END FUNCTION
```

See also:

ViewPreferences HideStylesBox ShowStylesBox ToggleIconBar ToggleTabRuler

This function shows or hides the tab ruler. If the tab ruler is displayed, it is removed. If the tab ruler is not displayed, it is displayed. Choosing this function is equivalent to choosing View/Show/Hide Ruler.

Syntax

ToggleTabRuler()

Return Value

This function returns 1.

Example

```
FUNCTION Example()
ToggleTabRuler()
END FUNCTION
```

See also:

ViewPreferences HideTabRuler ShowTabRuler ToggleIconBar ToggleStylesBox

This function scrolls the document to the beginning of the file without moving the insertion point. Choosing this function is equivalent to dragging the elevator on the vertical scroll bar to the extreme top using the mouse.

Syntax

TopOfFile()

Return Value

This function returns 0.

Example

```
FUNCTION Example()
Action = Query$("Move to (T)op of file, or (E)nd of file?")
Action = UCASE$(Left$(Action, 1))
SWITCH Action
CASE "T"
Message(TopOfFile())
CASE "E"
Message(EndOfFile())
default
Message("""T"" or ""E"" will do just fine, please.")
ENDSWITCH
END FUNCTION
```

See also:

CharLeft CharRight EndOfFile LeftEdge LineDown LineUp RightEdge ScreenDown ScreenLeft ScreenRight ScreenUp

This function removes any fractional part of a number.

Syntax

Truncate(Value)

Value is the number to be evaluated.

Return Value

The integer portion of the passed value.

Example

```
FUNCTION Example()
Number = 123.4455682
Message(Truncate(Number))
END FUNCTION
```

See also:

Round Mod IsNumeric

This function is used to type information into the current Ami Pro document. It is also used to type insertion point movement and function keys.

Syntax

TYPE(Text)

Text is the string which is typed into the document.

Text can be any of the keyboard characters. Any attributes or variables within the string are not typed into the document. To insert a left curly brace, type two curly braces ({{). To insert a left square brace, type two square braces ([[). To insert a double quote mark, type two double quote marks (""). Text can contain a variable name. If a variable name is used, enclose it in curly braces ({}) so that it can be recognized as a variable name. Text can also contain an insertion point movement or function key. To type a key, surround its name with square braces. An insertion point movement or function key can include the words CTRL, SHIFT or ALT to indicate a shifted state. The following key names can be used:

[Home] - Home Key
[End] - End Key
[PgUp] - Page Up Key
[PgDn] - Page Down Key
[Ins] - Insert Key
[Del] - Delete Key
[Backspace] - Backspace Key
[Enter] - Enter or Return Key
[Tab] - Tab Key
[ESC] - Escape Key
[Up] - Up Arrow Key
[Down] - Down Arrow Key
[Left] - Left Arrow Key
[Right] - Right Arrow Key
[F1] - [F12] - Function Keys F1 through F12

Return Value

This function does not return a value.

Example

```
FUNCTION Example()
TYPE("NumberCharacter[Enter]")
FOR I = 1 to 255
    Char = CHR$(I)
    TYPE("{I}={Char}[Enter]")
NEXT
END FUNCTION
```

See also:

FormatNum$ CurChar$ CurWord$ CurShade$

This function turns Typeover mode on or off. It acts as a toggle, turning off the Typeover mode if it is currently on or turning on the Typeover mode if it is currently off. Choosing this function is equivalent to pressing the Insert key.

Syntax

TypeOver()

Return Value

1 (TRUE) if the typeover status was changed.

0 (NoAction) if no action was taken.

Example

```
FUNCTION Example()
TypeOver()
END FUNCTION
```

See also:

TYPE

This function changes lowercase letters in the source string to upper case and returns the resulting string. It does not change punctuation or numbers.

Syntax

UCASE$(Text)

Text is the string which changes to upper case.

Return Value

The string with all uppercase letters.

Example

```
FUNCTION Example()
Name = UCASE$(Query$("What is your name?"))
Message(Name)
END FUNCTION
```

See also:

ASC CHR$ strcat$ LCASE$ strfield$ MID$ LEN FormatNum$

This function sets the underline attribute for selected text or for all following text if no text is selected. It acts as a toggle, turning off the attribute if it is currently on or turning on the attribute if it is currently off. Choosing this function is equivalent to choosing Text/Underline.

Syntax

Underline()

Return Value

0 if the underline attribute is toggled on and there are no attributes assigned to the text.

4 if the underline attribute is toggled on and the bold attribute is already assigned.

8 if the underline attribute is toggled on and the italic attribute is already assigned.

16 if the underline attribute is toggled off.

32 if the underline attribute is toggled on and the word underline attribute is already assigned.

The return values may be added together to identify the attributes that were previously assigned.

-2 (GeneralFailure) if the attribute was not changed.

Example

```
FUNCTION Example()
text = Query$("Enter some text:")
New("_default.sty", 0, 0)     'open a new file
TYPE("{text}")
TYPE("[enter]")
TYPE("[ctrlhome][ctrlshiftend]")
Copy()  'copy text to clipboard
FOR I = 1 to 10
    Paste()     'paste text 10 times
NEXT
Message("The text will be shaded, bolded, italicized, underlined, and centered.")
TYPE("[ctrlhome][ctrlshiftend]")
Bold()
Italic()
Underline()
Center()
TYPE("[ctrlhome]")
END FUNCTION
```

See also:

Bold Italic NormalText WordUnderline

This function undoes the previous editing function. Choosing this function is equivalent to choosing Edit/Undo.

Syntax

Undo()

Return Value

1 (TRUE) if the function was successfully undone.

0 (NoAction) if no action was taken.

Example

```
FUNCTION Example()
Undo()
END FUNCTION
```

See also:

UserSetup

This function displays the online Help for Ami Pro. Choosing this function is equivalent to choosing Help/Upgrade. This function does not select a Help topic automatically. Because Help displays in a separate window, further macro functions which cause a repaint of the Ami Pro window force Ami Pro to replace the Help window.

If this function is used, it should be the last function used in the macro.

Syntax

UpgradeHelp()

Return Value

1 (TRUE) if the Help window was displayed.

-2 (GeneralFailure) if the Help window could not be displayed for some other reason.

-6 (NoMemory) if the function failed because of insufficient memory.

Example

```
FUNCTION Example()
UpgradeHelp()
END FUNCTION
```

See also:

About EnhancementProducts HowDoIHelp KeyboardHelp MacroHelp UsingHelp Help

This function sets upper case for selected text or for all following text if no text is selected. It acts as a toggle, turning off upper case if it is currently on or turning on upper case if it is currently off. It is the equivalent to choosing Text/Caps/Upper Case.

Syntax

UpperCase()

Return Value

1 (TRUE) if the attribute was changed.

0 (UserCancel) if the user canceled the function.

-2 (GeneralFailure) if the attribute could not be changed.

-6 (NoMemory) if the function failed because of insufficient memory.

Example

```
FUNCTION Example()
UpperCase()
END FUNCTION
```

See also:

InitialCaps SmallCaps LowerCase

This function selects a new paragraph style sheet for the document on the screen. Choosing this function is equivalent to choosing Style/Use Another Style Sheet.

Modified in 3.0 There is a new parameter, Options, in Ami Pro release 3.0.

Syntax

UseAnotherStyle(Name, Options)

Name is the name of the new paragraph style sheet to use.

Options is a flag indicating the format for displaying the style sheet name in the Use Another Style Sheet dialog box.

(1) - show the style sheets by description

(2) - show the style sheets by file name

To display a dialog box to allow the user to select the paragraph style sheet to use: **UseAnotherStyle**

Return Value

1 (TRUE) if the paragraph style sheet was changed.

0 (UserCancel) if the user canceled the function.

-2 (GeneralFailure) if the paragraph style sheet could not be changed.

Example

```
FUNCTION Example()
UseAnotherStyle("_MACRO.STY")
END FUNCTION
```

See also:

ModifyStyle SaveAsNewStyle SelectStyle SetStyle

This function displays a Windows modeless dialog box titled "Macro Pause" with the specified prompt string and Resume and Cancel push buttons. The user can click outside this dialog box to do any regular Ami Pro function. When the user is done, he can click the Resume button to resume the macro or the Cancel button to cancel the macro.

Press **ALT+F6** to toggle between Ami Pro and the dialog box.

Syntax

UserControl(Prompt)

prompt is a string used as a prompt to the user. It can be a maximum of 80 characters.

Return Value

This function does not return a value. If the user selects Cancel, control passes to the routine defined by the ONCANCEL statement.

Example

```
FUNCTION Example()
UserControl("Click Resume to bring up Control Panel...")
ControlPanel
UserControl("Click Resume to shut down Control Panel...")
AppClose("Control Panel")
END FUNCTION
```

See also:

Decide DialogBox IgnoreKeyboard Message Messages MultiDecide Query$ SingleStep KeyInterrupt

This function sets defaults for using Ami Pro. Choosing this function is equivalent to choosing Tools/User Setup.

Syntax

UserSetup(UndoLevel, SaveTime, Options, Flag, UserName, UserInitials, Color, LoadMacro, ExitMacro, MacroOptions, RecentFiles)

UndoLevel is the desired undo level (0-4).

SaveTime is the time between saves, in minutes. If the autosave function is on.

Options is the flag containing other default options. It is one of the following values:

NoUndo (0) - No level set is undone
Undo1 (2) - Single undo level
Undo2 (4) - Two levels undone
Undo3 (8) - Three levels undone
Undo4 (16) - Four levels undone
(32) - Disable one-line Help
(64) - Disable the Drag & Drop feature
BackupOn (128) - Automatic Backup turned on
TimedSaveOn (256) - Timed Save turned on
(2048) - Disable warning messages

Flag is currently unused, and is set to 0.

UserName is the name of the user for the document locking function.

UserInitials is the initials of the user for note functions.

Color is the color for notes. It is one of the following values:

White (16777215)
Cyan (16776960)
White (16777215)
Cyan (16776960)
Yellow (65535)
Magenta (16711935)
Green (65280)
Red (255)
Blue (16711680)
Black (0)

LoadMacro is the name of the macro to run automatically when Ami Pro is loaded.

ExitMacro is the name of the macro to run automatically when Ami Pro is exited.

MacroOptions is the options for running the load and exit macros. It is one of the following values:

NoMacroRun (0) - Do not run the load macro or exit macro.
MacroLoad (2) - Set the flag to run the macro given for LoadMacro when Ami Pro is loaded.
MacroExit (4) - Set the flag to run the macro given for ExitMacro when Ami Pro is exited.

RecentFiles is the number of recent files to list on the File menu.

To display a dialog box to allow the user to select his defaults: **UserSetup**

Return Value

1 (TRUE) if the setup defaults were set.

0 (UserCancel) if the user canceled the function.

Example

```
FUNCTION Example()
UserSetup(4, 0, 0, 0, "Your Name", "YIN", 65535, "", "", 0, 0)
END FUNCTION
```

See also:

SetDefOptions SetDefPaths SetDocPath SetStylePath SetBackPath SetMacroPath LoadOptions ViewPreferences

This function determines whether to use the last directory you selected when a file was opened or saved. Choosing this function is equivalent to choosing Tools/User Setup/Paths and selecting or deselecting Use working directory.

Syntax

UseWorkingDir(Flag)

Flag is one of the following:

(1) - remembers the last directory used when opening or saving a file.

(0) - doesn't remember the last directory used when opening or saving a file.

The directory used to open or save the next file is based on the default document path defined in the Tools/User Setup/Paths/Document text box when the flag is set to zero. If the SetDocPath function is used, the working directory path resets to the new path but the UseWorkingDir flag is still valid.

To display the Default Paths dialog box: **SetDefPaths**

Return Value

This function returns 1.

Example

```
FUNCTION Example()
UseWorkingDirectory(1)
END FUNCTION
```

See also:

SetDefPaths SetDocPath UserSetupUSERSETUP

This function displays the online Help for Ami Pro. Choosing this function is equivalent to choosing Help/Using Help. This function does not select a Help topic automatically. Because Help displays in a separate window, further macro functions that cause a repaint of the Ami Pro window force Ami Pro to replace the Help window.

If this function is used, it should be the last function used in the macro.

Syntax

UsingHelp()

Return Value

1 (TRUE) if the Help window was displayed.

-2 (GeneralFailure) if the Help window could not be displayed for some other reason.

-6 (NoMemory) if the function failed because of insufficient memory.

Example

```
FUNCTION Example()
UsingHelp()
END FUNCTION
```

See also:

About UpgradeHelp EnhancementProducts HowDoIHelp KeyboardHelp MacroHelp Help

This function sets display preferences for using Ami Pro. Choosing this function is equivalent to choosing View/View Preferences.

Syntax

ViewPreferences(Options, ViewLevel)

Options are the display preferences to use, and can be one or more of the following:

ColumnGuides (1) - Displays column guides.

ColorMargin (2) - Displays margins in color.

ShowPictures (4) - Shows pictures in layout mode.

ShowTabs (8) - Shows tabs and returns.

ShowInitials (32) - Shows the initials with the note marker.

ShowMarks (64) - Shows marks.

ShowGrid (128) - Shows gridlines in table mode.

ShowRowCol (256) - Shows Row/Column headings in table mode.

ShowVertRuler (512) - Shows vertical ruler in layout mode.

ShowHorzScroll (1024) - Displays the horizontal scroll bar.

ShowNotes (2048) - Shows note marks in layout mode.

NoDisplayAsPrint (4096) - Does not use display as printed option. Setting this option turns off the display as printed option, unlike the other flags, which turn on the option.

ShowOutlineButtons (16384) - Shows outline buttons in outline mode.

ShowSysFont (32768) - Uses the system font in Draft and Outline mode.

To set multiple options, add the option values together before passing them to the function.

ViewLevel is the Custom View level to use for Custom View. The ViewLevel parameter should be set to the desired view level for the Custom View level, between 10% and 400%.

To show the Display Preferences dialog box and allow the user to select his display preferences: **ViewPreferences**

Return Value

1 (TRUE) if the display preferences were successfully set.

0 (UserCancel) if the user canceled the function.

-2 (GeneralFailure) if the preferences could not be set.

Example

```
FUNCTION Example()
ViewPreferences((1 + 2 + 4 + 8 + 64 + 128 + 256 + 512 + 2048 + 16384 + 4096), 91)
END FUNCTION
```

See also:

SetDefOptions

This function sets word only underlining for selected text or for all following text if no text is selected. It acts as a toggle, turning off word underline if it is currently on and turning on word underline if it is currently off. Choosing this function is equivalent to choosing Text/Word Underline.

Syntax

WordUnderline()

Return Value

(0) if the word underline is toggled on and there are no attributes assigned to the text.

(4) if the word underline attribute is toggled on and the bold attribute is already assigned.

(8) if the word underline attribute is toggled on and the italic attribute is already assigned.

(16) if the word underline attribute is toggled on and the underline attribute is already assigned.

(32) if the word underline attribute is toggled off.

(-2) (GeneralFailure) if the attribute was not changed.

The return values may be added together to identify the attributes that were previously assigned.

Example

```
FUNCTION Example()
text = Query$("Enter some text:")
New("_default.sty", 0, 0)     'open a new file
TYPE("{text}")
TYPE("[enter]")
TYPE("[ctrlhome][ctrlshiftend]")
Copy()  'copy text to clipboard
FOR I = 1 to 10
    Paste()     'paste text 10 times
NEXT
Message("The text will word underline only.")
TYPE("[ctrlhome][ctrlshiftend]")
WordUnderline()
TYPE("[ctrlhome]")
END FUNCTION
```

See also:

Bold Italic NormalText Underline

This function changes the current view level to Working View. Choosing this function is equivalent to choosing View/Working.

Syntax

WorkingView()

Return Value

This function returns 1.

Example

```
FUNCTION Example()
WorkingView()
END FUNCTION
```

See also:

EnlargedView FacingView
FullPageView GetViewLevel LayoutMode StandardView

This function writes an entry into a Windows .INI file.

Syntax

WriteProfileString(App, Item, Value[, FileName)

App is the name of the section in the .INI file. If the null string ("") is passed for this parameter, the [AmiPro] section is used.

Item is the name of the entry to be added/changed.

Value is the data to be added/changed in the .INI file.

FileName is the filename to modify. If this parameter is not used or is passed the null string (""), the Window's WIN.INI file is used. If the full path is not used, the file is assumed to be in the Windows subdirectory.

If the file does not exist, a new file is created and this entry is placed into it.

Return Value

This function does not return a value.

Example

```
FUNCTION Example()
Name = GetProfileString$("AmiPro", "UserName", "AMIPRO.INI")
FillEdit(8000, Name)
DIM Filters(10)
FOR I = 1 to 10
Filters(I) = strfield$(GetProfileString$("AmiPro", "application{I}", "AMIPRO.INI"), 1, ",")
FillList(Filters(I))
NEXT
FOR I = 1 to 10
FillEdit(9500, Filters(I))
NEXT
FillEdit(50, TRUE)
FillEdit(55, TRUE)
Box = DialogBox(".", "ExampleBox")
IF Box = -1
Message("Could not open dialog box; Exiting macro.")
EXIT FUNCTION
ELSEIF Box = 0
EXIT FUNCTION
ENDIF
Name2 = GetDialogField$(8000)
IF Name != Name2
IF Decide("Do you want to save your changes?")
WriteProfileString("AmiPro", "UserName", Name2, "AMIPRO.INI")
ENDIF
ENDIF
END FUNCTION

DIALOG ExampleBox
-2134376448 14 104 36 198 90 "" "" "Sample Dialog Box"
FONT 6 "Helv"
4 6 42 10 1000 1342308352 "static" "&User Name:" 0
52 4 92 12 8000 1350631552 "edit" "" 0
4 24 34 8 1001 1342308352 "static" "&Filters:" 0
4 34 66 52 9000 1352728579 "listbox" "" 0
74 24 52 8 1002 1342308352 "static" "&More Filters:" 0
74 34 70 40 9500 1344339971 "combobox" "" 0
```

```
74 46 60 40 24 1342308359 "button" "Group Box #1" 0
78 58 50 10 50 1342242825 "button" "Radio #1" 0
78 70 50 10 51 1342177289 "button" "Radio #2" 0
138 46 56 40 25 1342308359 "button" "Group Box #2" 0
142 56 48 12 55 1342242819 "button" "Check #1" 0
142 68 48 12 56 1342177283 "button" "Check #2" 0
154 4 40 14 1 1342373889 "button" "OK" 0
154 20 40 14 2 1342373888 "button" "Cancel" 0
END DIALOG
```

See also:

GetProfileString$

The macro documentation has the following major sections:

The following topics are discussed in the Macro Language Reference. Click on a topic to read more about it.

Using the ONERROR Statement
Using the ONCANCEL Statement
Using Results of Evaluations to Control a Macro
Using the FOR Statement
Using the IF/THEN Statements
Using the SWITCH and CASE Statements
Using the WHILE Statement
Using the BREAK Statement
See also:
Ami Pro Macro Language Contents

This is a list of the statements used in the Ami Pro Macro Language. Click on the statement name to see how to use the statement.

BREAK
CALL
CALLI
DECLARE
DEFSTR
DIM
END FUNCTION
EXIT FUNCTION
FOR
FUNCTION
GOTO
IF
LABEL
LET
ONCANCEL
ONERROR
SWITCH
WHILE

This section lists each built-in macro function by type. You can easily find the function you need by looking in the appropriate category.

Using Macro Functions Grouped by Category

The categories available are:

Variables

Strings

Ami Pro Menus

Ami Pro Word Processing

Windows Applications

Frames

Style

Page Layout

Dialog Box

Arrays

ASCII Files

NewWave

Macro Only Commands

DOS

See also:

- Ami Pro Macro Language Contents

About
ActivateApp
AddBar
AddCascadeMenu
AddCascadeMenuItem
AddFrame
AddFrameDlg
AddMenu
AddMenuItem
AddMenuItemDDE
AllocGlobalVar
AmiProIndirect
AnswerMsgBox
AppClose
AppGetAppCount
AppGetAppNames
AppGetWindowPos
AppHide
AppIsRunning
ApplyFormat
AppMaximize
AppMinimize
AppMove
AppRestore
AppSendMessage
AppSize
ArrayDelete
ArrayInsert
ArrayInsertByKey
ArraySearch
ArraySize
ArraySort
ASC
ASCIIOptions
Assign
AssignMacroToFile
AtEOF
Beep
BinToBrackets
Bold
BracketsToBin

BringFrameToFront
CascadeWindow
Center
ChangeCascadeAction
ChangeIcons
ChangeLanguage
ChangeMenuAction
ChangeShortcutKey
CharLeft
CharRight
ChartingMode
CheckMenuItem
CHR$
CleanScreenOptions
ClipboardRead
ClipboardWrite
ConnectCells
ControlPanel
Copy
CreateANew (NewWave)
CreateDataFile
CreateDescriptionFile
CreateStyle
CurChar$
CurShade$
CursorPosition$
CurWord$
CustomView
Cut

See also:

Ami Pro Macro Language Contents
Macro Functions D - E
Macro Functions F - G
Macro Functions H - M
Macro Functions N - R
Macro Functions S - Z

DateDiff
DDEAdvise
DDEExecute
DDEInitiate
DDELinks
DDEPoke
DDEReceive$
DDETerminate
DDEUnAdvise
Decide
DECLARE
DefineStyle
DeleteColumnRow
DeleteEntireTable
DeleteMenu
DeleteMenuItem
DialogBox
DlgKeyInterrupt
DLLCall
DLLFreeLib
DLLLoadLib
DLLLocate
DocInfo
DocInfoFields
DocumentCompare
DOSchdir
DOSCopyFile
DOSDelFile
DOSGetEnv$
DOSGetFileAttr
DOSmkdir
DOSRename
DOSrmdir
DOSSetFileAttr
DraftMode
DrawingMode
EditFormula
ElevatorLeftRight
ElevatorUpDown
EndOfFile
EnhancementProducts

EnlargedView

Equations

EvalField

Exec

See also:

Ami Pro Macro Language Contents
Macro Functions A - C
Macro Functions F - G
Macro Functions H - M
Macro Functions N - R
Macro Functions S - Z

FacingView
FastFormat
fclose
fgets$
FieldAdd
FieldAuto
FieldCommand
FieldEvaluate
FieldLock
FieldNext/FieldPrev
FieldRemove
FieldToggleDisplay
FieldUpdate
FieldUpdateAll
FileChanged
FileClose
FileManagement
FileOpen
FilePrint
FillEdit
FillList
FindFirst$
FindNext$
FindReplace
FloatingHeader
FontChange
FontFaceChange
FontPointSizeChange
FontRevert
Footnotes
fopen
FormatDate$
FormatNum$
FormatSeq$
FormatTime$
fputs
FrameLayout
FrameModBorders
FrameModColumns
FrameModFinish
FrameModInit

FrameModLines
FrameModType
fread
FreeGlobalVar
fseek
ftell
FullPageView
fwrite
Generate
GetAmiDirectory$
GetBackPath$
GetBookMarkCount
GetBookMarkNames
GetBookMarkPage
GetCurFontInfo
GetCurFrameBorders
GetCurFrameLines
GetCurFrameType
GetCurrentDir$
GetDialogField$
GetDlgItem
GetDlgItemText
GetDocInfo$
GetDocInfoKeywords$
GetDocPath$
GetDocVar
GetFmtPageStr$
GetGlobalArray$
GetGlobalVar$
GetGlobalVarCount
GetGlobalVarNames
GetIconPalette
GetLayoutLeftLines
GetLayoutPageSize
GetLayoutParameters
GetLayoutParmCnt
GetLayoutRightLines
GetLayoutType
GetMacPath$
GetMarkText$
GetMasterFiles
GetMasterFilesCount
GetMode

GetOpenFileCount
GetOpenFileName$
GetOpenFileNames
GetPageNo
GetPowerFieldCount
GetPowerFieldPage
GetPowerFields
GetProfileString$
GetRunningMacroFile$
GetRunningMacroName$
GetSpecialEffects$
GetStyleCount
GetStyleName$
GetStyleNames
GetStylePath$
GetTextBeforeCursor$
GetTime
GetViewLevel
GetViewPrefLevel
GetViewPrefOpts
GetWindowsDirectory$
Glossary
GlossaryAdd
GlossSet
GoToAgain
GoToCmd
GoToPowerField
GoToShade
GraphicsScaling
GrayMenuItem
GroupFrames

See also:

- Ami Pro Macro Language Contents
- Macro Functions A - C
- Macro Functions D - E
- Macro Functions H - M
- Macro Functions N - R
- Macro Functions S - Z

HeaderFooter
Heading
Help
HideIconBar
HideStylesBox
HideTabRuler
HourGlass
HowDoIHelp
IconBottom
IconCustomize
IconFloating
IconLeft
IconRight
IconTop
IgnoreKeyboard
ImageProcessing
ImportExport
ImportPicture
ImportText (NewWave)
Indent
IndentAll
IndentFirst
IndentRest
InitialCaps
InsertBullet
InsertCascadeMenu
InsertCascadeMenuItem
InsertColumnRow
InsertDate
InsertDocInfo
InsertDocInfoField
InsertLayout
InsertMenu
InsertMenuItem
InsertMerge
InsertNewObject
InsertNote
InsertVariable
Instr
IsFrameSelected
IsNewWave (NewWave)

IsNumeric
Italic
Justify
KeyboardHelp
KeyInterrupt
LayoutMode
LCASE$
LeaderDots
LeaderHyphs
LeaderLines
LeaderNone
Left$
LeftAlign
LeftEdge
LEN
LineDown
LineNumber
LineUp
ListObjects (NewWave)
LoadOptions
LowerCase
MacroEdit
MacroHelp
MacroOptions
MacroPlay
ManualFrame
MarkBookMark
MarkIndexWord
MarkTOCEntry
MasterDoc
MasterDocOpts
Maximize
Merge
MergeAction
MergeMacro
MergeToFile
Message
Messages
MID$
Minimize
Mod
ModifyAlignment
ModifyBreaks

ModifyEffects
ModifyFont
ModifyLayout
ModifyLines
ModifyReflow
ModifySelect
ModifySpacing
ModifyStyle
ModifyTable
ModLayoutFinish
ModLayoutInit
ModLayoutLeftFooter
ModLayoutLeftHeader
ModLayoutLeftLines
ModLayoutLeftPage
ModLayoutPageSize
ModLayoutRightFooter
ModLayoutRightHeader
ModLayoutRightLines
ModLayoutRightPage
MouseInterrupt
MoveLeftOrPromote
MoveParagraphDown
MoveParagraphUp
MoveRightOrDemote
MultiDecide

See also:

Ami Pro Macro Language Contents
Macro Functions A - C
Macro Functions D - E
Macro Functions F - G
Macro Functions N - R
Macro Functions S - Z

New
NewWindow
NextWindow
NoHyphenation
NormalText
Notes
Now
NWGetContainerCount (NewWave)
NWGetContainerNames (NewWave)
NWGetCurrentContainer (NewWave)
NWGetCurrentObject$ (NewWave)
NWGetObjectCount (NewWave)
NWGetObjectNames (NewWave)
NWGetParent (NewWave)
NWReferenceToFile$ (NewWave)
ObjectAttributes (NewWave)
OnKey
OnMDIActivate
OpenDataFile
OpenMergeFile
OpenObject (NewWave)
OpenPreviousFile1
OpenPreviousFile2
OpenPreviousFile3
OpenPreviousFile4
OpenPreviousFile5
OutlineLevels
OutlineMode
OutlineStyle
PageBreak
PageDown
PageNumber
PageUp
Paste
Pause
PhysicalToLogical
PrintEnvelope
PrintOptions
PrintSetup
ProtectCells
ProtectedText

Query$
QuickAddCol
QuickAddRow
ReadMail
RecClose
RecFieldCount
RecFieldName$
RecGetField
RecNextRec
RecOpen
RemoveLayout
RenameDocInfoField
RenameMenuItem
Replace
Restore
Revert
RevertLayout
ReviewRevisions
RevisionInsertion
RevisionMarking
RevisionMarkOpts
Right$
RightAlign
RightEdge
Round
RunLater

See also:

Ami Pro Macro Language Contents
Macro Functions A - C
Macro Functions D - E
Macro Functions F - G
Macro Functions H - M
Macro Functions S - Z

Save
SaveAs
SaveAsMaster (NewWave)
SaveAsNewStyle
SaveAsObject (NewWave)
ScreenDown
ScreenLeft
ScreenRight
ScreenUp
SelectColumn
SelectEntireTable
SelectFrameByName
SelectRow
SelectStyle
SelectWindow
SendFrameToBack
SendKeys
SendMail
SetBackPath
SetDataFile
SetDefOptions
SetDefPaths
SetDlgCallBack
SetDlgItemText
SetDocPath
SetDocVar
SetFormula
SetFrameDefaults
SetGlobalArray
SetGlobalVar
SetIconPath
SetIconSize
SetIndexFile
SetMacroPath
SetMasterFiles
SetStyle
SetStylePath
SetTOCFile
Share (NewWave)
ShowBar
ShowIconBar

ShowLinks (NewWave)
ShowStylesBox
ShowTabRuler
SingleStep
SizeColumnRow
SmallCaps
Sort
Spacing
SpecialEffects
SpellCheck
StandardView
StatusBarMsg
strcat$
strchr
strfield$
StyleManageAction
StyleManageFinish
StyleManageInit
StyleManagement
TableGetRange
TableLayout
TableLines
Tables
TabRuler
TabRulerInsert
TabRulerRemove
Thesaurus
TileWindow
TOCOptions
ToggleCleanScreen
ToggleIconBar
ToggleStylesBox
ToggleTabRuler
TopOfFile
Truncate
TYPE
TypeOver
UCASE$
Underline
Undo
UpgradeHelp
UpperCase
UseAnotherStyle

UserControl

UserSetup

UseWorkingDir

UsingHelp

ViewPreferences

WordUnderline

WriteProfileString

See also:

- Ami Pro Macro Language Contents
- Macro Functions A - C
- Macro Functions D - E
- Macro Functions F - G
- Macro Functions H - M
- Macro Functions N - R

Many Windows applications have macro language commands that support DDE functionality. Ami Pro can support these DDE commands through the use of bookmarks. When you name an existing bookmark in a DDEPoke message, Ami Pro replaces the contents of the bookmark with the poked data.

Generally there are three steps. First, you initiate the DDE conversation using the other application's macro language. Usually this is done with some type of Initiate or DDEInitiate command. Second, you use the other application's macro language to poke data from that application into an Ami Pro bookmark. Usually this is done with some type of Poke or DDEPoke command. Finally, you terminate the DDE conversation using the other application's macro language. Usually this is done with a Terminate or DDETerminate command.

For example, you could write the following macro in 123W to poke data from a spreadsheet into an Ami Pro document:

```
{dde-open AMIPRO;"(Untitled)"} ' Open a DDE channel to an untitled Ami Pro document
{dde-execute [MarkBookMark("DDE_BookMark1",4003)]} ' Create a bookmark in Ami Pro to poke data in to.
{dde-poke Data,"DDE_BookMark1"}  ' Data is the named range of information to poke into the Ami Pro bookmark
{dde-close} ' Terminate the DDE channel to Ami Pro
{Quit}
```

This online Ami Pro Macro documentation documents the Ami Pro Macro Language. Use this documentation to create your own macros and to customize Record and Play macros.

You do not need this documentation to use the macro Record and Play functions. Record and Play functions use the same macro and Ami Pro functions described here, but automatically access these functions.

The Ami Pro Macro Language is similar to BASIC. If you are not a computer programmer and have not written many macros, you may have difficulty with some of the information presented in this documentation. You should start with simple macros, skipping sections that you do not understand. If you are an experienced computer programmer and have written macros for other programs, use this documentation to become proficient quickly in the Ami Pro macro language.

Along with the macro functions you automatically access when using power fields, you can use many other macro functions in power fields. These power fields/macro functions use the same parameters listed in this documentation.

See also:

Ami Pro Macro Language Contents
Using the Macro Language Documentation
Understanding Macro Documentation Conventions
Understanding Macros
Understanding Power Fields

This documentation contains several sections:

Introducing the Ami Pro Macro Language
This section describes how to use the macro language documentation. It also describes the macro documentation conventions.

Introducing New and Upgraded Macro Functions for Ami Pro release 3.0
This section lists the new Ami Pro Release 3.0 macro functions. It also lists the functions upgraded from Ami Pro release 2.0.

Creating, Saving, and Formatting an Ami Pro Macro
This section describes how to create, save, and edit a macro. It also describes how to format a macro for ease of reading.

Using Macro Programming Statements
This section describes the difference between macro programming statements and functions. It also describes the FUNCTION and END FUNCTION statements.

Using Variables
This section describes variables, strings, and the DEFSTR and DIM statements. It also describes the global variables and the lifetime and visibility of a variable.

Using Numbers, Strings, and Operators
This section describes the precedence of operators and how to use the various operator types. It also describes the LET statement.

Using Constants
This section describes defined and predefined constants. It also describes the DEFINE statement.

Calling Built-In Functions and Other Macros
This section describes calling built-in functions and Ami Pro functions. It also describes how to determine the location of a macro and using the return value from called macros. This section describes the DECLARE, CALL, and CALLI statements.

Transferring Program Control
This section describes how to transfer control of a macro to another function or statement. It also describes the GOTO, LABEL, ONERROR, and ONCANCEL statements.

Using Results of Evaluations to Control a Macro
This section describes how to use the results of an evaluation to control a macro. It also describes the FOR, IF, THEN, SWITCH, CASE, WHILE, and BREAK statements.

Using the Lotus Dialog Editor
This section describes the Lotus Dialog Editor that is available with Ami Pro Release 2.0 and higher.

Using Macro Functions
This section lists each macro function, grouped by category. Use this section to locate the functions you need when you know a function's purpose but not its name. It also lists each Ami Pro function, grouped by menu. These functions execute Ami Pro commands within a macro. This section lists each macro and Ami Pro function in alphabetical order.

See also:

Ami Pro Functions By Menu Name
Dialog Editor
Using Macro Functions Grouped by Category
Macro Functions A - C
Macro Functions D - E
Macro Functions F - G
Macro Functions H - M
Macro Functions N - R

The macro documentation includes functions available in Ami Pro releases 2.0 and 3.0. When referring to Ami Pro release 2.0 functions, 2.0 follows the name of the function, for example, **WorkingView 2.0**. When referring to Ami Pro release 3.0 functions, 3.0 follows the name of the function, for example, **CustomView 3.0**.

Several functions contain additional parameters only available in Ami Pro release 3.0. Those functions that contain additional parameters or flags include the phrase **Modified in 3.0** and a paragraph describing the differences.

This documentation presents macro statements and functions syntax in a consistent manner. This should help you understand how to use each statement or function. The statements and functions use the following format:

Function(parm1[parm2[parm3...]])

Parm1 is a required parameter.

Parm2 is an optional parameter.

Parm3 is an optional parameter.

In this sample function, the function name is first, followed by parameters, if any, for the function. When you use the function in a macro, you replace the parameter names with values you provide.

Parameters that are surrounded by square braces ([ ]) are optional unless otherwise stated. If you use an optional parameter or group of parameters, you must include all the information shown in the square braces (do not type the square braces). In this example, Parm2 and Parm3 are both optional.

Parm3 is optional when using Parm2. Parm2 encloses Parm3 within its own set of square braces. You cannot use Parm3 unless you use Parm2. You do not have to use Parm3 if you use Parm2.

The three dots following Parm3 show that you can repeat it if needed. If you can repeat a parameter or portion of a function, the documentation follows the parameter with three dots. If you repeat a parameter, you must repeat all of the function that is within the square braces.

Most functions and statements use examples to illustrate how to use the function in a macro. This documentation shows example macros in the following style:

```
1  FUNCTION test()
2  TYPE ("This is a test.[ENTER]")
3  END FUNCTION
```

Line numbering appears in the example macros that are in this section to make their explanation easier. Do not type the line numbers when you type the macros. Most of the example macros illustrate more than one function or programming concept. Many of the examples perform functions that you can use. Look at the examples for some macro ideas

See also:

Ami Pro Macro Language Contents
To use the FUNCTION statement
TYPE

There are many new functions for use with Ami Pro Release 3.0. There are also several upgraded functions. These upgraded functions contain either more parameters or more options for a parameter. A small number of Ami Pro Release 2.0 functions are not available. This is usually due to functions becoming obsolete. In some cases, there are new Ami Pro Release 3.0 functions to replace Ami Pro Release 2.0 functions.

There are many new functions for use with Ami Pro Release 3.0. The following is a list of these functions:

ApplyFormat
AssignMacroToFile
CleanScreenOptions
CreateDataFile
CreateDescriptionFile
CustomView
DeleteEntireTable
DlgKeyInterrupt
FastFormat
GetDocVar
GetIconPalette
GetPowerFieldCount
GetPowerFieldPage
GetPowerFields
GetStyleCount
GetStyleNames
GetViewPrefLevel
GetViewPrefOpts
GotoPowerField
HeaderFooter
IndentAll
IndentFirst
IndentRest
InsertBullet
IsNewWave
KeyInterrupt
ManualFrame
MarkTOCEntry
MouseInterrupt
MoveLeftOrPromote
MoveParagraphDown
MoveParagraphUp
MoveRightOrDemote
OnMDIActivate
OpenDataFile
OpenMergeFile
OpenPreviousFile1
OpenPreviousFile2
OpenPreviousFile3
OpenPreviousFile4

OpenPreviousFile5
PrintEnvelope
SelectColumn
SelectEntireTable
SelectRow
SetDocVar
SetIconPath
SetIconSize
TabRuler
TOCOptions
ToggleCleanScreen
UseWorkingDir

There are several upgraded functions in Ami Pro Release 3.0. These upgraded functions contain either more parameters or more options for a parameter.

Decide

InsertLayout

Message

MultiDecide

PrintSetup

QuickAddCol

QuickAddRow

SaveAsNewStyle

SetFrameDefaults

SetMasterFiles

strfield$

TableLayout

UseAnotherStyle

A small number of Ami Pro Release 2.0 functions are not available. This is usually due to functions becoming obsolete. In some cases, there are new Ami Pro Release 3.0 functions to replace Ami Pro Release 2.0 functions.

BasicsHelp

InsertMergeField (replaced with FieldAdd)

SetTOCOpts

WorkingView (replaced with CustomView)

This section shows you how to create and save Ami Pro macros. While writing macros, you can format them so they contain a consistent look and feel. This section also contains information about editing macros and understanding error messages.

Type your macro as an Ami Pro document. To save the file as an Ami Pro macro, choose File/Save or File/Save As and then select Ami Pro Macro as the file type. This saves the file with .SMM as the extension. If you save the file with the .SAM extension, Ami Pro cannot recognize and compile your macro.

When you save a file as a macro, the path changes to the default macro path that you entered by choosing Tools/User Setup/Paths. Although you can save macros to any directory, saving them to the default directory makes them easier to locate, edit, or playback.

When you save a macro file, Ami Pro compiles it into a special format that allows the macro to run faster. During compilation, Ami Pro checks the macro for errors. If Ami Pro finds an error, the compilation stops and an error message displays. The insertion point is at the location of the error so that you can correct the error. In some cases, the insertion point is on the line following the error. For example, if you left a right parenthesis off at the end of a line, the insertion point is on the next line.

You can create and edit your macro file using the File/New, File/Open, and File/Save As functions. You can choose Tools/Macros/Edit from the menu to edit an existing macro file.

You can edit a macro created with the Tools/Macros/Record menu function. When you edit a record and play macro, you may see an Error 112 message. This indicates that one or more of the functions in the record and play macro did not translate into editable form. A macro created in a previous release of Ami Pro can contain functions that are no longer available or are available by using a different function name. If you want to correct the error, use the new function name with the correct number of parameters.

You can easily identify untranslated functions. They appear as commented numbers in the macro file. If you save the macro file, you will lose the untranslated functions. If you want to keep the original functionality of the macro, do not save the edited macro.

If you want to reference your macro from other macros, do not place an exclamation point (!) in the file name when you save your macro. Ami Pro uses the exclamation point in the syntax of some macro functions.

Ami Pro finds some types of macro errors when the macro runs. These are logic errors that are not detectable when the macro compiles. For example, your macro can try to create a frame in draft mode. Ami Pro does not allow this and the macro results in an error.

For more information on error messages and diagnosing errors, see the Macro Errors and Debugging Macros section by choosing Help/Macro Doc and double-clicking on Macro Error Messages.

See also:

Ami Pro Macro Language Contents
Overview of the Ami Pro Macro Language

Type your macro with any combination of fonts, paragraph styles, and text attributes to make the macro more readable. Ami Pro saves attribute and paragraph style information with the document containing your macro, but none of the attributes or paragraph styles affect how your macro plays. When you use numbers in a macro, you can use the hexadecimal equivalent by preceding the number with "0x" or "0X".

In many of the macro functions, length and size are in twips. There are 1440 twips in an inch (1 inch=1440 twips). Multiply the desired inches by 1440 to determine the size in twips.

Type each macro statement on a single line to improve readability. Use tabs or indented paragraph styles to indent portions of a macro that are parts of a FOR/NEXT loop or an IF/THEN statement.

You can use a double quotation mark (") at the beginning and end of text to represent a string or two double quotation marks ("") to include a single double quotation (") as part of the string.

In addition to treating any text outside a FUNCTION/END FUNCTION pair as a comment, Ami Pro macros treat any text that follows a single quotation mark (') as a comment. The single quotation mark can be at the beginning of a paragraph, causing Ami Pro to skip the entire paragraph. It can also follow a statement in a paragraph, causing the rest of the paragraph to be skipped.

The following macro illustrates the use of comments:

```
1  FUNCTION showcomment()
2  'This macro illustrates the use of comments in a macro
3  var = "Print Me"        ' this line assigns a value to a variable
4  var1 = "Print Me, Too"          ' this line also assigns a value to a variable, and this comment is longer than a single line of text.
5  TYPE ("{var}") ' TYPE ("{var1}") var 1 is a comment and not printed.
6  END FUNCTION
```

The output of the macro is as follows.

"Print Me"

Note that the second TYPE function is not acted on, as it is part of a comment.

See also:

Ami Pro Macro Language Contents
Overview of the Ami Pro Macro Language

The macro language consists of a rich variety of programming statements that control the action of the macro. By combining the programming statements with the built-in macro functions and the Ami Pro functions, you can create powerful tools for use with Ami Pro.

There are three components of the macro language: programming statements, macro functions, and Ami Pro functions.

Programming Statements
These statements control your macro. This section documents these control statements.

Macro Functions
These functions allow you to interact with the macro user, interact with other applications, and get and send information from and to Ami Pro. These functions are generally not available in Ami Pro menus.

Ami Pro Functions
These functions are those that you can execute in Ami Pro by using the menu bar and dialog boxes.

See also:

Ami Pro Macro Language Contents
Overview of the Ami Pro Macro Language
Creating and Saving a Macro

Each macro contains at least one function. A macro function must begin with a FUNCTION statement and end with an END FUNCTION statement. You can have several macro functions within one macro file.

The FUNCTION statement defines the beginning of a macro, names the macro, and defines any arguments passed to the macro. The syntax of a FUNCTION statement is:

FUNCTION Name ([Argument1][[,] Argument2...])

Name is the name of the macro. It must begin with a letter, but can contain both numbers and letters.

Argument1 and **Argument2** are variables that are passed to the macro when the macro runs.

You can pass as many arguments as needed to a macro or none at all. You cannot pass arguments to a macro started through the Tools/Macro/Play function or through a shortcut key. You can only pass arguments to macros you call as functions of other macros. Argument names must be character strings, numbers, or defined variable names as listed in the Using the LET statement section.

The END FUNCTION statement defines the end of a macro function. It causes a return to the calling macro or the end of a macro play. The syntax of an END FUNCTION statement is:

END FUNCTION

Every macro must have an END FUNCTION statement as the last line of the macro. If the macro has more than one exit point, use the EXIT FUNCTION statement at the other exit points in the macro.

The following example is a macro to calculate the average of two numbers and return the result to the calling macro:

```
1  FUNCTION average (p1, p2)
2  average = ((p1 + p2) / 2)
3  END FUNCTION
```

Line 1 assigns the name average to this macro. The calling macro passes the arguments p1 and p2 to this macro. The second line of the macro calculates the average and the last line ends the function.

See also:

Ami Pro Macro Language Contents
Overview of the Ami Pro Macro Language
To Use Multiple Macros in a Single File
Returning Values From Called Macros

You can have more than one macro in a macro file. Keeping related macros in the same file improves the speed of macro playback. To have more than one macro in a macro file, use several FUNCTION and END FUNCTION pairs within a single file. The macro function that executes when you run the macro is the first macro in the macro file.

Text that appears outside the FUNCTION/END FUNCTION pair is comments and therefore ignored; however, Ami Pro recognizes the DECLARE and DEFINE statements. Use the comment format to describe the functionality of the macro.

The following macro functions ask the user to provide two numbers. The function average uses the numbers to obtain an average, and the function main displays the results.

This macro asks the user for 2 numbers, then displays their average.

```
1  FUNCTION main()
2  num1 = Query$ ("Type the first number:")
3  num2 = Query$ ("Type the second number:")
4  result = CALL average (num1, num2)
5  message ("The average of the two numbers is {result}")
6  END FUNCTION
```

This subroutine calculates the average of two passed numbers.

```
1  FUNCTION average (p1, p2)
2  average = ((p1 + p2) / 2)
3  END FUNCTION
```

The function main asks for the two numbers to average using the Query$ function. It then calls the function average to determine the average and displays the result using the Message function. The function average returns the result to the main function by assigning the average to the variable average within the called macro. If the return value is different from the name of the function, use the RETURN statement.

See also:

Ami Pro Macro Language Contents
Overview of the Ami Pro Macro Language

The EXIT FUNCTION statement allows a macro to stop before reaching the end of the macro. A macro usually ends with an END FUNCTION statement. You can use the EXIT FUNCTION statement when there are several different exit points to the same macro. While there can be only one END FUNCTION statement in a macro function, there can be multiple EXIT FUNCTION statements.

The syntax of the EXIT FUNCTION statement is:

EXIT FUNCTION

The following example uses the EXIT FUNCTION statement to exit the macro before the macro ends:

```
1   FUNCTION saveit()
2   ' This macro saves a file only if the file has changed. Otherwise, it cancels the save
3   stat = FileChanged (0, 0)       ' get file changed status
4   IF stat = false             ' file hasn't changed
5           EXIT FUNCTION         ' quit; no need to save
6   ENDIF
7   save()          ' save file
8   END FUNCTION      ' end it
```

In this example, the EXIT FUNCTION statement in line 5 exits the macro if the file does require saving. The END FUNCTION statement in line 8 exits the macro following the saving of the file.

See also:

Ami Pro Macro Language Contents
Overview of the Ami Pro Macro Language
Returning Values From Called Macros

You can call most functions with or without parameters. If you do not pass parameters, Ami Pro displays the dialog box as if the user accessed the function through the menu system. For example:

```
1   FileOpen("DOC.SAM", 1, "") ' opens and displays the file DOC.SAM
2   FileOpen ' brings up the File/Open dialog box
```

You can access some functions that are several levels deep in the menu system. For example, there is a command to insert a glossary record. Executing this function brings up the Insert Glossary Record dialog box.

This language includes macro menu commands so you can develop replacement menu bars for Ami Pro. You can use the menu only commands in these macros.

Several functions generate child processes of Ami Pro. These include the FileManagement, Help, and ControlPanel functions. When you use these functions, the appropriate window opens and the program runs--therefore, you should end the macro immediately after the function call or use a pausing function. If your macro executes statements after one of these windows opens, the Ami Pro window may display on top of the function's window.

See also:

Ami Pro Macro Language Contents
Calling Ami Pro Functions

You can use both local single element variables and array variables in macro programs, but only in the macro that creates them. The macro language does provide functions that allow for creation and maintenance of global variables that any macro can use during an Ami Pro session.

In Ami Pro, you can access global variables by name or by number. Ami Pro does not have different variable types, such as integers, floating point numbers, and characters. Ami Pro stores each variable as a string and converts the string to the appropriate numeric type when running the macro.

When you reference a variable in an arithmetic expression, Ami Pro converts it to integer or floating point format and then processes it. Because Ami Pro does not evaluate variable contents until needed, variables that do not contain numbers cause run-time errors if you use them in arithmetic expressions.

Before you can use a variable in the Ami Pro macro language, you must define it. By defining it, you tell the macro language that the variable exists. While you must formally dimension array variables, you can formally or implicitly declare single element variables. The limit for string variables is 498 characters.

You can pass the **address** of a variable or array to a routine. This allows a subroutine to modify the calling routine's variable. This is useful when a subroutine needs to "return" more than one value. To pass its address use the "&" directly before the variable, for example:

```
passvar(&myvar)
```

This would pass the address of the variable myvar to subroutine passvar. Passvar does not declare this variable in any special way; however, to access it, the reference needs to have an "*" before the variable. The asterisk (*) and the ampersand (&) denote indirection, for example:

```
1  FUNCTION passvar(pv)
2  *pv=*pv+1
3  END FUNCTION
```

Here, passvar increments the caller's variable by one.

To allow for variable indirection, use a semicolon to denote an end of statement. It is optional except when you use indirection. For example:

```
y=2;
*p1="text"
```

See also:

- Ami Pro Macro Language Contents
- Overview of the Ami Pro Macro Language
- Declaring Formal Variables Using the DEFSTR and DIM Statements
- Understanding the Lifetime and Visibility of a Variable
- Using Global Variables to Hold Values
- Using the LET Statement

Many macro functions accept strings as their arguments. You can incorporate single element variables into strings by surrounding them with curly braces ({ }). Consider the following line of macro code:

TYPE ("The first item on the list is {item}")

In this example, the macro knows when to type the word 'item' and when to use the variable referenced by the name 'item'. By placing curly braces around the variable name when you use it as part of a string, you use the contents of the variable rather than its name.

You can also use curly braces around a single element variable name at any time to improve the clarity of the macro. Using curly braces around variable names indicates that the word inside the braces is a variable name rather than something else.

You cannot use curly braces around array variables.

See also:

Ami Pro Macro Language Contents
Overview of the Ami Pro Macro Language
Understanding the Lifetime and Visibility of a Variable
The TYPE function

You can formally declare single element variables using the DEFSTR statement and array variables using the DIM statement. The most common use for DEFSTR is to declare a variable for use with indirection (&). The syntax for the DEFSTR statement is:

DEFSTR Name1[, Name2]...

Name1 and **Name2** are the names of the variables you wish to define.

The following example illustrates the use of the DEFSTR statement:

```
1  FUNCTION typeami()
2  DEFSTR progname
3  progname = "Ami Pro"
4  TYPE ("The name of this program is {progname}.")
5  END FUNCTION
```

In this example, progname is a single element variable in line 2. Line 3 assigns progname the value "Ami Pro", and in the fourth line, the TYPE function types a message to the screen with the variable name and other text.

The DIM statement does the same thing that the DEFSTR statement does, except that it defines an array variable, rather than a single element variable. It includes an expression that allows you to declare the number of elements in the array. Ami Pro evaluates the DIM statement each time and updates the contents of the array. The syntax of the DIM statement is:

DIM array1 (count1)[array2 (count2)]...

Array1 and **array2** are the names of the arrays to be dimensioned.

Count1 is the maximum number of elements in array1.

Count2 is the maximum number of elements in array2.

The following example illustrates the use of the DIM statement to define an array:

```
1  FUNCTION typenames()
2  DIM names(4)
3  names(1) = "Cassie Conners"
4  names(2) = "David Zane"
5  names(3) = "Cris Walker"
6  names(4) = "Peyton Myers"
7  FOR count = 1 to 4
8        curname = names(count)
9        TYPE ("Name number {count} is {curname}.[ENTER]")
10 NEXT
11 END FUNCTION
```

Line 2 of this macro declares that there are four elements in the array names. Lines 3 — 6 assign each element of the array a string value. Line 7 sets up a FOR/NEXT loop with four iterations. Line 8 assigns the value of the current element in the names array to the variable curname. Then, line 9 types the text into the current file, followed by the ENTER key, which ends the paragraph. Lines 10 and 11 end the FOR loop and the macro.

You do not have to declare single element variables formally. You can implicitly declare a single element variable by assigning it a value within the macro. You cannot implicitly dimension array variables. The only way to tell the macro program how many elements to expect in the array is to dimension it.

```
1  FUNCTION main ()
2  your name = Query $ ("What is your name?")
3  Message ("Your name is {your name}.")
4  END FUNCTION
```

This macro implicitly declares the single element variable, your name, on Line 2. Line 3 displays a dialog box with the variable your name.

See also:

Ami Pro Macro Language Contents
Overview of the Ami Pro Macro Language
To Identify Strings as Variables
Understanding the Lifetime and Visibility of a Variable

The **lifetime** of a variable is the time that the variable maintains its value while running a macro. The **visibility** of a variable refers to macro routines that recognize the variable.

Within Ami Pro, each variable is visible only to the macro that defined it and lasts only for the duration of that macro. This means that if one macro calls another macro as a function, the variables in the calling macro are not available to, and cannot be modified by, the called macro. In addition, if a macro finishes running with certain values stored in variables, running the macro a second time reinitializes the variables and loses previously existing values.

When one macro calls another macro as a function, it can pass arguments to the called macro, which are usually variables from the parent macro. When this happens, Ami Pro makes a copy of each variable and passes the copy to the function. Since the function only receives a copy of the original, the function cannot modify the original variable. The function can return a value to the calling macro, and you can assign that value to a variable within the calling macro. The following example illustrates passing variables to functions:

```
1   FUNCTION main()
2   var1 = "original value, main macro"
3   TYPE ("{var1}[ENTER]")
4   CALL upcase (var1)
5   TYPE ("{var1}[ENTER]")
6   var1 = CALL upcase (var1)
7   TYPE ("{var1}[ENTER]")
8   END FUNCTION

9   FUNCTION upcase( var2)
10 TYPE("original value, upcase macro[ENTER]")
11 var3 = UCASE$( var2)
12 TYPE ("{ var2}[ENTER]")
13 upcase =   var3
14 END FUNCTION
```

The output from the above macro is as follows:

```
1   original value, main macro
2   original value, upcase macro
3   original value, main macro
4   original value, main macro
5   original value, upcase macro
6   original value, main macro
 7   ORIGINAL VALUE, MAIN MACRO
```

The first line of output comes from line 3 of the main macro, and displays the original value of var1 as defined in the main macro. Line 3 of the output comes from line 13 of the upcase macro. Even though var2 passes as an argument to the UCASE$ function (which uppercases the argument), its value has not changed. Line 4 of the output is from line 5 of the main macro, and illustrates again that passing an argument to another macro does not affect the value of that argument once it has returned to the main macro.

Lines 5 and 6 of the output show another iteration of the upcase macro, and have the same results as occurred the first time through. Line 7 of the output is from line 7 of the main macro, and shows that the

var1 string from the main macro became uppercase when assigned the result of the upcase macro in line 6.

See also:

Ami Pro Macro Language Contents
Overview of the Ami Pro Macro Language
Using Global Variables to Hold Values
To Use Multiple Macros in a Single File

Global variables are like safety deposit boxes that you can use to store information when it is not in use. When a macro needs to store information for later use, it creates a global variable and assigns the value of the program variable to the global variable. Later, another macro, or another instance of the same macro, can retrieve the value of the global variable, assign it to a variable name in the macro, and use or change it as needed.

See also:

Ami Pro Macro Language Contents
Overview of the Ami Pro Macro Language
Understanding the Lifetime and Visibility of a Variable
To Use Multiple Macros in a Single File

An **expression** is a combination of numbers, strings, variables, and operators used to determine a result. You can assign the result to a variable, use it as an argument to a function or macro, return it as the result of a macro, or use it to determine whether or not to execute a macro routine.

Some examples of expressions are:

```
1  var
2  var + var1
3  var1 | var2
4  var * 3 >var2
5  var < 10 AND var >0
```

In line 1, the expression is a single element variable. If you do not use operators in the expression, the result of the expression is the variable or constant. In line 2, the expression uses the mathematical operator + to add two variables together. Line 3 uses the bitwise OR operator | to "OR" the bit values of var1 and var2. Line 4 uses the relational operator > to compare two values. Line 5 uses the logical operator AND to compare two logical values to determine the result.

Operators are either mathematical, relational, or logical, and operate on the other elements of the expression to determine the result.

The four types of operators are Mathematical Operators, Bitwise Operators, Relational Operators, and Relational Operators.

See also:

Ami Pro Macro Language Contents
Overview of the Ami Pro Macro Language
Understanding Operator Precedence

Ami Pro evaluates expressions with multiple operators by first performing operations with a higher precedence, followed by operations with a lower precedence. If two operations have the same precedence, Ami Pro evaluates from left to right. The precedence of operators is:

NOT

Multiplication and Division

Addition and Subtraction

Bitwise AND and Bitwise OR

Greater Than, Less Than, Greater Than or Equal To, Less Than or Equal To

Equal To and Not Equal To

AND and OR

You can use parentheses to determine the order of expression evaluation. If you use parentheses, Ami Pro first evaluates expressions inside the innermost set of parentheses, followed by the next set of parentheses, etc.

If you use two operators in a row, you must separate the operators by parentheses.

The following examples illustrate expression evaluation. Some of the examples use parentheses to change the evaluation order.

```
var > var1    ' is true if var > var1
var = 4       ' is true if var = 4
name < "Jones"       'is true if name is anything besides Jones
var + var1 <= var2 * var3
var > var1 AND var1 < 10
var <>0 OR (NOT IsNumeric(var))       ' two operands in a row
' determines if the variable c is alphabetic
(c >= "A" AND c <= "Z") OR (c >= "a" AND c <= "z")
' true if var3 >0 or if var = 0 and either var1 or var2 is numeric
var = 0 AND (IsNumeric(var1) OR IsNumeric (var2)) OR var3 >0
' true if var = 0 and either var1 or var2 is numeric or var3 >0
var = 0 AND ((IsNumeric(var1) OR IsNumeric (var2)) OR var3 >0)
```

See also:

Ami Pro Macro Language Contents
Overview of the Ami Pro Macro Language

Mathematical operators include addition, subtraction, multiplication, and division. The mathematical operators are:

Name (Character) - Definition

Multiplication (*) - Multiplies the value to the left of the operator by the value to the right of the operator.

Division (/) - Divides the value to the left of the operator by the value to the right of the operator.

Addition (+) - Adds the value to the left of the operator to the value to the right of the operator.

Subtraction (-) - Subtracts the value to the right of the operator from the value to the left of the operator.

Values used by mathematical operators must be numeric otherwise, a runtime macro error can result.

See also:

Ami Pro Macro Language Contents
Overview of the Ami Pro Macro Language

Bitwise operators include the bitwise AND and the bitwise OR. The bitwise operators are:

Name (Character) - Definition

Bitwise AND (&) - Performs a logical AND between each bit of the value to the left of the operator and the value to the right of the operator.

Bitwise OR (|) - Performs a logical OR between each bit of the value to the left of the operator, and the value to the right of the operator.

Values used by bitwise operators must be integers or a runtime macro error can result. Bitwise operators, which compare each bit of a value, are not the same as logical operators, which compare the entire value.

See also:

Ami Pro Macro Language Contents
Overview of the Ami Pro Macro Language

Relational operators include equality, greater than, less than, greater than or equal to, less than or equal to, or not equal to. The relational operators are:

Name (Character) - Definition

Equals (=) - Compares the value to the left of the operand and the value to the right of the operand. The result of the comparison is TRUE if the values are equal.

Greater Than (>) - Compares the value to the left of the operand and the value to the right of the operand. The result is TRUE if the value to the left is greater than the value to the right.

Less Than (<) - Compares the value to the left of the operand and the value to the right of the operand. The result is TRUE if the value to the left is less than the value to the right.

Greater Than or Equal To (>=) - Compares the value to the left of the operand and the value to the right of the operand. The result is TRUE if the value to the left is greater than or equal to the value to the right.

Less Than or Equal To (<=) - Compares the value to the left of the operand and the value to the right of the operand. The result is TRUE if the value to the left is less than or equal to the value to the right.

Not Equal To (<> or !=) - Compares the value to the left of the operand and the value to the right of the operand. The result is TRUE if the value to the left is not equal to the value to the right.

Values used by relational operators can be strings, numbers, or expressions.

See also:

Ami Pro Macro Language Contents
Overview of the Ami Pro Macro Language

Logical operators are:

Logical And (AND) - Logically compares the value to the left of the operand with the value to the right of the operand. The result is TRUE if both expressions evaluate to TRUE.

Logical Or (OR) - Logically compares the value to the left of the operand with the value to the right of the operand. The result is TRUE if either expression evaluates to TRUE.

Logical Not (NOT) - Evaluates the value to the right of the operand. The result is TRUE if the expression evaluates to FALSE. The result is FALSE if the expression evaluates to TRUE.

Values used by logical operators can be integers, numbers, strings or expressions. If the value is equal to 0 or to the null string (""), it is FALSE. If it is anything else, it is TRUE.

See also:

Ami Pro Macro Language Contents
Overview of the Ami Pro Macro Language

You can assign the result of an expression to a single element variable or to an element of an array. Use the LET statement to make a variable assignment. The syntax of the LET statement is:

[LET] Var = Expression

Var is the variable name receiving the assignment.

Expression is a valid expression as defined above.

The syntax of a variable assignment statement to an array variable is:

[LET] Array(Expr) = Expression

Array is the name of the array receiving the value of the expression.

Expr is an expression that evaluates to the number of the element of the array receiving the value of the expression.

Expression is a valid expression as defined above.

The following are some examples of variable assignment:

```
' simple assignment to value
LET name = "string"
' assignment of one variable to another
name2 = name
' assignment of an array element to a value
array(1) = "String"
' assignment of variable to an array element determined with expression
array (i+5) = name
' assignment to result of a function
var = Query$ ("Type a value")
' assignment to result of a macro
var = CALL mac2()
```

See also:

Ami Pro Macro Language Contents
Overview of the Ami Pro Macro Language

A constant expression is an expression whose value does not change for the duration of the macro program. Normally, constants are strings of text that the macro uses.

There are two types of constants: defined and predefined. You create defined constants in your macro. Ami Pro provides predefined constants you may use in your macro.

A constant expression is normally a string of text in a macro. There are several characters that you need to represent in a special manner so that Ami Pro can recognize them within strings. These characters are the left curly brace ({), the left square brace ([), and the double quotation mark (").

Because curly braces surround variable names in strings, you must type two curly braces to type a curly brace within a string. Use the TYPE function to represent key names in strings. Since square braces surround a key name, you must type two square braces to represent a square brace within a string.

Double quotation marks surround strings to define their boundaries. To type a double quotation mark within a string, you must type two quotation marks in a row. You do not need to type two right curly braces or two right square braces when you are using them in strings. The following macro illustrates the representation of strings within a macro:

```
1   FUNCTION strings()
2   'a string assigned to a variable
3   var = "Variable"
4   'a typical string
5   TYPE ("This is a constant string of text. ")
6   'a string with a key name
7   TYPE ("This string has a key name inside it.[ENTER]")
8   'a string with a variable name
9   TYPE ("This string contains a variable with the value {var}. ")
10   'a string with curly braces embedded
11   TYPE ("The curly brace character, {{, is used to fence a variable. ")
12   'a string with a square brace character
13   TYPE ("The square brace character, [[, is used to fence a key name. ")
14   'a string with embedded quotes
15   TYPE ("Use quotation marks like this ""string"" to define a string.")
16   END FUNCTION
```

The output of the macro looks like the following:

This is a constant string of text. This string has a key name inside it.

This string contains a variable with the value Variable. The curly brace character, {, fences a variable. The square brace character, [, fences a key name. Use quotation marks like this "string" to define a string.

See also:

Ami Pro Macro Language Contents
Overview of the Ami Pro Macro Language
Predefining Constants

To make a macro more readable, you can use constants to represent values that do not change in a macro. Ami Pro uses a list of defined constants to perform substitutions when they are used in a macro. This list is most useful when using Ami Pro functions that require numeric arguments be given to them for interpretation. When the macro is compiled, the compiler substitutes the constant in the macro for the number in the list of defined constants, unless the name is inside a string.

Many functions use flags as parameters to the function. A flag is a number that conveys a meaning to the function. By adding numbers together, a number can extract several different values when the macro runs.

The following is an example of using predefined constants in a macro:

```
1  FUNCTION lineson()
2  flag = NumberLines+NumberEveryOther+ResetEachPage
3  ' compiles as flag = 1+4+16, so flag = 21
4  LineNumber (flag, "Body Text")
5  message ("Line Numbering On")
6  END FUNCTION


1  FUNCTION linesoff()
2  LineNumber (off, "Body Text")
3  ' compiles as LineNumber (0,"Body Text")
4  message ("Line Numbering Off")
5  END FUNCTION
```

In this example, the two macros turn line numbering on and off in a file. The lineson macro turns on line numbering every other line. Then it resets the numbering to 1 on each page. The linesoff macro turns off line numbering.

One of the parameters to the LineNumber function is a flag that specifies which line numbering options to use. Ami Pro adds the numbers representing these options together and passes them to the function. The option to turn the numbering on is 1; the option to number every other line is 4, and the option to reset numbering each page is 16. You can use predefined constants to add these numbers and then place the total in the function call. This makes it clear exactly what the macro is doing.

The macro uses the same procedure in the linesoff macro. To turn numbering off, the macro uses a flag of 0. Since off is defined as 0 in the substitution list, this substitution makes the macro more readable.

Ami Pro stores the substitution list in an Ami Pro glossary file named MACDEFIN.SAM in the Ami Pro directory (usually C:\AMIPRO). The file contains substitutions for predefined constant names. You can add to the glossary file. The glossary item name is the predefined constant name in the macro. Ami Pro substitutes the predefined constant name with the item. Neither the defined name nor the replacement can contain spaces. The defined name must begin with an alphabetic character and can contain only letters and numbers.

If you modify this file, macros compiled under the modified MACDEFIN.SAM may not compile on other systems. Copy the MACDEFIN.SAM to the Ami Pro directory (usually C:\AMIPRO) to compile successfully.

See also:

Ami Pro Macro Language Contents
Overview of the Ami Pro Macro Language

You can use the DEFINE statement to do a simple token replacement ("this" item for "that" item). The syntax is:

DEFINE identifier replacement

This replacement allows you to define a descriptive name to a number. For example,

```
DEFINE ALL 1
DEFINE JUSTONE 2
MyFunction(ALL)
        .........
FUNCTION MyFunction(type)
        .........
if (type=ALL)
elseif(type=JUSTONE)
```

When the macro runs, it replaces ALL and JUSTONE by 1 and 2.

A more powerful use of the DEFINE statement is to replace parameters. The syntax is:

DEFINE identifier([optional descriptive parameters]) replacement([parameters])

You can use this format to use a Replace function. For example,

```
DEFINE Findit() Replace(0,0,4,%1,0)
```

Then you enter the text to replace.

```
Findit("text")
```

When the macro is compiled, Findit("text") is replaced with Replace(0,0,4,"text",0). The replacement uses the DOS batch file notation for parameters. If you have more than one parameter they would be %2, %3, etc. You can use these more than once and in any order. For more information about parameters, see your DOS manual.

When Ami Pro encounters the DEFINE, it remembers the complete replacement. When the next occurrence happens, parentheses surround the parameters and separate the parameters by white characters or commas. Parameters can be either quoted strings, a string surrounded by parentheses, or any non-white sequence of characters. Once Ami Pro identifies the parameters, it scans the replacement for %1, %2, etc., and replaces the parameters. The whole string is then re-parsed. The total replacement, after parameter substitution, must be less than 500 characters. An example of calling DLL functions would be:

```
DllCall(Dllid,p1,p2)
```

or use DEFINE:

```
DEFINE LookUp() DLLCall(Dllid,%1,%2)
```

It could be:

```
LookUp(p1,p2)
```

See also:

Ami Pro Macro Language Contents
Overview of the Ami Pro Macro Language

To execute a subroutine, one macro can call another macro and then return a single value to the calling macro. Before a macro can call another macro, you can identify it prior to its first use with the DECLARE statement. You can also identify it with each use by using either the CALL or CALLI statement.

It does not make any difference whether you use the DECLARE statement once at the beginning of a calling macro or you use the CALL or CALLI statement each time in the subroutine.

When executing another macro, Ami Pro follows a search path to find the macro. You do not have to specify a full path to the macro.

You can also call macro functions within a macro. You do not need to declare or call built-in functions. To use a built-in function, use its name. You can call Ami Pro functions by providing the parameters required for the function and having the function execute. You can also call an Ami Pro function by giving the function name without parentheses. This displays the dialog box for the function and then allows the macro user to choose the parameters for the function.

See also:

Ami Pro Macro Language Contents
Overview of the Ami Pro Macro Language
Calling Built-In Functions
Determining the Location of a Macro When it is Run
Calling Ami Pro Functions
Returning Values From Called Macros

The Ami Pro macro language offers a variety of functions you can call from within a macro. The macro language includes functions to control the execution of the macro, to allow interaction with the user, to allow interaction with the Ami Pro program itself, and to allow interaction with other programs using Microsoft Windows Dynamic Data Exchange (DDE) protocol.

To call a built-in function use the syntax below:

[Result =]Function([Parm1][[,] [Parm2...])

Result is a variable that receives the return value of the function.

Function is the name of the function you want to use.

Parm1 and **parm2** are the parameters required by the function, if any.

The following macro uses the built-in functions GetDocPath$, SetDocPath, Decide and Query$ to get the user's default document path, display it, and to allow it to be changed:

```
1  FUNCTION changepath()
2  path = GetDocPath$()          ' gets current path
3  response = Decide ("Change Document Path From {path}?")
4  IF (response)          ' user said yes to dialog box
5          path = Query$ ("Type New Path:")
6          SetDocPath (path)
7  ENDIF
8  END FUNCTION
```

Line 2 of the macro uses the GetDocPath$ function to get the current path. This line stores the result of this function in the variable path. Line 3 of the macro uses the Decide function to display a dialog box and ask the user if he wants to change his path. The Decide function requires one parameter, the prompt you should use in the dialog box. It returns either TRUE or FALSE, depending on whether the user chose Yes or No in the dialog box. This line assigns the return value from the Decide function to the variable response. If the user decided to change the path, the Query$ function in line 5 displays an edit box for the user to type the new path. This line returns the typed value and assigns it to the variable path. Finally, line 6 sets the new document path using the SetDocPath function.

See also:

Ami Pro Macro Language Contents
Overview of the Ami Pro Macro Language

You can execute almost any Ami Pro function using a macro. You can access Ami Pro functions through pulldown menus that have equivalent functions in the macro language.

When using an Ami Pro function, the macro can provide all the functions the program requires, which causes the function to automatically execute. The macro can also start the function and allow the user to make choices for that function in the dialog box. The syntax for calling an Ami Pro function that can execute without user intervention is:

[Result =]Function([Parm1][[,] [Parm2...])

Result is a variable that receives the return value of the function.

Parm1 and **Parm2** are the parameters required by the function, if any.

The syntax for calling an Ami Pro function that prompts the user for parameters is:

[Result =] Function

Function is the name of the function you want to use.

In general, Ami Pro functions return TRUE (1) if the function completes successfully. They return FALSE (0) if the function did not complete because the user canceled the function. If the state of the program prohibited the function at the time or the function could not complete for some other reason, they return GENERALFAILURE (-2).

The following macro sets the line spacing in the current document to single spacing:

```
1   FUNCTION singlespace()
2   result = Spacing(SpaceSingle)
3   IF (result <> TRUE) ' the spacing was not equal to single
4           message ("Failed to set single spacing.")
5   ENDIF
6   END FUNCTION
```

In line 2, the Ami Pro function Spacing is called. The Spacing function takes a single argument, with the desired spacing. SpaceSingle is defined to -1, which is the correct value for single spacing. The result of the function is returned to the variable result. This variable is examined in line 4, and if the spacing change did not occur, an error message is displayed for the user.

The following macro sets line spacing but allows the user to determine the desired spacing:

```
1   FUNCTION setspace()
2   result = Spacing
3   IF (result = false)
4           message ("Did not set spacing")
5   ENDIF
6   END FUNCTION
```

In line 2, the macro calls the Spacing function without any parentheses. This causes the Line Spacing dialog box to display on the screen so the user can set the spacing. As in the previous example, this line assigned the result of the function call to the variable so that a message can display if the spacing did not change.

See also:

Ami Pro Macro Language Contents
Overview of the Ami Pro Macro Language

Ami Pro uses a search path to find the location of the macros to execute. If a macro contains a full path specification, Ami Pro looks there. If there is no path specified, Ami Pro searches first the default macro directory, second the default document directory, and third the directory where Ami Pro stores program files (usually C:\AMIPRO). If Ami Pro can not find the macro in any of these locations, Ami Pro displays an error message.

In an Ami Pro installation the default macro path is a subdirectory of the AMIPRO directory (usually C:\AMIPRO\MACROS). This directory may contain several example macros. Whenever the macros dialog box displays, it defaults to displaying macros in that directory. You can change this path by modifying the path in User Setup/Paths, by editing the AMIPRO.INI file, or by using the macro functions GetProfileString$ and WriteProfileString.

If you write macros that other people may use, you may not know the final location of the macro file. If the macro calls another macro, assigns a macro to a menu to call later, or uses an external dialog box, then you may not be able to specify the path to the desired file. The macro search path allows you to specify only the file name and macro name and allows Ami Pro to find the macro file.

Ami Pro does not use this search path when looking for an external dialog box, a dynamic link library (DLL), or other supporting files. If these supporting files are in the same directory as the macro that needs them, you can use the GetMacPath$ function to determine the default macro path. You can then use that path to locate the supporting macro files. To determine the name of the macro that is running, use the macro function, GetRunningMacroFile$.

See also:

Ami Pro Macro Language Contents
Overview of the Ami Pro Macro Language

You can use the DECLARE statement to identify callable macros before you call the macro. The syntax of the DECLARE statement is:

DECLARE MacroName([Parm1][[,] Parm2...])[ALIAS ShortName]

MacroName is the name of the macro that you want to call. It may contain a file name. If it does not, the macro must be in the current file.

Parm1 and **parm2** are prototypes of the parameters that pass to the macro when called. They must be variable names, but you do not need to declare them.

ALIAS ShortName is the name of the alias to use instead of the file name.

As an example, type the following statement:

DECLARE Heapsort.smm!Sort(p1,p2) ALIAS sort

When you need to sort later you would then just type:

Sort(&Names,Count)

Using the DECLARE statement does not cause the macro referenced to execute; it simply identifies a macro that you can use within the calling macro. Once declared, a macro can execute from another macro by typing its name and parameters.

See also:

Ami Pro Macro Language Contents
Overview of the Ami Pro Macro Language
The CALL and CALLI statements
Example of the DECLARE, CALL and CALLI Statements

The CALL and CALLI statements execute one macro from another macro. They differ in that you use the CALL statement when you know the name of the macro to execute when the macro compiles. You can use the CALLI statement when you do not know the name until you run the macro.

The syntax of the CALL statement is:

CALL [[Drive:Path\]MacroFile!]Function([Parm1][[,] Parm2...])

drive, **path**, and **MacroFile** are the drive letter, full path, and macro file name that contain the called macro.

MacroName is the name of the macro to execute.

Parm1 and **parm2** are the parameters to pass to the macro.

You must supply the parentheses even if you are not passing any parameters to the called function. For example, CALL mymacro.smm!myfunction()

The syntax of the CALLI statement is:

CALLI Variable ([Parm1][[,] Parm2...])

variable is a string variable that evaluates to the name of a macro. The name of the macro may contain the drive, path, and file name as outlined above for the CALL statement.

Parm1 and **Parm2** are the parameters to pass to the macro.

You must supply the parentheses even if you are not passing any parameters to the called function. For example, CALLI mytestfunction()

The parameters given can be any expression.

You must follow the drive, path, and macro file name, if given, by an exclamation point and the name of the macro. You do not need the drive, path, and macro file name if the called macro is in the same file as the calling macro. If given, Ami Pro looks for the macro in the location of the macro path. If no path is given, Ami Pro searches the macro directory, the document directory, and the Ami Pro directory for the called macro.

If the disk location of the called macro is not on one of the locations in the macro search path, and must be determined at runtime, the CALLI statement should be used instead. The only difference between the CALL and the CALLI statements is that the macro name and path can be specified in a variable when the CALLI statement is used, while they must be specified explicitly when using the CALL statement.

See also:

Ami Pro Macro Language Contents
Overview of the Ami Pro Macro Language
Example of the DECLARE, CALL and CALLI Statements

The following example macro illustrates the use of the DECLARE, CALL and CALLI statements. This macro determines either the number count of the letters in a word, or the number of digits in a number.

```
1   DECLARE getopt()   ' declare this function so we can use it later

2   FUNCTION main()
3   string = Query$ ("Type a word or a number:")
4   flag = getopt()          ' since declared, do not have to use CALL statement
5   IF (flag = 6)
6       result=CALL sub1 (string) ' since macro is here, just use CALL statement
7       message ("There are {result} letters in {string}.")
8   ELSE
9       thismacro=GetRunningMacroFile$()
10        macname = strcat$(thismacro, "!SUB2")      ' put into var
11        result=CALLI macname (string)           ' use variable cause do not know filename
12        message ("There are {result} digits in {string}.")
13  ENDIF
14  END FUNCTION

1   FUNCTION sub1 (string)       ' this is in original file
2   sub1 = LEN(string)
3   END FUNCTION

1   FUNCTION getopt()  ' this is in original file
2   getopt = MultiDecide ("Choose Yes for letters; No for numbers", 35)
3   END FUNCTION

1   FUNCTION sub2 (string)       ' this is called by the CALLI funciton in Main
2   numcount = 0
3   FOR count = 1 to LEN(string)
4       c = MID$(string, count, 1)
5       IF (isnumeric(c))
6           numcount = numcount +1
7       ENDIF
8   NEXT
9   sub2 = numcount
10  END FUNCTION
```

This macro examines a word or number and returns the total number of characters or digits. In the main function, line 3 retrieves a string. The getopt function, declared on Line 1, is used on Line 4 just as if it were an Ami Pro function. This function presents a MULTIDecide box to determine if the string is a word or a number. If the MULTIDecide box returns a "yes," Line 6 calls the sub1 function, which returns the number of characters in the string. In Line 7, the returned value is displayed. Otherwise, in line 10 the

macro calls the variable function (which in this case evaluates to the current macro filename followed with "!sub2"). Sub2 computes and returns the number of digits in the string. In Line 12, the returned value is displayed.

See also:

Ami Pro Macro Language Contents
Overview of the Ami Pro Macro Language
Determining the Location of a Macro When it is Run
GetMacPath$

A called macro can return a value to the calling macro. The returned value can be a number or a string or it can be a status code to indicate the macro successfully completed.

You should assign the value to return to the calling macro to a variable name with the same name as the called macro. When the called macro ends at the END FUNCTION or EXIT FUNCTIONTO_USE_THE_EXIT_FUNCTION_STATEMENT statement, the value of the variable with the same name as the called macro returns to the calling macro.

The following is an example of returning a value from a called macro to the calling macro:

```
1   FUNCTION calc()
2   'this macro gets two numbers from the user, adds them together and displays the result.
3   num1 = Query$ ("Type the first number:")
4   num2 = Query$ ("Type the second number:")
5   result = CALL addit (num1, num2)
6   result2 = CALL subtractit (num1, num2)
7   message ("The sum of {num1} and {num2} is {result}.")
8   message("The difference between {num1} and {num2} is {result2}.")
9   END FUNCTION

10   FUNCTION addit(num1, num2)
11   addit = num1 + num2
12   END FUNCTION

13   FUNCTION subtractit(num1, num2)
14   Difference = num1 - num2
15   RETURN(Difference)
16   END FUNCTION
```

In lines 3 and 4, the calc macro uses the Query$ function to get the two numbers from the user. It then calls the addit macro in line 5, and stores the return value in the variable result. Finally, the result displays in the message function in line 7.

The addit macro assigns the sum of the two variables to the variable addit and then ends. By assigning the sum to the variable with the same name as the macro, its value returns to the calc macro.

The Subtractit macro assigns the difference of the two variables to the variable subtractit and then ends. Its value returns to the Calc macro. By using the RETURN statement, the macro assigns the return value to a variable name that is different from the name of the called macro.

If you need to have more than one value returned from a called function, use indirection (&). It passes the memory address of a variable to the called function. The called function may then directly modify the variable. When the called macro finishes, any modifications to the variable are reflected. You can pass any number of variables in this manner.

See also:

Ami Pro Macro Language Contents
Overview of the Ami Pro Macro Language

The Ami Pro macro language provides a method of transferring control to another section of the same macro. This is useful when the macro needs to continue from another location and it is not necessary to return to the original location. In addition, Ami Pro provides special commands to transfer control to an error handling routine and a user cancel handling routine.

You can use a label to define the location where the macro transfers control from the GOTO statement. The label has the following syntax:

Label:

Label is a single word followed immediately by a colon. Labels cannot be function names or statement names. The following examples show some permissible label names:

TopOfLoop:

Any1:

proceed:

See also:

Ami Pro Macro Language Contents
Overview of the Ami Pro Macro Language
Using the GOTO Statement

You can use the GOTO statement to unconditionally transfer control to another location in a macro program. The location that the GOTO statement transfers control to must be in the same macro as the GOTO statement and defined by the Label. The syntax of the GOTO statement is:

GOTO Label

Label is a label name defined elsewhere in the macro program. When you use it in the GOTO statement, do not type a colon following the label name.

The following example shows the use of the GOTO and Label statements:

```
1   FUNCTION main()
2   string = Query$ ("Type a word or a number:")
3   flag = (MultiDecide ("Choose Yes for letters; No for numbers", 35))
4   IF (flag = 6) GOTO letters
5   ELSEIF (flag = 7) GOTO numbers
6   ENDIF
7   letters:
8   result = LEN(string)
9   message ("There are {result} letters in {string}.")
10   GOTO endit
11   numbers:
12   numcount = 0
13   FOR count = 1 to LEN(string)
14         c = MID$ (string, count, 1)
15         IF (isnumeric(c))
16               numcount = numcount +1
17         ENDIF
18   NEXT
19   message ("There are {numcount} digits in {string}.")
20   endit:
21   END FUNCTION
```

This macro is a variation of the example used to illustrate the CALL and DECLARE statements. Instead of using subroutines to get the user's choice of counting letters and numbers and to do the count, this macro first determines the user's choice by using the Query$ function in line 2. If the user chooses letters, the GOTO statement in line 4 transfers control to the letters label in line 7. If the user chooses numbers, the program uses the GOTO statement in line 5 to jump to the numbers label in line 11.

The macro requires a GOTO statement in line 10 to jump to the end of the macro so that the letters option does not also execute the numbers option.

Using GOTO statements to control program flow makes a macro more difficult to read, and increases the risk of causing an error if you edit the macro. You can minimize these problems by using macro subroutines and other program control statements.

See also:

Ami Pro Macro Language Contents
Overview of the Ami Pro Macro Language

A macro runtime error can occur in several circumstances, including calling a macro that cannot be found, or passing a non-numeric string to a function that requires a number. If this happens, the macro program normally terminates with an error message that identifies the cause of the error. By using the ONERROR statement, you can direct program control to an error handling routine that could inform the user of the error, and suggest corrective action and/or perform a cleanup. The syntax of the ONERROR statement is:

ONERROR Label

Label is the name of a label to which you should transfer control in case of error.

Another important function of the ONERROR statement is to allow the macro to clean up after itself. This is particularly important if the macro uses Global Variable Functions or ASCII File Functions. Exiting a macro without clearing these routines could affect the later operation of Ami Pro and Windows.

If the macro encounters an error condition, Ami Pro looks for an ONERROR statement in the currently running macro function. If Ami Pro finds no ONERROR, it searches succeeding parent functions for an ONERROR statement, until it encounters the main function. This means that each macro function can have its own error processing.

If an error occurs in a subroutine and the macro processes it in that routine, Ami Pro does not search any farther for other ONERROR statements. You should ensure that parent functions correctly handle error conditions.

The user can cancel the running of a macro by typing a key while the macro is playing back or by choosing Cancel in the Query$ function dialog box. Normally, canceling the macro terminates the macro with no notice to the macro user. By using the ONCANCEL statement, you can inform the user that the macro has canceled or you can cause the execution of the macro to continue elsewhere. The syntax of the ONCANCEL statement is:

ONCANCEL Label

Label is the name of a label to which control transfers in case the user cancels the macro.

If the macro encounters an error or cancel condition, Ami Pro looks for an ONCANCEL statement in the currently running macro function. If Ami Pro finds no ONCANCEL, it searches succeeding parent functions for an ONCANCEL statement, until it encounters the main function. This means that each macro function can have its own error processing.

If an error occurs in a subroutine, and the macro processes it in that routine, Ami Pro does not search any farther for other ONCANCEL statements. You should ensure that parent functions correctly handle error conditions.

The following macro gives an example of the ONERROR and ONCANCEL statements:

```
1   FUNCTION main()
2   ONERROR ErrorRoutine 'set error routine
3   ONCANCEL CancelRoutine 'set cancel routine
4   Query$ ("Choose OK or CANCEL.") ' set up routine
5   CALL notthere() ' this causes an error, since macro not there
6   GOTO endit
7   CancelRoutine:
8   message ("You canceled the macro.")
9   GOTO endit
10   ErrorRoutine:
11   message ("There has been a macro error.")
12   endit:
13   END FUNCTION
```

This macro defines program locations for error and cancel handling in lines 2 and 3. It then asks the macro user to type in a value using the Query$ function, which has OK and Cancel buttons. If the user chooses OK, the macro attempts to call a nonexistent macro named notthere in line 5. This causes a jump to the error routine in line 10. If the user chooses cancel in the Query$ function, the program jumps to the cancel routine in line 7.

When using the ONERROR and ONCANCEL statements, make sure that they are at the beginning of the macro. If an error occurs before these statements, control may not transfer to the error or cancellation routines.

See also:

Ami Pro Macro Language Contents
Overview of the Ami Pro Macro Language
Using the GOTO Statement

Frequently, the value of a variable determines the next step a macro should take. At other times, the macro should complete a series of steps a number of times and then the program should execute other steps. The macro language provides several statements that can control macro execution based on the value of expressions.

The available statements include FOR/NEXT, which executes a loop a defined number of times; IF/THEN, which executes statements only if an expression is TRUE; the SWITCH/CASE statements, which execute a set of steps based on the value of a variable; and the WHILE/WEND statement that repeatedly executes a series of steps as long as an expression evaluates to TRUE.

See also:

Ami Pro Macro Language Contents
Overview of the Ami Pro Macro Language
Macro Programming Statements

The FOR statement executes a series of program steps one or more times or skips the program steps. The syntax of the FOR statement is:

FOR Assignment TO Expression1
[STEP Expression2]

Program statements

NEXT

Assignment is an assignment of value to a single element variable.

Expression1 is an expression to which the value of the single element variable is compared.

Expression2 is an expression that is added to the single element variable once for each loop iteration.

Program statements are the macro statements executed if the value of the single element variable is less than expression1 at the beginning of each loop.

When Ami Pro encounters the FOR statement, the variable specified in the assignment initializes to the value of the assignment. Ami Pro compares this variable to expression1. If the variable is less than or equal to expression1, Ami Pro executes the program steps between the FOR statement and the NEXT statement. If the result of the comparison is false, the program continues with the first statement following the NEXT statement.

When the macro executes the NEXT statement, control returns to the FOR statement. The variable increments by the amount specified in expression2. If there is not an expression2, the variable increments by 1. Ami Pro compares the new value of the variable to expression1. If the two values are not equal, program steps execute again. This process continues until the value of the variable is greater than expression1 and program control passes to the first statement following the NEXT statement.

The following macro illustrates the use of the FOR and NEXT statements:

```
1  FUNCTION typename()
2  FOR loop1 = 1 to 5
3       TYPE("Ami Pro ")
4  NEXT
5  END FUNCTION
```

Line 2 initializes the variable loop1 to 1. Line 3 types "Ami Pro " to the screen, excluding the quotation marks. Line 4 transfers the control back to Line 2. Line 2 checks to see if loop1 is equal to 5. If loop1 is less than 5, the macro increments loop1 by 1 and continues to Line 3. The loop continues for a total of 5 iterations. When loop1 is equal to 5, the macro ends.

See also:

Ami Pro Macro Language Contents
Overview of the Ami Pro Macro Language
Macro Programming Statements

The IF/THEN statement executes a series of steps if a condition is TRUE. IF/THEN statements can test several conditions and execute a particular series of steps for the appropriate condition. The syntax of the IF/THEN statement is:

IF Condition1 [THEN]

Program step 1

[ELSEIF Condition2 [THEN]

Program step 2]...

[ELSE

Program step 3]

ENDIF

Condition 1 is an expression that causes program step 1 to execute if it evaluates to TRUE.

Program step 1 is a series of macro statements that execute if condition1 evaluates to TRUE.

Condition 2 is an expression, that causes program step 2 to execute if it evaluates to TRUE.

Program step 2 is a series of macro statements that execute if condition2 evaluates to TRUE and previous conditions are FALSE.

Program step 3 is a series of macro statements that execute if none of the previous conditions were TRUE.

When the macro encounters the IF statement, the macro evaluates the condition specified in condition1. If the condition evaluates to TRUE, Program step 1 executes and control passes to the first program line beyond the ENDIF. If the condition is FALSE, the macro evaluates to condition2.

If this condition is TRUE, then Program step 2 executes and control passes to the first program line beyond the ENDIF. The macro evaluates subsequent ELSEIF's (if present) until one of the ELSEIF conditions is TRUE and its statements execute or the ELSE statement, if present, executes.

The following macro deletes two carriage returns in a row in a document. This allows removal of the blank line between paragraphs. It uses the IF statement to find two carriage returns in a row.

```
1   FUNCTION delcrs()
2   TYPE ("[ctrlhome]") ' go to top of file
3   loop:
4   TYPE ("[ctrldown]") ' to next RETURN symbol
5   c = CurChar$()
6   TYPE ("[RIGHT]")
7   IF (AtEOF()) ' at end of file?
8           EXIT FUNCTION 'end macro execution
9   ELSEIF (c = CurChar$()) ' look for two in a row
10          TYPE ("[DEL]")
11  ENDIF
12          GOTO loop
13  END FUNCTION
```

The macro starts by going to the beginning of the file in line 2. It then moves to the end of the first paragraph in line 4. In line 5, the macro stores in the variable c the value of the character under the insertion point. The insertion point then advances one position in line 6. The IF statement in line 7 determines if the macro reached the end of the file. If so, the macro exits in line 8. If not, the ELSEIF statement in line 9 compares the value of the character at the insertion point position with the value of the previously stored character c. If they are the same (the condition is TRUE), line 10 deletes the second

return symbol. There is no ELSE condition in this IF statement, since we are not interested in any other characters besides return symbols and the end of the file. Therefore, control passes through the ENDIF statement and jumps to the label LOOP in line 3, where the sequence repeats.

See also:

Ami Pro Macro Language Contents
Overview of the Ami Pro Macro Language
Macro Programming Statements

The SWITCH and CASE statements work together to execute a series of program steps based on the value of a single variable. These statements are similar to the IF/ELSEIF statements. The major difference is that while you can execute different statements based on the values of different variables using the IF statement (IF a = 1 then ... ELSEIF b = 2 ... ELSEIF c = 3 ...), all of the statements executed with a CASE statement depend on the variable defined in the SWITCH statement. The syntax for the SWITCH statement is:

SWITCH Variable

    CASE Expression1

        Statements1

    [CASE Expression2

        Statements2...]

    [DEFAULT

        Statements3]

ENDSWITCH

Variable is a defined variable name that is compared to the expressions in the CASE statements.

Expression1 is an expression that is compared to Variable.

Statements1 are program statements that are executed if Variable is equal to Expression1.

Expression2 is an expression that is compared to Variable.

Statements2 are program statements that are executed if Variable is equal to Expression2.

Statements3 are program statements that are executed if none of the expressions in any CASE statement is equal to Variable.

When the macro encounters the SWITCH statement, the value of Variable is stored. This value is then compared to the expression in the first CASE statement. If they are equal, the statements between that CASE statement and the next CASE statement, the DEFAULT statement, or the ENDSWITCH statement, whichever comes first, is executed. Control then passes to the first statement beyond the ENDSWITCH statement. If the variable and the CASE expression are not equal, the statements are not executed. The program continues to the next CASE statement, which is evaluated in the same way.

If none of the CASE expressions matched the variable by the time the DEFAULT statement is encountered, the statements between the DEFAULT and ENDSWITCH statements are executed. If there is no DEFAULT statement, control passes to the first statement beyond the ENDSWITCH statement.

See also:

Ami Pro Macro Language Contents
Overview of the Ami Pro Macro Language
Macro Programming Statements
Example of The SWITCH and CASE Statements

The following macro counts the frequency of occurrence of four words in a document and reports the percentage of each word used:

```
1   FUNCTION wordfreq()
2   word1 = Query$ ("Type the first word:")
3   word2 = Query$ ("Type the second word:")
4   word3 = Query$ ("Type the third word:")
5   word4 = Query$ ("Type the fourth word:")
6   DraftMode()' put in DraftMode
7   TYPE ("[ctrlhome]") ' go to top of file
8   w1cnt = 0 ' initialize counters
9   w2cnt = 0
10   w3cnt = 0
11   w4cnt = 0
12   ocnt = 0
13   loop: 'loop point for each word
14   word = CurWord$()     ' get word contents
15   SWITCH word
16          CASE (word1) ' if a match
17                w1cnt = w1cnt + 1 ' bump word count
18          CASE (word2)
19                w2cnt = w2cnt + 1
20          CASE (word3)
21                w3cnt = w3cnt + 1
22          CASE (word4)
23                w4cnt = w4cnt +1
24          default
25                ocnt = ocnt +1 ' didn't match our words
26   ENDSWITCH
27   TYPE ("[ctrlright]") ' to next word
28   IF (0 = AtEOF()) ' at the end of the file yet ?
29          GOTO loop ' no; continue
30   ENDIF
31   totcount = w1cnt + w2cnt + w3cnt + w4cnt + ocnt
32   p1 = (w1cnt / totcount) * 100
33   p1 = FormatNum$ ("", "", 2, p1)
34   p2 = (w2cnt / totcount) * 100
35   p2 = FormatNum$ ("", "", 2, p2)
36   p3 = (w3cnt / totcount) * 100
37   p3 = FormatNum$ ("", "", 2, p3)
38   p4 = (w4cnt / totcount) * 100
39   p4 = FormatNum$ ("", "", 2, p4)
```

```
40   message ("There are {w1cnt} cases of {word1}, or {p1}%")
41   message ("There are {w2cnt} cases of {word2}, or {p2}%")
42   message ("There are {w3cnt} cases of {word3}, or {p3}%")
43   message ("There are {w4cnt} cases of {word4}, or {p4}%")
44   message ("Total words in document: {totcount}")
45   END FUNCTION
```

Lines 2 - 5 of this macro ask the macro user for the four words to be counted using the Query$ function and assign values to the variables word1 to word4. Line 6 uses the macro function DraftMode to put Ami Pro in Draft Mode, which speeds up processing of the macro. Lines 8 through 12 define the variables used for counters and initializes them to 0. After these setup statements, the body of the macro loop begins with the macro function getword in line 14, which gets the current word in the document, and places its value in the variable word.

Line 15 of the macro is the SWITCH statement. It sets the value of the variable the following CASE statements use to compare to the value of word. The CASE statement in line 16 compares the value of word to word1, the first word whose frequency we are trying to count. If there is a match, then the counter for the first word is incremented, and control jumps to line 27 following the ENDSWITCH statement. If not, control passes to the following line where the value of word is compared to the second word we are counting, and so forth. If a match is not found by line 24, the default case, the program increments the counter of other, non-matched words in the document. The ENDSWITCH statement in line 26 indicates the end of the SWITCH statement.

Line 27 advances the insertion point to the next word and lines 28 - 30 jump back to line 13. The process is repeated for the following word in the file until the entire file has been examined. Line 31 adds the count of each word type together to determine the total number of words in the document. Lines 32 - 39 calculate the average usage for each of the words, and use the FormatNum$ macro function to format the numbers as percentages. Finally, lines 40 - 44 display the result of the word frequency count to the user.

See also:

Ami Pro Macro Language Contents
Overview of the Ami Pro Macro Language
Macro Programming Statements

The WHILE statement repeatedly executes a series of statements as long as a condition evaluates to TRUE. The WHILE statement is useful if you want to ensure that some condition has been fulfilled before continuing with other steps in the macro.

The syntax of the WHILE statement is:

WHILE (Condition)

    Statements

WEND

Condition is a condition that evaluates to either TRUE or FALSE.

Statements are macro program statements.

When the WHILE statement is encountered in a macro program, the condition specified in condition is evaluated. If the condition evaluates to TRUE, then the statements between the WHILE and WEND statements are executed. If the condition is false, macro execution continues beyond the WEND statement. Following execution of the statements, the program goes back to the WHILE statement and reevaluates the condition. As long as the condition remains TRUE, the statements through the WEND statement continue to be executed.

The statements between a WHILE/WEND loop are not executed at all if the condition specified in the WHILE statement is FALSE when first evaluated. It is also possible to create an infinite loop in a WHILE/WEND loop if the statements between them do not affect the condition in the WHILE statement.

The following example illustrates using the WHILE loop in a macro to count the number of characters.

```
1   FUNCTION wc()
2   DraftMode() ' ensure we are in DraftMode
3   TYPE ("[ctrlhome]") ' go to top of file
4   counter = 0 ' initialize counter
5   WHILE (NOT(AtEOF())) ' as long as not at end of file
6           counter = counter +1
7           TYPE ("[right]") ' ahead a character
8   WEND
9   message ("There are {counter} characters in the file.")
10   END FUNCTION
```

Line 2 of this macro changes Ami Pro to Draft Mode for faster operation, line 3 positions the insertion point at the beginning of the file, and line 4 initializes the counter variable to 0. Line 5, the WHILE statement, tells the macro to execute the following statements as long as the insertion point is not at the end of the file. Lines 6 and 7 form the loop of the WHILE statement, incrementing the counter and advancing to the next word in the file. The WEND statement in line 8 tells the macro to go to the WHILE statement and repeat the loop. Line 9 displays the character count to the user, and line 10 ends the macro.

See also:

Ami Pro Macro Language Contents
Overview of the Ami Pro Macro Language
Macro Programming Statements

When in a FOR/NEXT loop, there are times when the loop should be ended before the total number of repetitions specified in the FOR statement. In a SWITCH/CASE loop, there are times when only certain statements should be processed for a matching case, based on some other variable. Similarly, there are times when the macro needs to break out of a WHILE/WEND loop.

The BREAK statement allows control to pass out of the loop, and ends the action of the FOR/NEXT, SWITCH/CASE, and WHILE/WEND statements.

The syntax for the Break statement is

BREAK

See also:

Ami Pro Macro Language Contents
Overview of the Ami Pro Macro Language
Macro Programming Statements
Example of the BREAK Statement

This example bolds every occurrence of the word "and".

```
1   FUNCTION BoldWord()
2   TYPE("[CTRLHome]") ' go to the beginning of the document
3   WHILE(TRUE) ' create a loop
4       findit = Replace("", "", "", "and", "") ' search for "and"
5       IF findit <> 1 ' if we can't find the word, break out of the loop
6           BREAK
7       ENDIF
8       Bold()
9       TYPE("[ESC]") ' deselect the word
10  WEND
11  END FUNCTION
```

Line 2 returns the cursor to the beginning of the document. Line 3 begins the WHILE loop that always executes because the value is TRUE. Line 4 finds the next occurrence of "and". Line 5 evaluates the variable findit with TRUE. If findit is TRUE, Line 8 bolds the word "and". Line 9 deselects the word. Line 10 ends the loop and returns control to Line 3 and the loop continues. If the variable findit is FALSE, the IF statement will evaluate Line 6 and the macro stops the loop. The macro runs the next line, in this case Line 11 is an END FUNCTION.

See also:

Ami Pro Macro Language Contents
Overview of the Ami Pro Macro Language
Macro Programming Statements

This section lists each built-in macro function by type. You can easily find the function you need by looking in the appropriate category.

Variables

Most variables are local and private to the macro in which you declare them. They are deleted when the macro completes. Ami Pro provides variables that are saved when a macro completes.

Strings

These functions manipulate strings of text. Many of these functions are similar to, and have the same syntax as, BASIC language string functions.

Ami Pro Menus

You can use these functions to create and change the Ami Pro menu bars and to add, remove, or change pulldown menus displayed from these menu bars. There is also a function to assign a macro to a function key.

Ami Pro Word Processing

These functions allow you to access the word processing features that are available in Ami Pro.

Windows Applications

These functions allow Ami Pro to use Windows Dynamic Data Exchange (DDE) to interact with other Windows programs and access Windows Dynamic Link Libraries (DLL) and the routines built into these DLL libraries. A function is also included to execute another application.

Frames

These functions give you control over the frame position, frame type, lines around, background color, and drop shadow color. With the frame control functions, you can change the frame's position, appearance, and type.

Style

These functions give you control over creating or modifying a paragraph style.

Page Layout

These functions give you control over creating or modifying the page layout.

Dialog Box

You can store dialog boxes within a macro. You can define a function or functions to run while the box is displayed. These functions can change the components of the box while the box is displayed. Bitmaps can be included as part of a dialog box. These bitmaps can be static or defined as a button. All functionality of the Windows interface can be reproduced with the Ami Pro macro dialog box.

Arrays

These functions allow you to position insert, delete, and sort arrays.

ASCII Files

These functions allow a macro to read and write ASCII text files. Reading and writing binary files is not supported. File operations on binary files may behave unpredictably.

These are low-level functions that require you to properly open and close files in use. Windows may have difficulty if a macro does not close its files.

NewWave

These functions allow you to manipulate NewWave objects. They are only available in the NewWave release of Ami Pro.

Macro Only Commands

These functions allow you to control the execution of a macro. This includes pausing and single stepping through a macro and reading from and writing to the AMIPRO.INI file.

DOS

These functions allow you to perform the most common DOS commands as macro functions.

See also:

Ami Pro Macro Language Contents

Most variables are local and private to the macro in which they are declared. They are deleted when the macro completes. Ami Pro provides variables that are saved when a macro completes.

Assign

AllocGlobalVar

FreeGlobalVar

GetGlobalVar$

GetGlobalVarCount

GetGlobalVarNames

SetGlobalVar

See also:

- Ami Pro Macro Language Contents
- Using Macro Functions Grouped by Category
- Using Global Variables to Hold Values

These functions manipulate strings of text. Many of these functions are similar to, and have the same syntax as, BASIC language string functions.

ASC
BinToBrackets
BracketsToBin
CHR$
CurChar$
CurShade$
CursorPosition$
CurWord$
DateDiff
DOSGetEnv$
fgets$
FormatDate$
FormatNum$
FormatSeq$
FormatTime$
GetAmiDirectory$
GetBackPath$
GetCurrentDir$
GetDialogField$
GetDocInfo$
GetDocInfoKeywords$
GetDocPath$
GetFmtPageStr$
GetGlobalArray$
GetGlobalVar$
GetMacPath$
GetMarkText$
GetOpenFileName$
GetProfileString$
GetRunningMacroFile$
GetRunningMacroName$
GetSpecialEffects$
GetStylePath$
GetTextBeforeCursor$
GetWindowsDirectory$
Instr
IsNumeric
LCASE$
Left$

LEN
MID$
Mod
Right$
Round
strcat$
strchr
strfield$
Truncate
TYPE
UCASE$

See also:

Ami Pro Macro Language Contents
Using Macro Functions Grouped by Category

These functions are used to create and change the Ami Pro menu bars and to add, remove, or change pulldown menus displayed from these menu bars. For a full description of each function, refer to the description for the macro function.

AddBar

AddCascadeMenu

AddCascadeMenuItem

AddMenu

AddMenuItem

AddMenuItemDDE

ChangeCascadeAction

ChangeMenuAction

CheckMenuItem

DeleteMenu

DeleteMenuItem

GrayMenuItem

InsertCascadeMenu

InsertCascadeMenuItem

InsertMenu

InsertMenuItem

RenameMenuItem

ShowBar

See also:

Ami Pro Macro Language Contents
Using Macro Functions Grouped by Category

These functions allow you to access the word processing features that are available in Ami Pro.

About
ASCIIOptions
AssignMacroToFile
Bold
CascadeWindow
Center
ChangeIcons
ChangeLanguage
ChangeShortcutKey
CharLeft
CharRight
ChartingMode
CleanScreenOptions
ClipboardRead
ClipboardWrite
ConnectCells
ControlPanel
Copy
CreateDataFile
CreateDescriptionFile
CreateStyle
CustomView
Cut
DeleteColumnRow
DeleteEntireTable
DocInfo
DocInfoFields
DocumentCompare
DraftMode
DrawingMode
EditFormula
ElevatorLeftRight
ElevatorUpDown
EndOfFile
EnhancementProducts
EnlargedView
Equations
EvalField
FacingView
FastFormat

FieldAdd
FieldAuto
FieldCommand
FieldEvaluate
FieldLock
FieldNext/FieldPrev
FieldRemove
FieldToggleDisplay
FieldUpdate
FieldUpdateAll
FileClose
FileManagement
FileOpen
FilePrint
FindReplace
FloatingHeader
FontChange
FontFaceChange
FontPointSizeChange
FontRevert
Footnotes
FormatDate$
FormatNum$
FormatSeq$
FormatTime$
FullPageView
Generate
Glossary
GlossaryAdd
GlossSet
GoToAgain
GoToCmd
GraphicsScaling
HeaderFooter
Heading
Help
HideIconBar
HideStylesBox
HideTabRuler
HowDoIHelp
IconBottom
IconCustomize
IconFloating

IconLeft
IconRight
IconTop
ImageProcessing
ImportPicture
Indent
IndentAll
IndentFirst
IndentRest
InitialCaps
InsertBullet
InsertColumnRow
InsertDate
InsertDocInfo
InsertDocInfoField
InsertLayout
InsertMerge
InsertNewObject
InsertNote
InsertVariable
Italic
Justify
KeyboardHelp
LayoutMode
LeaderDots
LeaderHyphs
LeaderLines
LeaderNone
LeftAlign
LeftEdge
LineDown
LineNumber
LineUp
LoadOptions
LowerCase
MacroEdit
MacroHelp
MacroOptions
MacroPlay
MarkBookMark
MarkIndexWord
MarkTOCEntry
MasterDoc

MasterDocOpts
Maximize
Merge
MergeAction
MergeMacro
MergeToFile
Minimize
MoveLeftOrPromote
MoveParagraphDown
MoveParagraphUp
MoveRightOrDemote
New
NewWindow
NextWindow
NoHyphenation
NormalText
Notes
OnKey
OpenDataFile
OpenMergeFile
OpenPreviousFile1
OpenPreviousFile2
OpenPreviousFile3
OpenPreviousFile4
OpenPreviousFile5
OutlineLevels
OutlineMode
OutlineStyle
PageBreak
PageDown
PageNumber
PageUp
Paste
PrintEnvelope
PrintOptions
PrintSetup
ProtectCells
ProtectedText
QuickAddCol
QuickAddRow
ReadMail
RemoveLayout
RenameDocInfoField

Replace
Restore
Revert
RevertLayout
ReviewRevisions
RevisionInsertion
RevisionMarking
RevisionMarkOpts
RightAlign
RightEdge
Save
SaveAs
SaveAsNewStyle
ScreenDown
ScreenLeft
ScreenRight
ScreenUp
SelectColumn
SelectEntireTable
SelectRow
SelectStyle
SelectWindow
SendMail
SetBackPath
SetDataFile
SetDefOptions
SetDefPaths
SetDocPath
SetFormula
SetIconPath
SetIconSize
SetIndexFile
SetMacroPath
SetMasterFiles
SetStyle
SetStylePath
SetTOCFile
ShowIconBar
ShowStylesBox
ShowTabRuler
SizeColumnRow
SmallCaps
Sort

Spacing
SpecialEffects
SpellCheck
StandardView
TableLayout
TableLines
Tables
TabRuler
TabRulerInsert
TabRulerRemove
Thesaurus
TileWindow
TOCOptions
ToggleCleanScreen
ToggleIconBar
ToggleStylesBox
ToggleTabRuler
TopOfFile
TypeOver
Underline
Undo
UpgradeHelp
UpperCase
UseAnotherStyle
UserSetup
UseWorkingDir
UsingHelp
ViewPreferences
WordUnderline

See also:

Ami Pro Macro Language Contents
Using Macro Functions Grouped by Category

These functions allow Ami Pro to use Windows Dynamic Data Exchange (DDE) to interact with other Windows programs and access Windows Dynamic Link Libraries (DLL) and the routines built into these DLL libraries. A function is also included to execute another application.

ActivateApp
AddMenuItemDDE
AppClose
AppGetAppCount
AppGetAppNames
AppGetWindowPos
AppHide
AppIsRunning
AppMaximize
AppMinimize
AppMove
AppRestore
AppSendMessage
AppSize
ControlPanel
DDEAdvise
DDEExecute
DDEInitiate
DDELinks
DDEPoke
DDEReceive$
DDETerminate
DDEUnAdvise
DLLCall
DLLFreeLib
DLLLoadLib
DLLLocate
GetProfileString$
WriteProfileString

See also:

Ami Pro Macro Language Contents
Using Macro Functions Grouped by Category

These functions give you control over the frame position, frame type, lines around, background color, and drop shadow color. With the frame control functions, you can change the frame's position, appearance, and type.

AddFrame

AddFrameDlg

BringFrameToFront

FrameLayout

FrameModBorders

FrameModColumns

FrameModFinish

FrameModInit

FrameModLines

FrameModType

GetCurFrameBorders

GetCurFrameLines

GetCurFrameType

GroupFrames

IsFrameSelected

ManualFrame

SelectFrameByName

SendFrameToBack

SetFrameDefaults

See also:

Ami Pro Macro Language Contents
Using Macro Functions Grouped by Category

These functions give you control over creating or modifying a paragraph style.

CreateStyle
DefineStyle
ModifyAlignment
ModifyBreaks
ModifyEffects
ModifyFont
ModifyLines
ModifyReflow
ModifySelect
ModifySpacing
ModifyStyle
ModifyTable
StyleManageAction
StyleManageFinish
StyleManageInit
StyleManagement

See also:

Ami Pro Macro Language Contents
Using Macro Functions Grouped by Category

These functions give you control over creating or modifying the page layout.

GetLayoutLeftLines
GetLayoutPageSize
GetLayoutParameters
GetLayoutParmCnt
GetLayoutRightLines
GetLayoutType
ModLayoutFinish
ModLayoutInit
ModLayoutLeftFooter
ModLayoutLeftHeader
ModLayoutLeftLines
ModLayoutLeftPage
ModLayoutPageSize
ModLayoutRightFooter
ModLayoutRightHeader
ModLayoutRightLines
ModLayoutRightPage

See also:

Ami Pro Macro Language Contents
Using Macro Functions Grouped by Category

You can store dialog boxes within a macro. You can define a function or functions to run while the box is displayed. These functions can change the components of the box while the box is displayed. Bitmaps can be included as part of a dialog box. These bitmaps can be static or defined as a button. You can reproduce the functions of the Windows interface using the Ami Pro dialog box.

DialogBox

DlgKeyInterrupt

FillEdit

FillList

GetDialogField$

GetDlgItem

GetDlgItemText

SetDlgCallBack

SetDlgItemText

See also:

Ami Pro Macro Language Contents
Using Macro Functions Grouped by Category

These functions allow you to position insert, delete, and sort arrays.

ArrayDelete

ArrayInsert

ArrayInsertByKey

ArraySearch

ArraySize

ArraySort

GetGlobalArray$

SetGlobalArray

See also:

Ami Pro Macro Language Contents
Using Macro Functions Grouped by Category

These functions allow a macro to read and write ASCII text files. Reading and writing binary files are not supported. File operations on binary files may behave unpredictably.

These are low-level functions that require you to open and close files properly when you use them. Windows may have difficulty if a macro does not close its files.

fclose

fgets$

fopen

fputs

fread

fseek

ftell

fwrite

See also:

Ami Pro Macro Language Contents
Using Macro Functions Grouped by Category

These functions allow you to manipulate NewWave objects. They are only available in the NewWave release of Ami Pro.

CreateANew

ImportText

IsNewWave

ListObjects

NWGetContainerCount

NWGetContainerNames

NWGetCurrentContainer

NWGetCurrentObject$

NWGetObjectCount

NWGetObjectNames

NWGetParent

NWReferenceToFile$

ObjectAttributes

OpenObject

SaveAsMaster

SaveAsObject

Share

ShowLinks

See also:

Ami Pro Macro Language Contents
Using Macro Functions Grouped by Category

These functions allow you to control the execution of a macro. This includes pausing and single stepping through a macro and reading from and writing to the AMIPRO.INI file.

AmiProIndirect
AnswerMsgBox
ApplyFormat
AtEOF
DateDiff
Decide
DECLARE
Exec
FileChanged
FindFirst$
FindNext$
GetAmiDirectory$
GetBackPath$
GetBookMarkCount
GetBookMarkNames
GetBookMarkPage
GetCurFontInfo
GetCurrentDir$
GetDocInfo$
GetDocInfoKeywords$
GetDocPath$
GetDocVar
GetFmtPageStr$
GetIconPalette
GetMacPath$
GetMarkText$
GetMasterFilesCount
GetMasterFiles
GetMode
GetOpenFileCount
GetOpenFileName$
GetOpenFileNames
GetPageNo
GetPowerFieldCount
GetPowerFieldPage
GetPowerFields
GetProfileString$
GetRunningMacroFile$
GetRunningMacroName$

GetSpecialEffects$
GetStyleCount
GetStyleName$
GetStyleNames
GetStylePath$
GetTextBeforeCursor$
GetTime
GetViewLevel
GetViewPrefLevel
GetViewPrefOpts
GetWindowsDirectory$
GoToPowerField
GoToShade
HourGlass
IgnoreKeyboard
ImportExport
IsFrameSelected
IsNumeric
KeyInterrupt
Message
Messages
MouseInterrupt
MultiDecide
Now
OnMDIActivate
Pause
PhysicalToLogical
Query$
RecClose
RecFieldCount
RecFieldName$
RecGetField
RecNextRec
RecOpen
Round
RunLater
SelectFrameByName
SendKeys
SetDocVar
SingleStep
StatusBarMsg
TableGetRange
UserControl

WriteProfileString

See also:

Ami Pro Macro Language Contents
Using Macro Functions Grouped by Category

These functions allow you to perform the most common DOS commands.

Beep
DOSchdir
DOSCopyFile
DOSDelFile
DOSGetEnv$
DOSGetFileAttr
DOSmkdir
DOSRename
DOSrmdir
DOSSetFileAttr

See also:

Ami Pro Macro Language Contents
Using Macro Functions Grouped by Category

This section contains Ami Pro functions you can use in a macro. Each function name is shown with its Ami Pro menu equivalent. For a complete list of the parameters required by the function, click on the function name. To learn how to use Ami Pro functions, refer to the Calling Ami Pro Functions section of the documentation.

Functions By Menu Name:

See also:

Ami Pro Macro Language Contents
Calling Functions With and Without Parameters

ActivateApp	System/Switch To
AppClose	System/Close
AppMove	System/Move
AppSize	System/AppSize
ControlPanel	System/Control Panel
Maximize	System/Maximize
Minimize	System/Minimize
Restore	System/Restore

See also:

Ami Pro Macro Language Contents
Using Ami Pro Functions Grouped By Menu

ASCIIOptions File/Open/ASCII Options
DocInfoFields File/Doc Info/Other Fields
DocInfo File/Doc Info
FileClose File/Close
FileManagement File/File Management
FileOpen File/Open
FilePrint File/Print
ImportPicture File/Import Picture
ImportText File/Open/Insert
MasterDoc File/Master Document
MasterDocOpts File/Master Document/Options
Merge File/Merge
New File/New
OpenPreviousFile1 File/1
OpenPreviousFile2 File/2
OpenPreviousFile3 File/3
OpenPreviousFile4 File/4
OpenPreviousFile5 File/5
PrintEnvelope File/Print Envelope
PrintOptions File/Print/Options
PrintSetup File/Printer Setup
RenameDocInfoField File/Doc Info/Other Fields/Rename Fields
Revert File/Revert to Saved
Save File/Save
SaveAs File/Save As
SendMail File/Send Mail
SetIndexFile File/Master Document/Options
SetMasterFiles File/Master Document
SetTOCFile File/Master Document/Options
TOCOptions
File/Master Document/Options/TOC Options

See also:

Ami Pro Macro Language Contents
Using Ami Pro Functions Grouped By Menu

Copy	Edit/Copy
Cut	Edit/Cut
DDELinks	Edit/Link Options
FieldAdd	Edit/Power Fields/Insert
FieldAuto	Edit/Power Fields/Insert/Auto run box
FieldCommand	Edit/Power Fields/Insert
FieldEvaluate	Edit/Power Fields/Update
FieldLock	Edit/Power Fields/Insert/Lock box
FieldNext/FieldPrev	Edit/Power Fields/Next Field or Edit/Power Fields/Prev Field
FieldUpdate	Edit/Power Fields/Update
FieldUpdateAll	Edit/Power Fields/Update All
FindReplace	Edit/Find & Replace
Glossary	Edit/Insert/Glossary Record
GlossaryAdd	Edit/Mark Text/Glossary
GlossSet	Edit/Insert/Glossary Record/Data File
GoToAgain	Edit/Go To
GoToCmd	Edit/Go To
InsertBullet	Edit/Insert/Bullet
InsertDate	Edit/Insert/Date/Time
InsertDocInfo	Edit/Insert/Doc Info Field
InsertDocInfoField	Edit/Insert/Doc Info Field/Insert (no dialog box is displayed).
InsertMerge	Edit/Insert/Merge Field
FieldAdd	Edit/Insert/Merge Field (no dialog box is displayed).
InsertNewObject	Edit/Insert/New Object
InsertNote	Edit/Insert/Note
InsertVariable	Edit/Insert/Date/Time
MarkBookMark	Edit/Bookmarks
MarkIndexWord	Edit/Mark Text/Index Entry
MarkTOCEntry	Edit/Mark Text/TOC Entry
NoHyphenation	Edit/Mark Text/No Hyphenation
Notes	Edit/Insert/Note
Paste	Edit/Paste
ProtectedText	Edit/Mark Text/Protected Text
Replace	Edit/Find & Replace
RevisionInsertion	Edit/Mark Text/Revision Insertion
Undo	Edit/Undo

See also:

Ami Pro Macro Language Contents
Using Ami Pro Functions Grouped By Menu

CleanScreenOptions View/View Preferences/Clean Screen Options
CustomView View/Custom View
DraftMode View/Draft Mode
EnlargedView View/Enlarged
FacingView View/Facing Pages
FieldToggleDisplay View/Show/Hide Power Fields
FullPageView View/Full Page
HideIconBar View/Hide SmartIcons
HideStylesBox View/Hide Styles Box
HideTabRuler View/Hide Ruler
LayoutMode View/Layout Mode
OutlineMode View/Outline Mode
ShowIconBar View/Show SmartIcons
ShowStylesBox View/Show Styles Box
ShowTabRuler View/Show Ruler
StandardView View/Standard
ToggleCleanScreen View/Show/Hide Clean Screen
ToggleIconBar View/Show/Hide SmartIcons
ToggleStylesBox View/Show/Hide Styles Box
ToggleTabRuler View/Show/Hide Ruler
ViewPreferences View/View Preferences

See also:

Ami Pro Macro Language Contents
Using Ami Pro Functions Grouped By Menu

Bold	Text/Bold
Center	Text/Alignment/Center
FastFormat	Text/Fast Format
FontChange	Text/Font
FontRevert	Text/Font/Revert to style
Indent	Text/Indention
IndentAll	Text/Indention/Indent All
IndentFirst	Text/Indention/Indent First
IndentRest	Text/Indention/Indent Rest
InitialCaps	Text/Caps/Initial Caps
Italic	Text/Italic
Justify	Text/Alignment/Justify
LeftAlign	Text/Alignment/Left
LowerCase	Text/Caps/Lower case
NormalText	Text/Normal
RightAlign	Text/Alignment/Right
SmallCaps	Text/Caps/Small Caps
Spacing	Text/Spacing
SpecialEffects	Text/Special Effects
Underline	Text/Underline
UpperCase	Text/Caps/Upper Case
WordUnderline	Text/Word Underline

See also:

Ami Pro Macro Language Contents
Using Ami Pro Functions Grouped By Menu

CreateStyle	Style/Create Style
DefineStyle	Style/Define Style
ModifyAlignment	Style/Modify Style/Alignment
ModifyBreaks	Style/Modify Style/Breaks
ModifyEffects	Style/Modify Style/Bullets & numbers
ModifyFont	Style/Modify Style/Font
ModifyLines	Style/Modify Style/Lines
ModifyReflow	Style/Modify Style
ModifySelect	Style/Modify Style/Select Paragraph Style
ModifySpacing	Style/Modify Style/Spacing
ModifyStyle	Style/Modify Styles
ModifyTable	Style/Modify Style/Table Format
OutlineStyle	Style/Outline Styles
SaveAsNewStyle	Style/Save as a Style Sheet
SelectStyle	Style/Select a Style
SetStyle	Style/Select a Style
StyleManageAction	Style/Style Management
StyleManageFinish	Style/Style Management
StyleManageInit	Style/Style Management
StyleManagement	Style/Style Management
UseAnotherStyle	Style/Use Another Style Sheet

See also:

Ami Pro Macro Language Contents
Using Ami Pro Functions Grouped By Menu

FloatingHeader Page/Header/Footer/Floating Header/Footer
HeaderFooter Page/Header/Footer
InsertLayout Page/Insert Page Layout/Insert
LineNumber Page/Line Numbering
ModifyLayout Page/Modify Page Layout
ModLayoutFinish Page/Modify Page Layout
ModLayoutInit Page/Modify Page Layout
ModLayoutLeftFooter Page/Modify Page Layout/Footer
ModLayoutLeftHeader Page/Modify Page Layout/Header
ModLayoutLeftLines Page/Modify Page Layout/Lines
ModLayoutLeftPage Page/Modify Page Layout/Left Pages
ModLayoutPageSize Page/Modify Page Layout/Page settings
ModLayoutRightFooter Page/Modify Page Layout/Footer
ModLayoutRightHeader Page/Modify Page Layout/Header
ModLayoutRightLines Page/Modify Page Layout/Lines
ModLayoutRightPage Page/Modify Page Layout/Right Pages
PageBreak Page/Breaks
PageNumber Page/Page Numbering
RemoveLayout Page/Insert Page Layout/Remove
RevertLayout Page/Insert Page Layout/Revert
TabRulerInsert Page/Ruler/Insert
TabRulerRemove Page/Ruler/Remove

See also:

Ami Pro Macro Language Contents
Using Ami Pro Functions Grouped By Menu

AddFrame	Frame/Create Frame (NO DIALOG BOX IS DISPLAYED).
AddFrameDlg	Frame/Create Frame
BringFrameToFront	Frame/Bring to Front
FrameLayout	Frame/Modify Frame Layout
FrameModBorders	Frame/Modify Frame Layout/Size & position
FrameModColumns	Frame/Modify Frame Layout/Columns & tabs
FrameModFinish	Frame/Modify Frame Layout
FrameModInit	Frame/Modify Frame Layout
FrameModLines	Frame/Modify Frame Layout/Lines & shadows
FrameModType	Frame/Modify Frame Layout/Type
GraphicsScaling	Frame/Graphics Scaling
GroupFrames	Frame/Group
ManualFrame	Frame/Create Frame/Manual
SendFrameToBack	Frame/Send to Back
SetFrameDefaults	Frame/Modify Frame Layout/Make Default

See also:

Ami Pro Macro Language Contents
Using Ami Pro Functions Grouped By Menu

AssignMacroToFile Tools/Macros/Edit/Assign
ChangeIcons Tools/SmartIcons
ChangeLanguage Tools/Spell Check/Language options
ChangeShortcutKey Tools/Macros/Edit/Change Shortcut Key
ChartingMode Tools/Charting
DocumentCompare Tools/Doc Compare
DrawingMode Tools/Drawing
Equations Tools/Equations
Footnotes Tools/Footnotes
Generate Tools/TOC, Index/Generate
IconBottom Tools/SmartIcons/Position
IconCustomize Tools/SmartIcons
IconFloating Tools/SmartIcons/Position
IconLeft Tools/SmartIcons/Position
IconRight Tools/SmartIcons/Position
IconTop Tools/SmartIcons/Position
ImageProcessing Tools/Image Processing
LoadOptions Tools/User Setup/Load
MacroEdit Tools/Macros/Edit
MacroOptions Tools/Macros/Record/Options
MacroPlay Tools/Macros/Playback
OnKey Tools/Macros/Edit
ReviewRevisions Tools/Revision Marking/Review Rev
RevisionMarking Tools/Revision Marking
RevisionMarkOpts Tools/Revision Marking/Options
SetBackPath Tools/User Setup/Paths
SetDefOptions Tools/User Setup/Options
SetDefPaths Tools/User Setup/Paths
SetDocPath Tools/User Setup/Paths
SetIconPath Tools/User Setup/Paths
SetIconSize Tools/SmartIcons/Icon Size
SetMacroPath Tools/User Setup/Paths
SetStylePath Tools/User Setup/Paths
Sort Tools/Sort
SpellCheck Tools/Spell Check
Tables Tools/Tables
Thesaurus Tools/Thesaurus
TOCOptions Tools/TOC, Index/TOC Options
UserSetup Tools/User Setup
UseWorkingDir Tools/User Setup/Paths

See also:

Ami Pro Macro Language Contents
Using Ami Pro Functions Grouped By Menu

ConnectCells	Table/Connect Cells
DeleteColumnRow	Table/Delete Column/Row
DeleteEntireTable	Table/Delete Entire Table
EditFormula	Table/Edit Formula
Heading	Table/Headings
InsertColumnRow	Table/Insert Column/Row
LeaderDots	Table/Leaders/...
LeaderHyphs	Table/Leaders/---
LeaderLines	Table/Leaders/___
LeaderNone	Table/Leaders/None
ProtectCells	Table/Protect Cells
QuickAddCol	Table/Quick Add/Column
QuickAddRow	Table/Quick Add/Row
SelectColumn	Table/Select Column
SelectEntireTable	Table/Select Entire Table
SelectRow	Table/Select Row
SetFormula	Table/Edit Formula
SizeColumnRow	Table/Column/Row Size
TableLayout	Table/Modify Table Layout
TableLines	Table/Lines & Color

See also:

Ami Pro Macro Language Contents
Using Ami Pro Functions Grouped By Menu

CascadeWindow	Window/Cascade
NewWindow	Window/New Window
TileWindow	Window/Tile

See also:

Ami Pro Macro Language Contents
Using Ami Pro Functions Grouped By Menu

About Help/About Ami Pro
EnhancementProducts Help/Enhancement Products
Help Help/Contents
HowDoIHelp Help/How Do I?
KeyboardHelp Help/Keyboard
MacroHelp Help/Macro Doc
UpgradeHelp Help/For Upgraders
UsingHelp Help/Using Help

See also:

Ami Pro Macro Language Contents
Using Ami Pro Functions Grouped By Menu

ChangeIcons Equivalent to selecting the SmartIcons symbol on the status bar and selecting the desired set.

CharLeft Equivalent to clicking on the right arrow key to the right of the scroll bar.

CharRight Equivalent to clicking on the left arrow key to the left of the scroll bar.

ElevatorLeftRight Equivalent to dragging the horizontal scroll box (elevator) to a new position on the horizontal scroll bar. Does not reposition insertion point.

ElevatorUpDown Equivalent to dragging the vertical scroll box (elevator) to a new position on the vertical scroll bar. Does not reposition insertion point.

EndOfFile Equivalent to dragging the vertical scroll box to the bottom of the scroll bar.

FontFaceChange Equivalent to selecting a font on the status bar.

FontPointSizeChange Equivalent to selecting a point size on the status bar.

LeftEdge Equivalent to dragging the horizontal scroll box to the left side of the scroll bar.

LineDown Equivalent to clicking on the up arrow key at the top of the scroll bar.

LineUp Equivalent to clicking on the down arrow key at the bottom of the scroll bar.

NextWindow Equivalent to pressing CTRL+TAB.

OutlineLevels Equivalent to clicking on the icon for the number of outline levels while in Outline Mode.

PageDown Equivalent to clicking on the PageDown icon on the status bar.

PageUp Equivalent to clicking on the PageUp icon on the status bar.

ReadMail Equivalent to clicking on the Mail button on the status bar.

RightEdge Equivalent to dragging the horizontal scroll box to the right side of the scroll bar.

ScreenDown Equivalent to clicking above the vertical scroll box on the vertical scroll bar. Scrolls down one screen.

ScreenLeft Equivalent to clicking on the right half of the horizontal scroll bar.

ScreenRight Equivalent to clicking on the left half of the horizontal scroll bar.

ScreenUp Equivalent to clicking below the vertical scroll box on the vertical scroll bar. Scrolls up one screen.

SetStyle Equivalent to selecting a paragraph style name from the status bar or the Styles Box.

TabRuler Equivalent to changing the margins, indention, tabs, and columns on the tab ruler.

TopOfFile Equivalent to dragging the vertical scroll box to the top of the scroll bar.

TypeOver Insert Key (toggles insert/typeover mode)

See also:

Ami Pro Macro Language Contents
Using Ami Pro Functions Grouped By Menu

This section of the documentation describes error messages that can appear when compiling or running macros. It also discusses some of the techniques you can use to determine the cause of the errors, and how you can test your macros to make sure they do what you want them to.

Error Messages When Macros Are Compiled

When an error occurs during macro compilation, the insertion point stops at the location of the error, and the appropriate message displays. Error messages that only contain an error number are out of memory error messages. The macro is not saved. You can determine the cause of the error, make the change, and recompile the macro. Compilation stops at the first error found, so you may need to compile the macro several times in order to find all the errors.

Error Messages While Macros Are Running

Runtime error handling is dependent on whether the SingleStep function has been used in the compiled macro. If a SingleStep function has been used, a runtime error displays a message box with the error message as the title of the message box and the offending line inside the dialog box. When the user acknowledges the error, the macro is canceled.

If no SingleStep statement has been used, the error message displays on the screen without the line information. When the user acknowledges the error, the macro is canceled.

A run-time error, whether a macro error, or an Ami Pro or Windows error, stops macro execution unless an ONERROR statement has been included in the macro. If an ONERROR statement has been included, control transfers to the location specified by that statement.

Determining the Cause of Runtime Errors

Runtime errors can be difficult to trace and debug. If other people will be using your macros, you should test them and make sure they work correctly before distributing them. Ami Pro provides several tools you can use to determine the cause of errors and trace the progress of your macro. These tools are:

SingleStep and IgnoreKeyboard
Message
Messages

See also:

Using the SingleStep and IgnoreKeyboard Functions
Using the Message Function
Using the Messages Function

The SingleStep function is the most powerful tool you can use to help debug your macros. If you use the SingleStep function and an error occurs, you will see information about the line that caused the problem. If the SingleStep function is not used, all that is shown is the error message with no indication of where the problem is.

The SingleStep function is also useful because it allows you to trace the progress of your macro. When you use control statements in your macro, it may do something you don't expect if the value tested by the control statement is something you hadn't planned on. In single step mode, each statement is displayed as it is executed so that you can tell what your macro is doing. You can verify that the tests placed in your control statements do what you planned.

Any variables defined before the SingleStep function will not be available for viewing or changing. Also, any variable not yet defined when the macro is interrupted will not be available. When you use the StepThrough feature of SingleStep and the next statement is a Call function, all statements in the macro and any macros called will be executed without interruption. Once the called macro ends, the macro will again be interrupted. If the next statement is not a macro call, then StepThrough acts just like SingleStep. If a SingleStep(0) is encountered while stepping through the macro, it will not have an effect on the macro. If a SingleStep(1) is encountered, the macro will be interrupted.

Other tools that can help you debug your macros are the IgnoreKeyboard, Message, and Messages functions. The IgnoreKeyboard function determines if any key will interrupt a macro or if only the Escape key will interrupt the macro. The Message function displays a message on the screen and then waits for the user to acknowledge it. The Messages function determines if messages issued by Ami Pro will be shown to the user or if they will be accepted and the default action taken.

If the SingleStep function has not been used in your macro, the macro user can press a key to interrupt macro play. When a key is pressed, a message appears asking the user if macro play should be continued or canceled. The IgnoreKeyboard function determines if pressing any key pauses the macro (the default) or if pressing only the Escape key pauses the macro.

If you use the SingleStep function, pausing the macro by pressing a key will cause the macro to enter single step mode. You will be able to continue macro play normally, continue in single step mode, or cancel macro play completely. This allows you to see what is happening in your macro if it appears to be in an infinite loop.

The following macro illustrates using the SingleStep Function to determine the cause of an infinite loop:

```
FUNCTION debug
SingleStep (Off)   ' allow debugging
var1 = 1   ' initialize var1
var2 = 2   ' initialize var2
while (var1 < 10)   ' set loop to continue until var1 = 10
        IF (var2 > 0)
                var1 = var1 - 1   ' decrement var1 if var2 > 0
        ENDIF
        var1 = var1 + 1 ' increment var2
WEND
END FUNCTION
```

In this example, a while loop is used to execute program steps while var1 is less than 10. The last statement in the WHILE loop increments var1, so the macro should execute the loop 10 times and then end. The IF statement in line 6 decrements var1 if var2 is greater than 0. Since var2 is equal to 2, this means that var1 will always be decremented. This means that var1 will never increase as it passes through the loop, since it will be decremented in line 7 and then incremented in line 9. This means the loop will never end.

If this macro were run, Ami Pro would appear to freeze with the hourglass icon displayed. By using SingleStep mode, pressing a key while the macro was running would allow the programmer to view the

progress of the loop and determine the problem.

See also:

- Using the Message Function
- Using the Messages Function

The Message function can also be used to display the value of a variable as your macro executes. This can assist you in determining if variable values stay within the range you expect. Consider the following example:

```
FUNCTION testswitch
choice = Query$ ("Type a number between 1 and 3")
SWITCH choice
    CASE ("1")
        ' statements for case 1
    CASE ("2")
        ' statements for case 2
    CASE ("3")
        ' statements for case 3
    DEFAULT
        message ("Got back {choice} for user's choice")
ENDSWITCH
END FUNCTION
```

This example uses a SWITCH statement to execute some macro statements based on a value typed in by the user. The Message statement in line 11 displays a message indicating the value of the switch variable if it wasn't one of the values expected. While this example is simple, placing messages in your macros can be helpful when debugging. This can be particularly true if the value being tested in a SWITCH statement is determined by the macro or is the result of a function.

See also:

Using the SingleStep and IgnoreKeyboard Functions
Using the Messages Function

The Messages function determines the action taken by a macro in response to unexpected Ami Pro messages. For instance, when a new file is opened, Ami Pro asks if the current file's changes should be saved. By using the Messages function, the macro can automatically take the default action for these messages without interrupting the macro user.

In the early stages of developing a complicated macro, you may want to leave message display on. This helps you determine that the functions in the macro do what you want them to. Once you are sure that the macro is working correctly, you can turn messages off.

See also:

- Using the SingleStep and IgnoreKeyboard Functions
- Using the Message Function

The macro has been interrupted

This message appears if the SingleStep function has been used and the macro is running in single step mode. It also appears if the SingleStep function has been used and the macro user presses a key.

The statement being executed appears in the dialog box along with three choices for continuing the macro: Single-Step, Continue, and Cancel. Choose Single-Step and the next statement is executed and you are again asked whether to continue single step mode. Choose Continue and the macro continues in normal run mode. Choose Cancel and macro execution is canceled.

Internal Runtime Error *Number*

This is an internal error. It signifies an unexpected runtime error. It should not appear. If this message does appear, note the number that is given and then call Ami Pro customer support for further assistance.

Do you want to cancel the macro?

This message appears when you did not use SingleStep mode and you pressed a key during macro execution.

If you choose Yes and the label defined by the ONCANCEL statement was used, the control transfers to the label. If the label was not used, the macro ends.

If you choose no, macro execution continues.

Error 1: Incorrect format used for this function

The format used for this function is not the correct format. Check for missing parentheses after the function name or at the end of the function. Also check for improper characters within the parentheses.

Error 2: Symbol *Name* not recognized

The character at the insertion point is not recognized by the macro compiler. If it is a variable, check to make sure it has been defined first. If it is a function, check the spelling of the function name.

Error 3: Memory not available

There is not enough memory available to compile the macro. Reduce memory usage by exiting other applications and then recompile the macro.

Error 4: *Name* is already defined as an array

The array name used in this DIM statement is already defined as something else in this macro. Use another array name.

Error 5: Array name must be followed by)

When using an array name, the element number of the array must be surrounded by parentheses. Check the parentheses and then recompile the macro.

Error 6: Incorrect expression as array argument

The element number of this array is not a number or the expression you used doesn't evaluate to an integer. Correct the expression and then recompile the macro.

Error 7: Incorrect expression, expecting)

The number of closing parentheses in this expression does not match the number of opening parentheses. Count the parentheses, correct errors, and then recompile the macro.

Error 8: Incorrect parameter count

You did not provide the correct number of parameters for this function. Check the requirements of this function, correct errors, and then recompile the macro.

Error 9: Misplaced keyword *KeyWordName*

The word *KeyWordName* does not belong at this point in the macro. Check the syntax of what you were trying to do, correct errors, and then recompile the macro.

Error 10: Token file is unreadable

This indicates the file MACTOKEN.SAM in the AMIPRO directory is damaged or missing. You cannot compile the macro until it has been replaced. Call Ami Pro support for instructions on how to replace this file.

Error 11: FUNCTION argument value is already defined

The arguments you used in the FUNCTION statement have already been defined and cannot be used at that point. You may have used a function name or a constant as the macro argument. Check the syntax, correct errors, and then recompile the macro.

Error 12: FUNCTION statement has incorrect format

The format of the FUNCTION statement is incorrect. The syntax of the FUNCTION statement is *FUNCTION macroname([arguments]).* Macroname cannot be a name of a statement or a function. Make sure that you have parentheses following the macro name and that the name is not a reserved word. Correct errors and then recompile the macro.

Error 13: DECLARE statement has incorrect format

The format of the DECLARE statement is not correct. The syntax of the DECLARE statement is *DECLARE macroname([arguments])*. Make sure that you have parentheses following the macro name and that you have not used function names for the arguments. Correct errors and then recompile the macro.

Error 14: DECLARE or CALL statement has incorrect format

The format of the DECLARE or CALL statement is not correct. The syntax of the DECLARE statement is *DECLARE macroname([arguments]).* The format of the CALL statement is *CALL macroname([arguments]).* Make sure that you have parentheses following the macro name and that you have not used function names for the arguments. Correct errors and then recompile the macro.

Error 15: Variable name *VariableName* is not appropriate here

A variable name does not belong at this point in the macro. The macro may be expecting an array name instead of a variable name. You may have incorrectly used a variable name instead of a function or statement name. Correct errors and then recompile the macro.

Error 16: Symbol *String* is not recognized

The character at the insertion point is not recognized by the macro compiler. If it is a variable, check to make sure it has been properly defined. If it is a function, check the spelling of the function name. Correct errors and then recompile the macro.

Error 17: END FUNCTION statement was not found.

The macro does not have an END FUNCTION statement. Insert an END FUNCTION statement and then recompile the macro.

Error 18: Expecting (instead of *Statement* here

The IF statement should be followed by an open parentheses and the condition to evaluate the IF statement against. Correct the IF statement and then recompile the macro.

Error 19: Expecting ENDIF instead of *Statement* here

You cannot use an ELSEIF or ELSE statement following an ELSE statement. Rearrange the order of your ELSE and ELSEIF statements and then recompile the macro.

Error 20: Expecting *Statement* instead of ELSEIF here

The location of the ELSEIF statement is not correct or the macro is not being controlled by an IFUSING_THE_IF_THEN_STATEMENTS statement at this point. The ELSEIF condition must be prior to the ELSE and ENDIF statements governed by the IF statement. Make sure the conditions are positioned correctly and then recompile the macro.

Error 21: Expecting *Statement* instead of ELSE here

The location of the ELSE statement is not correct, or the macro is not being controlled by an IFUSING_THE_IF_THEN_STATEMENTS statement at this time. The ELSE condition must be the last condition before the ENDIF statement. Make sure the conditions are positioned correctly and then recompile the macro.

Error 22: Expecting *Statement* instead of ENDIF here

The location of the ENDIF statement is not correct, or the macro is not being controlled by an IF statement at this time. The ENDIF statement should follow all ELSEIF statements. If the ELSE statement is used, the ENDIF statement should also follow it. Make sure the location of the ENDIF statement is appropriate and then recompile the macro.

Error 23: Expecting *Statement* instead of NEXT here

The location of the NEXT statement is not correct, or the macro is not being controlled by an FOR statement at this time. Make sure the location of the NEXT statement is appropriate and then recompile the macro.

Error 24: WHILE statement has incorrect format

The format of the WHILE is not correct. The syntax of the WHILE statement is *WHILE (expression) statements WEND.* Make sure that the WHILE statement is formatted correctly and then recompile the macro.

Error 25: CASE statement must follow SWITCH statement

The location of the CASE statement is not correct. All CASE statements must appear between a SWITCH and an ENDSWITCH statement. They cannot appear following the DEFAULT statement. Make sure the SWITCH loop is formatted correctly and then recompile the macro.

Error 26: Variable name must follow SWITCH statement

The SWITCH statement must have a variable name immediately following it that can be evaluated against the expressions used in the following CASE statements. Insert a variable name following the SWITCH statement and then recompile the macro.

Error 27: Expecting *Statement* instead of CASE here

The location of the CASE statement is not correct. All CASE statements must appear between a SWITCH and an ENDSWITCH statement. They cannot appear following the DEFAULT statement. Make sure the SWITCH loop is formatted correctly and then recompile the macro.

Error 28: Expecting *Statement* instead of DEFAULT here

The location of the DEFAULT statement is not correct, or the macro is not evaluating a SWITCH/ENDSWITCH condition at the location of the DEFAULT statement. The DEFAULT statement must appear following all CASE statements and just before the ENDSWITCH statement. Check to make sure the SWITCH loop is formatted correctly and then recompile the macro.

Error 29: Expecting *Statement* instead of ENDSWITCH here

The location of the ENDSWITCH statement is not correct, or the macro is not evaluating a SWITCH/ENDSWITCH condition at the location of the ENDSWITCH statement. The ENDSWITCH statement must appear following all CASE statements and the DEFAULT statement. Make sure the SWITCH loop is formatted correctly and then recompile the macro.

Error 30: BREAK statement is inappropriate here

The location of the BREAK statement is not correct, or the macro is not evaluating a SWITCH/ENDSWITCH, FOR/NEXT or WHILE/WEND condition at the location of the BREAK statement. Make sure the BREAK statement is in a FOR/NEXT, SWITCH/ENDSWITCH or WHILE/WEND loop and then recompile the macro.

Error 31: Expecting *Statement* instead of FUNCTION here

The location of the FUNCTION statement is not correct. The FUNCTION statement must be the very first thing in a macro. If this is not the first macro in the file, make sure the previous macro is terminated with an END FUNCTION statement. Make corrections and then recompile the macro.

Error 32: Expecting *Statement* instead of END FUNCTION here

The location of the END FUNCTION statement is not correct. The END FUNCTION statement must be the last statement in a macro. Prior to the END FUNCTION statement, all IF/THEN, WHILE/WEND, SWITCH/ENDSWITCH and FOR/NEXT loops must be terminated. Correct errors and then recompile the macro.

Error 33: Incorrect format used for CALL statement

The syntax you used for the CALL statement is not correct. The correct syntax is: *CALL macroname([parameters])*. Make sure that the macroname is followed by parentheses and that the arguments to the called macro are correct. Correct errors and then recompile the macro.

Error 34: Label *Name* is not defined

You referenced the label *name* in a GOTO statement, and the label was not found in the macro. Check the spelling of the label and make sure that the label is formatted correctly. Correct errors and then recompile the macro.

Error 35: *LabelName* is already defined

The label name you used is already used as a label, or it is used as a variable name or function name. Select a unique name for the label and then recompile the macro.

Error 36: Incorrect number of parameters: function

There are too few or too many parameters used for this function. Check the required syntax, correct the parameters, and then recompile the macro.

Error 37: Incorrect format used for DIM statement

The syntax you used in the DIM statement is not correct. The correct syntax is *DIM arrayname(count), [arrayname(count)...]*. Check the statement, correct errors, and then recompile the macro.

Error 38: Incorrect format used for DEFSTR statement

The syntax you used in the DEFSTR statement is not correct. The correct syntax is *DEFSTR variablename [, variablename...]*. Check the statement, correct the errors, and then recompile the macro.

Error 39: Incorrect array expression

Something is wrong with the array reference you used. The array name must be followed by the element number in parentheses. The element number must be a number or the expression must evaluate to an integer. Correct the expression and then recompile the macro.

Error 40: Assignment expected after FOR statement

You formatted the FOR statement incorrectly. The correct syntax for the FOR statement is *FOR assignment TO expression [STEP expression]*. Make sure you initialize the counting variable in the FOR loop and then recompile the macro.

Error 41: TO expected in FOR statement

You formatted the FOR statement incorrectly. The correct syntax for the FOR statement is *FOR assignment TO expression [STEP expression]*. Make sure you initialize the counting variable in the FOR loop and then recompile the macro.

Error 42: Improper label used in GOTO statement

The label name you used in the GOTO statement is not a label. Labels cannot be variable names, function names, or expressions. Make sure that the label name is correct and then recompile the macro.

Error 43: Incorrect syntax used in DECLARE statement

The DECLARE statement is formatted incorrectly. The correct syntax for the DECLARE statement is *DECLARE macroname([arguments])*. Make sure the macroname is followed by parentheses and any arguments. Correct errors and then recompile the macro.

Error 44: No macros found in file

The macro file you saved does not have any macros in it. If the file is an Ami Pro document file, you should save it with a .SAM extension. If this file is a macro file, make sure that the FUNCTION statement is spelled and formatted correctly. Correct errors and then recompile the macro.

Error 45: Incorrect character *Character* used here

The character you typed here does not belong at this point in the macro. Check the syntax for the function you were trying to perform. If the character should be part of a string, make sure that there are quotation marks around it. Correct the statement and then recompile the macro.

Error 46: Macro line can't end like this

The line the insertion point is on cannot end the way it does. Perhaps there are too few closing parentheses on the line, or perhaps you forgot to end a string with quotation marks. Check the statement, make corrections, and then recompile the macro.

Error 47: Token file has become damaged

This indicates the file MACTOKEN.SAM in the AMIPRO directory has become damaged or is missing. You cannot compile the macro until it has been replaced. Call Ami Pro support for instructions on how to replace this file.

Error 48: *String* is an improper keyname

The keyname *string* is not the name of a key. Either the keyname is misspelled or you used a square brace ([) in this string. Check the list of permissible keynames. If you wanted to type a square brace in the string, use two braces ([[) instead. Correct errors and then recompile the macro.

Error 49: Too many characters in this string

There is a limit of 80 characters that can be used in any single string. You have exceeded this limit. Shorten the string or break it into two strings and then recompile the macro.

Error 50: Called macro must have a filename or be in the current macro file

One of the macros in this file calls another macro that is not in this file and does not have the name of another macro file associated with it. Any called macros must either be in the macro file you are executing or must have filenames supplied with the macro so they can be found. Make sure the macro name is spelled correctly and its location is identified correctly. Correct errors and then recompile the macro.

Error 51: Macro not found

One of the macros in this file should be in another macro file, but is not. When a filename for an macro is specified, Ami Pro searches the macpath directory, the documents directory, and the Ami Pro program directory for the specified file. If a full path to the macro is specified, Ami Pro only checks that path. Check the filename and paths for the called macro, and ensure that they are correct. Make changes and recompile the macro.

See also:

Determining the Location of a Macro When it is Run

Error 52: Out of memory for variable space
There is not enough free memory available to allocate space for variables used in this macro.

Error 53: Insufficient memory to execute this macro

There is not enough free memory available to run this macro. Close other open windows and try running the macro again.

Error 54: DDEReceive$ could not be executed

This message indicates that the DDEReceive$ function was not able to execute. This could be because the DDEInitiate function was not used to set up communications, or because the DDEInitiate function failed to communicate with the other Windows application. Make sure that you correctly started the DDE conversation, and that the other application is responding before using the DDEReceive$ function. Correct errors and then recompile the macro.

Error 55: DDEReceive$ did not get data back from the other program

This message indicates that the DDEReceive$ function was able to communicate with the other program, but that no data was received from the application by the DDEReceive$ function. This could be because the other application was not initialized correctly prior to using the DDEReceive$ function. Check the parameters required by the other program, and make sure that the DDE conversation was set up correctly. Correct errors and then recompile the macro.

Error 56: Incorrect parameters used for DDEReceive$ function

This message indicates that the DDEReceive$ function was not formatted using the correct parameters. This could be because the ChannelID was not correct, or because the data requested from the other application was not presented in the format required by the other application. Check the ChannelID and the format required for DDE requests from the other application. Correct errors and then recompile the macro.

Error 57: Insufficient memory to execute this macro
Unused.

Error 58: Incorrect parameters used for DDEExecute function

This message indicates that the DDEExecute function was not able to execute because the commands used in the DDEExecute function were not correct for the other application. Check the format required for giving commands to the other application. Correct and then recompile the macro.

Error 59: Incorrect Parameters used *MacroStatement* Line *Number*

One of the parameters you used for this function is not correct for the function. Perhaps the function requires a numeric parameter and you used a string instead. Check the required parameters, make corrections, and then recompile and rerun the macro.

Error 60: Can't read macro file

This error message indicates that Ami Pro cannot read the coded instructions to execute the macro stored in the macro file. If this message should appear, you should display the macro file, and save it again to regenerate the internal instructions. If the file saves successfully, and the macro still will not run, call Ami Pro Customer support for further instructions. When calling support, make sure you have the error number displayed, and any other parameters displayed in the error message.

Error 61: Internal error, start address *String*

Error 62: Internal error, opcode *String*

Error 63: Internal error, function *Number*

Internal errors should not appear. They indicate a problem with the way the macro was saved, or with the Ami Pro logic that controls macro execution. If an internal error should appear, you should first display the macro file. Save the file again to make sure that the file has not become damaged. If the file saves successfully and the macro still will not run, call Ami Pro Customer support for further instructions. When calling support, make sure you have the error number displayed and any other parameters displayed in the error message.

Error 64: Internal error, parm count *String* Line *Number*

Error 65: Internal error, no function number *String* Line *Number*

Error 66: Internal error, bad logop

Internal errors should not appear. They indicate a problem with the way the macro was saved, or with the Ami Pro logic which controls macro execution. If an internal error should appear, you should first display the macro file. Save the file again to make sure that the file has not become damaged. If the file saves successfully, and the macro still will not run, call Ami Pro Customer support for further instructions. When calling support, make sure you have the error number displayed and any other parameters displayed in the error message.

Error 71: Attempt to add non-numeric values *MacroStatement* Line *Number*

This message appears when a macro is running, and an assignment statement or evaluation of variables attempts to add variables that are not numbers. This most frequently occurs when a variable which has a string value is added to a number. Check to make sure that the variables in the statement evaluate to numbers rather than strings. Remember that an uninitialized variable is equal to the null string (""), rather than zero. You must specifically initialize variables in order to guarantee that they will be treated as numeric.

Determine the value of the offending variable and correct its value to ensure it is numeric. Then recompile the macro and run it again.

Error 72: Attempt to divide by zero, *MacroStatement* Line *Number*

This message appears when a macro is running and a division operation attempts to divide by zero. Macro execution stops when this message appears. Determine the variable that is zero at the time of the statement execution. Perhaps you can evaluate this variable before doing the division and allow the user to change it, or skip the division if it is zero. Make corrections, recompile, and then rerun the macro.

Error 73: Attempted math function on non-numeric values, *MacroStatement* Line *Number*

This message appears when a macro is running, and an assignment statement or evaluation of variables attempts to perform binary arithmetic or multiplication and division on variables that are not numbers. This most frequently occurs when a variable that has a string value is divided or multiplied by a number. This message could also appear if a non-integral number was used as an operand in a binary operation.

Check to make sure that the variables in the statement evaluate to numbers rather than strings. Remember that an uninitialized variable is equal to the null string (""), rather than zero. You must specifically initialize variables in order to guarantee that they will be treated as numeric. Determine the value of the offending variable and correct its value to ensure it is numeric. Then recompile the macro and run it again.

Error 70: Internal error, bad mathop

Error 74: Internal error, bad mulop

Error 75: Internal error, POPVAR stack

Internal errors should not appear. They indicate a problem with the way the macro was saved, or with the Ami Pro logic which controls macro execution. If an internal error should appear, first display the macro file and then save it to make sure that the file has not become damaged. If the file saves successfully, and the macro still will not run, call Ami Pro Customer support for further instructions. When calling support, make sure you have the error number displayed and any other parameters displayed in the error message.

Error 67: Internal error, logop stack

Error 68: Internal error, relop stack

Error 69: Internal error, mathops stack

Internal errors should not appear. They indicate a problem with the way the macro was saved, or with the Ami Pro logic which controls macro execution. If an internal error should appear, first display the macro file and then save it to make sure that the file has not become damaged. If the file saves successfully, and the macro still won't run, call Ami Pro Customer support for further instructions. When calling support, make sure you have the error number displayed and any other parameters displayed in the error message.

Error 76: Internal error, JMPF stack

Error 77: Internal error, JMPF stack

Error 78: Internal error, POPRLT stack

Internal errors should not appear, and indicate a problem with the way the macro was saved, or with the Ami Pro logic which controls macro execution. If an internal error should appear, first display the macro file and then save it to make sure that the file has not become damaged. If the file saves successfully, and the macro still will not run, call Ami Pro Customer support for further instructions. When calling support, make sure you have the error number displayed and any other parameters displayed in the error message.

Error 79: Internal error, call address *String*

Internal errors should not appear. They indicate a problem with the way the macro was saved, or with the Ami Pro logic which controls macro execution. If an internal error should appear, first display the macro file and then save it to make sure that the file has not become damaged. If the file saves successfully, and the macro still will not run, call Ami Pro Customer support for further instructions. When calling support, make sure you have the error number displayed and any other parameters displayed in the error message.

Error 80: Array index out of range

The index value of an array is larger than the number of elements assigned to it using the DIM statement. This can occur when a FOR/NEXT loop or WHILE/WEND loop is used to increment the value of an array index and the element limit in the DIM statement is exceeded. Macro execution stops when this occurs. Determine the statement that is causing the index to grow too large, correct the problem or dimension the array with more elements, and then recompile and rerun the macro.

Error 81: Macro called Ami Pro Function that has been grayed

This error message will occur if an Ami Pro function is called at an inappropriate time. An example of this would be to display the tab ruler in draft mode, or to create a frame in draft mode. This message also appears when you call an Ami Pro function such as FileOpen that is not available under NewWave, or you call a NewWave specific function, such as CreateANew, when not running NewWave.

When this error appears, macro execution is canceled. Determine the Ami Pro function that could not be executed. Perhaps you can use other Ami Pro commands to guarantee that the program is in the correct operating mode to let the function run correctly, or you could jump over the function if the program is not in the correct mode. Correct the problem, recompile the macro, and then run it again.

Error 82: ONERROR or ONCANCEL can't be in a called macro

This message appears when a macro is running, and an ONERROR or ONCANCEL statement is found in a macro that is called by another macro. This message is an internal error and should not occur. If this message displays, call Ami Pro customer support.

Error 83: Macro referenced in CALL statement can't be found

The macro used a CALL statement to call another macro, but the called macro was not in the 'main' macro file or the file referenced in the CALL statement. Any called macros must either be in the macro file executed by the user, or must have filenames supplied with the macro so they can be found. Make sure the macro name is spelled correctly and that its location is identified. Correct errors and then recompile the macro.

Error 84: Insufficient memory for AddMenu functions

There is not enough free memory available to run this macro. Close other open windows to reduce memory and try running the macro again.

Error 86: Unable to complete CreateMenu function

The macro was unable to create the menu you requested. This could be because you used an invalid bar ID or because there was already a menu with the name you requested. Check the AddMenu and AddBar statements in the macro. Corect errors and then recompile the macro.

Error 87: Maximum menu count reached in AddMenu function

No more menus could be added because the maximum number of menus on the menu bar had been reached. You will need to rewrite the macro to include fewer menus, or use a different menu bar. Make corrections and then recompile the macro.

Error 89: Unable to complete menu function, Lock failure

The menu function you wanted could not be completed because of an error communicating with the Windows Menu Manager. Try closing other windows you may have open, or exit Windows and restart. If you continue to have problems, call Ami Pro customer support.

Error 90: Unable to create new menu

The macro was unable to create the menu item you requested. This could be because you used an invalid bar ID or menu ID, or because there was already a menu item with the name you requested. Check the AddMenu, AddBar, and AddMenuItem statements in the macro. Correct errors and then recompile the macro.

Error 91: Maximum count reached in AddMenuItem

No more menu items could be added to the menu because the maximum had already been reached. You will need to rewrite the macro to include more menus or fewer items. Correct errors and then recompile the macro.

Error 92: Insufficient memory for menu functions.

There is not enough free memory available to run this macro. Close other open windows to reduce memory, and try running the macro again.

Error 93: Internal Error, shrinking menus

Error 94: Internal Error, symbol number

Error 95: Internal Error, symbol number

Internal errors should not appear. They indicate a problem with the way the macro was saved, or with the Ami Pro logic that controls macro execution. If an internal error should appear, first display the macro file and then save it to make sure that the file has not become damaged. If the file saves successfully, and the macro still will not run, call Ami Pro Customer support for further instructions. When calling support, make sure you have the error number displayed and any other parameters displayed in the error message.

Error 96: Internal Error, grow strings

Internal errors should not appear. They indicate a problem with the way the macro was saved, or with the Ami Pro logic which controls macro execution. If an internal error should appear, first display the macro file and then save it to make sure that the file has not become damaged. If the file saves successfully, and the macro still will not run, call Ami Pro Customer support for further instructions. When calling support, make sure you have the error number displayed and any other parameters displayed in the error message.

Error 97: Internal Error, strings

Error 98: Internal Error, grow strings

Error 99: Internal Error, lock strings

Internal errors should not appear. They indicate a problem with the way the macro was saved, or with the Ami Pro logic which controls macro execution. If an internal error should appear, first display the macro file and then save it to make sure that the file has not become damaged. If the file saves successfully, and the macro still will not run, call Ami Pro Customer support for further instructions. When calling support, make sure you have the error number displayed and any other parameters displayed in the error message.

Error 100: Internal Error, realloc array

Error 101: Internal Error, string frame

Error 102: Internal Error, no stack

Internal errors should not appear. They indicate a problem with the way the macro was saved, or with the Ami Pro logic which controls macro execution. If an internal error should appear, first display the macro file then save it to make sure that the file has not become damaged. If the file saves successfully, and the macro still will not run, call Ami Pro Customer support for further instructions. When calling support, make sure you have the error number displayed and any other parameters displayed in the error message.

Error 103: Index out of bounds in global array

The index value of a global array is larger than the number of elements assigned to it using the AllocGlobalVar function. This can occur when a FOR/NEXT loop or WHILE/WEND loop is used to increment the value of an array index, and the limit set in the AllocGlobalVar function is exceeded. Macro execution stops when this occurs. Determine the statement thet is causing the index to grow too large or increase the size of the array in the AllocGlobalVar function. Correct errors and then recompile and rerun the macro.

Error 104: Insufficient memory to run this macro

There is not enough free memory available to run this macro. Close other open windows to reduce memory and try running the macro again.

Error 105: Unknown global variable used

The global variable ID you used has not been defined. Each global variable is assigned an ID when it is created, and this ID must be used when that variable is accessed. Check the ID to make sure it is valid. Correct errors and then recompile and rerun the macro.

Error 106: Incorrect parameter used for this function

One of the parameters you used for this function is not correct for the function. Perhaps the function requires a numeric parameter and you used a string instead. Check the required parameters. Correct errors and then recompile and rerun the macro.

Error 107: Insufficient memory to run this macro

There is not enough free memory available to run this macro. Close other open windows to reduce memory and try running the macro again.

Error 108: *Filename* is not a macro file

The macro you wanted to display or run is not an Ami Pro macro file. If it is a document file, it should have the extension .SAM. If it is a macro file, it should have the extension .SMM. Make sure the contents of the file are correct.

Error 109: Incorrect combination of keyword(s)

The line following an END FUNCTION statement does not have a FUNCTION statement, or you have used a FUNCTION statement without having ended the previous function with an END FUNCTION statement. Check the macro, make corrections, and then recompile.

Error 110: Incorrect offset for start of string

The offset you provided for a MID$ or strchr function argument is 0. The minimum possible offset is 1, which will begin the search with the first character in the string. Change the argument and then recompile the macro.

Error 111: THEN statement is inappropriate here

The location of the THEN statement is not correct. The THEN statement must follow the condition specified in the IF statement. Check the function, make corrections, and then recompile the macro.

Error 112: Record Function(s) couldn't be translated

One of the functions you are translating from a Record/Play macro to an editable macro did not translate. Some Ami Pro functions, such as charting, can be recorded and played back, but cannot be edited. You can recognize these functions in the editable macro because they will appear as a series of numbers that are commented out of the macro. If you want to keep the original macro as it was, do not save the edited version.

Error 113: Unknown token *number*, is not a valid parameter.

You used the specified item as an argument to a macro function or to another macro. The item is not a valid variable name or definition substitution, and cannot be passed to a function. Check the spelling of the variable name or definition, recompile the macro, and try again.

Error 114: Function *FunctionName* has not been declared.

You used FunctionName in your macro, and it is not a valid macro function or user defined function. Check the spelling of the built-in function, or make sure you have correctly declared your user-defined function. Recompile the macro and try again.

Error 115: Failure to lock global memory.

Ami Pro was unable to access memory needed for the macro. Try running the macro again; if it fails, then redisplay the macro, resave and run it again. If the problem continues call Ami Pro Customer Support.

Error 116: Corrupt variable.
A string variable for the macro has been corrupted. Check string manipulations for errors.

Error 117: Invalid variable type (CodeId).
Unable to obtain value of the variable due to unknown type. Converting to code:

- 2 - Integer
- 3 - Floating point
- 4 - String

Error 118: Mismatch on assignment (CodeId).
Unable to assign value to a macro variable due to a type mismatch. Converting to code:

- 2 - Integer
- 3 - Floating point
- 4 - String

Error 119: Memory allocation failure (CodeId).

Ami Pro was unable to obtain enough memory to store a macro variable. Trying closing one or more of the other Windows applications and running the application again. Also check AllocGlobalVar calls to be sure they are requesting reasonable amounts of memory.

Error 120: Wrong data type for operation (CodeId).
The variable passed to a macro function could not be converted to the correct format for use.

Error 121: GlobalLock failure (CodeId).
Internal error: Ami Pro could not access previously stored data. Call Ami Pro Customer Support.

Error 122: Variable confusion.
Unable to determine the type of variable being substituted.

Error 123: Unrecognized Field: %s.
Unused

Error 124: Field does not end properly.
Power field is missing correct termination.

Error 125: Field number out of range.
Macro attempted to reference a non existent field.

Error 126: Quoted string not properly terminated.
The string was not correctly ended.

Error 127: Call statement must reference a macroname.

Check to be sure the Call statement is not using a standard macro function name, and that the MacroName in the Call statement is an existing Ami Pro macro. See the Call statement documentation for information on syntax.

Error 129: Invalid number for FormatDate.
The Style field of the FormatDate function call is invalid. Check to be sure that it is one of the values listed in the FormatDate function documentation.

Error 130: Failed to Exec ProgramNames.
Ami Pro was unable to execute the program. Be sure that the application exists on your path.

Error 131: Indirect variable required. (Use &variable)

The variable used is a direct variable. Ami Pro needs an indirect variable that points to the intended variable. Trying placing an ampersand (&) in front of the variable and resaving the macro.

Error 132: Array variable required for this function.
An array variable is needed to run this function.

Error 133: Incorrect number of parameters.
Check the function documentation for the correct number of parameters required for the function.

Error 134: Only one macro may be paused at a time.
Use the pause function for only one macro at a time.

Error 135: Invalid DIALOG BOX format.

The format of the dialog box is not correct. See the DialogBox Call information in the macro documentation for format information.

Error 136: Misplaced Keyword DIALOG. DIALOG boxes must be defined outside of functions.

DIALOG is reserved for use in Ami Pro macro. Change the name of the variable to correct the problem. If you are trying to create a dialog box see the DialogBox function documentation.

Error 137: Misplaced Keyword END DIALOG.
END DIALOG is a reserved string in Ami Pro macros.

Error 138: Illegal use of indirection.

The ampersand character (&) is used illegally. Try deleting it and use the remaining variable name. See the Variables section for further information.

Error 139: Invalid argument to ALIAS.
The alias created in the DECLARE section for referencing another macro is not acceptable. See the DECLARE section for further information.

Error 140: Invalid argument.
Not Correct parameter or number of parameters in function call.

Error 141: Misplaced ALIAS.
Unused.

Error 142: DEFINE (Names) identifier must be undefined.

Name is a reserved word and can not be used in a DEFINE statement. Check the DEFINE statements and be sure they each have two parameters and contain no reserved words.

Error 143: DEFINE closing parenthesis expected.
The DEFINE statement includes an open parenthesis and expects a matching closing parenthesis.

Error 144: Invalid syntax for DEFINE.

DEFINE statement syntax is not correct. Check the macro documentation for further documentation for DEFINE.

Error 145: Replacement too long for DEFINE.
The replacement string, including inserted parameters must be less than 500 characters.

Error 146: Duplicate DEFINE/Variable name (Names).
The variable name is defined more than once.

Error 147: Line too long.
Macro lines must be less than 255 characters in length.

Error 148: NewWave Function called.

Macro called a NewWave function while not running the NewWave application. Note: NewWave functions are not available in the Windows version of Ami Pro.

Error 149: Open parenthesis unexpected.
Expected closing parenthesis in DEFINE statement.

Error 150: Invalid Dll ID.
The DLL referenced in a DLLCall function was not previously loaded, or the load was unsuccessful.

Error 151: Failed to load DLL.

The macro was unable to load the DLL. Check to be sure that either the DLL is in the Windows directory or that a full path is used to reference it.

Error 152: Failed to open control file.

Ami Pro could not open the macro control file, MACTOKEN.SAM. Check to be sure that it exists and that it resides in the Ami Pro directory.

This section of the macro documentation lists each of the error messages that could appear during macro compilation and execution, along with suggestions for determining the cause of the problem, and how to fix the problem. It does not cover the regular Ami Pro or Windows error messages. These are documented in the User's Guide.

Each error message is numbered. This lets you quickly identify the error, and look it up here. Messages in this section are listed in numerical order, except for the first few 'status messages'.

Italicized words in this section, such as string, number, and filename are replaced when the message is displayed on screen by the actual statement, filename, number, etc. causing the problem.

To read the suggestions for error correction, click on the error message number.

Error 1: Incorrect format used for this function
Error 2: Symbol *Name* not recognized
Error 3: Memory not available
Error 4: *Name* is already defined as an array
Error 5: Array name must be followed by)
Error 6: Incorrect expression as array argument
Error 7: Incorrect expression, expecting)
Error 8: Incorrect parameter count
Error 9: Misplaced keyword *KeyWordName*
Error 10: Token file is unreadable
Error 11: FUNCTION argument value is already defined
Error 12: FUNCTION statement has incorrect format
Error 13: DECLARE statement has incorrect format
Error 14: DECLARE or CALL statement has incorrect format
Error 15: Variable name *VariableName* is not appropriate here
Error 16: Symbol *String* is not recognized
Error 17: END FUNCTION statement was not found.
Error 18: Expecting (instead of *Statement* here
Error 19: Expecting ENDIF instead of *Statement* here
Error 20: Expecting *Statement* instead of ELSEIF here
Error 21: Expecting *Statement* instead of ELSE here
Error 22: Expecting *Statement* instead of ENDIF here
Error 23: Expecting *Statement* instead of NEXT here
Error 24: WHILE statement has incorrect format
Error 25: CASE statement must follow SWITCH statement
Error 26: Variable name must follow SWITCH statement
Error 27: Expecting *Statement* instead of CASE here
Error 28: Expecting *Statement* instead of DEFAULT here
Error 29: Expecting *Statement* instead of ENDSWITCH here
Error 30: BREAK statement is inappropriate here
Error 31: Expecting *Statement* instead of FUNCTION here
Error 32: Expecting *Statement* instead of END FUNCTION here
Error 33: Incorrect format used for CALL statement
Error 34: Label *Name* is not defined
Error 35: *LabelName* is already defined
Error 36: Incorrect number of parameters: function
Error 37: Incorrect format used for DIM statement
Error 38: Incorrect format used for DEFSTR statement
Error 39: Incorrect array expression
Error 40: Assignment expected after FOR statement

Error 41: TO expected in FOR statement
Error 42: Improper label used in GOTO statement
Error 43: Incorrect syntax used in DECLARE statement
Error 44: No macros found in file
Error 45: Incorrect character *Character* used here
Error 46: Macro line can't end like this
Error 47: Token file has become damaged
Error 48: *String* is an improper keyname
Error 49: Too many characters in this string
Error 50: Called macro must have a filename or be in the current macro file
Error 51: Macro not found
Error 52: Out of memory for variable space
Error 53: Insufficient memory to execute this macro
Error 54: DDEReceive$ could not be executed
Error 55: DDEReceive$ did not get data back from the other program
Error 56: Incorrect parameters used for DDEReceive$ function
Error 57: Insufficient memory to execute this macro
Error 58: Incorrect parameters used for DDEExecute function
Error 59: Incorrect Parameters used *MacroStatement* Line *Number*
Error 60: Can't read macro file
Error 61: Internal error, start address *String*
Error 62: Internal error, opcode *String*
Error 63: Internal error, function *Number*
Error 64: Internal error, parm count *String* Line *Number*
Error 65: Internal error, no function number *String* Line *Number*
Error 66: Internal error, bad logop
Error 67: Internal error, logop stack
Error 68: Internal error, relop stack
Error 69: Internal error, mathops stack
Error 70: Internal error, bad mathop
Error 71: Attempt to add non-numeric values *MacroStatement* Line *Number*
Error 72: Attempt to divide by zero, *MacroStatement* Line *Number*
Error 73: Attempted math function on non-numeric values, *MacroStatement* Line *Number*
Error 74: Internal error, bad mulop
Error 75: Internal error, POPVAR stack
Error 76: Internal error, JMPF stack
Error 77: Internal error, JMPF stack
Error 78: Internal error, POPRLT stack
Error 79: Internal error, call address *String*
Error 80: Array index out of range

Error 81: Macro called Ami Pro Function that has been grayed
Error 82: ONERROR or ONCANCEL can't be in a called macro
Error 83: Macro referenced in CALL statement can't be found
Error 84: Insufficient memory for AddMenu functions
Error 86: Unable to complete CreateMenu function
Error 87: Maximum menu count reached in AddMenu function
Error 89: Unable to complete menu function, Lock failure
Error 90: Unable to create new menu
Error 91: Maximum count reached in AddMenuItem
Error 92: Insufficient memory for menu functions.
Error 93: Internal Error, shrinking menus
Error 94: Internal Error, symbol number
Error 95: Internal Error, symbol number
Error 96: Internal Error, grow strings
Error 97: Internal Error, strings
Error 98: Internal Error, grow strings
Error 99: Internal Error, lock strings
Error 100: Internal Error, realloc array
Error 101: Internal Error, string frame
Error 102: Internal Error, no stack
Error 103: Index out of bounds in global array
Error 104: Insufficient memory to run this macro
Error 105: Unknown global variable used
Error 106: Incorrect parameter used for this function
Error 107: Insufficient memory to run this macro
Error 108: *Filename* is not a macro file
Error 109: Incorrect combination of keyword(s)
Error 110: Incorrect offset for start of string
Error 111: THEN statement is inappropriate here
Error 112: Record function(s) couldn't be translated
Error 113: Unknown token *number* is not a valid parameter.
Error 114: Function *FunctionName* has not been declared.
Error 115: Failure to lock global memory.
Error 116: Corrupt variable.
Error 117: Invalid variable type (%ld).
Error 118: Mismatch on assignment (%ld).
Error 119: Memory allocation failure (%ld).
Error 120: Wrong data type for operation (%ld).

Error 121: GlobalLock failure (%ld).
Error 122: Variable confusion.
Error 123: Unrecognized Field: %s.
Error 124: Field does not end properly.
Error 125: Field number out of range.
Error 126: Quoted string not properly terminated.
Error 127: Call statement must reference a macroname.
Error 129: Invalid number for FormatDate.
Error 130: Failed to Exec %s.
Error 131: Indirect variable required. (Use &variable)
Error 132: Array variable required for this function.
Error 133: Incorrect number of parameters.
Error 134: Only one macro may be paused at a time.
Error 135: Invalid DIALOG BOX format.
Error 136: Misplaced Keyword DIALOG. DIALOG boxes must be defined outside of functions.
Error 137: Misplaced Keyword END DIALOG.
Error 138: Illegal use of indirection.
Error 139: Invalid argument to ALIAS.
Error 140: Invalid argument.
Error 141: Misplaced ALIAS.
Error 142: DEFINE (%s) identifier must be undefined.
Error 143: DEFINE closing parenthesis expected.
Error 144: Invalid syntax for DEFINE.
Error 145: Replacement to long for DEFINE.
Error 146: Duplicate DEFINE/Variable name (%s).
Error 147: Line too long.
Error 148: NewWave Function called.
Error 149: Open parenthesis unexpected.
Error 150: Invalid Dll ID.
Error 151: Failed to load DLL.
Error 152: Failed to open control file.