Contents

Click an icon or a title to find information.

Macro Command Index Product and programming commands

Concepts Information about macros, dialog boxes, coaches, macro examples, and Macro Facility

Search Index of help topics and features

Additional Help Glossary, Print Topics, GroupWise Help file

Macro Commands Index

A
B
C
D
E
F
G
H
I
J
K
L
M
N
O
P
Q
R
S
T
U
V
W
X
Y
Z
Print
Close

This index contains all the product and programming macro commands. Programming commands are in uppercase, and product commands are in mixed case. The Programming Commands Index contains only programming commands. For more information on using programming and product commands, see Macro Basics and Creating Macros.

// (Comment)

A

AboutDlg

AddressBookCompare
AddressBookDeletePersonalGroup
AddressBookDlg
AddressBookGetEntry
AddressBookGetField
AddressBookGetFullName
AddressBookIsGroupMember
AddressBookResolve
AddressBookResolveFullName
AddressBookSearch
AddressBookSearchBegin
AddressBookSearchEnd
AddressBookSearchNext
AddressBookSearchPrevious
AddressInfoDlg
AddressListAdd
AddressListCreate
AddressListCreateFromGroup
AddressListDelete
AddressListEdit
AddressListGetAddress
AddressListGetCount
AddressListGetEntry
AddressListGetField
AddressListGetFullName
AddressListIsInSection
AddressListSaveAsPersonalGroup
AlarmSet
AlarmSetDlg
APPACTIVATE
AppAllowClose
APPEXECUTE
APPEXECUTEEXT
APPLICATION
APPLOCATE
ASSERT
ASSIGN
AttachmentAdd
AttachmentAddDlg
AttachmentDelete
AttachmentListDlg
AttachmentOpen
AttachmentSaveAs
AttachmentSaveAsDlg
AttachmentView

B

BEEP
BIFFILEPATH
BIFINFO
BIFREAD
BIFWRITE
BREAK

BusySearch
BusySearchDlg
ButtonBarCopy
ButtonBarOptions
ButtonBarOptionsDlg
ButtonBarPreferences
ButtonBarSave
ButtonBarSelect
ButtonBarSelectDlg
ButtonBarSetFont
ButtonBarSetLocation
ButtonBarSetRows
ButtonBarSetStyle
ButtonBarShow

C

CALL
CANCEL
CASE CALL
CASE
CHAIN
CHARLEN
CHARPOS
CheckboxSelect
CLOSEFILE
CloseWindow
COACHANIMATE
COACHFILTERADD
COACHFILTERDESTROY
COACHFILTERDISABLE
COACHFILTERENABLE
COACHGETREGIONINFO
COACHMESSAGEBOX
COACHMESSAGEBOXCLOSE
COACHPROMPT
COACHPROMPTCLOSE
COACHREMOVEDIALOGFILTER
COACHSETDIALOGFILTER
COPYFILE
CREATEDIRECTORY
CTON
CustomCommandDelete
CustomCommandExecute
CustomCommandInstall
CustomMessageCompose
CustomMessageDelete
CustomMessageInstall

D

DateAbsoluteGoTo
DateDifferenceDlg

DateParse
DateParseDay
DateParseDayOfWeek
DateParseDurationHours
DateParseDurationMinutes
DateParseHours
DateParseMinutes
DateParseMonth
DateParseYear
DateRelativeGoTo
DateRelativeGoToDlg
DateSetAutodateMode
DateSetStartDateMode
DateSwitchToAutoDate
DateSwitchToDuration
DateSwitchToEndDate
DateSwitchToStartDate
DayColumnAdd
DayColumnDelete
DBToggle
DBUseArchive
DBUsePrimary
DDEEXECUTE
DDEEXECUTEEXT
DDEINITIATE
DDEPOKE
DDEREQUEST
DDETERMINATE
DDETERMINATEALL
DECLARE
DEFAULTUNITS
Delete
DeleteCharPrevious
DELETEDIRECTORY
DELETEFILE
DeleteWord
DeleteWordLeft
DeleteWordRight
DIALOGADDCHECKBOX
DIALOGADDCOLORWHEEL
DIALOGADDCOMBOBOX
DIALOGADDCONTROL
DIALOGADDCOUNTER
DIALOGADDEDITBOX
DIALOGADDFILENAMEBOX
DIALOGADDFRAME
DIALOGADDGROUPBOX
DIALOGADDHLINE
DIALOGADDHOTSPOT
DIALOGADDICON
DIALOGADDLISTBOX
DIALOGADDLISTITEM
DIALOGADDPOPUPBUTTON
DIALOGADDPUSHBUTTON
DIALOGADDRADIOBUTTON

DIALOGADDSCROLLBAR
DIALOGADDTEXT
DIALOGADDVIEWER
DIALOGADDVLINE
DIALOGDEFINE
DIALOGDESTROY
DIALOGDISMISS
DIALOGDISPLAY
DIALOGHANDLE
DIALOGSHOW
DIALOGUNDISPLAY
DIMENSIONS
DISCARD
DLLCALL
DLLCALL PROTOTYPE
DLLFREE
DLLLOAD
DOESDIRECTORYEXIST
DOESFILEEXIST

E

EditCopy
EditCut
EditPaste
ELSE
EmptySelectedItems
EmptyTrash
ENDAPP
ENDFOR
ENDFUNC
ENDFUNCTION
ENDIF
ENDIFPLATFORM
ENDPROC
ENDPROCEDURE
ENDSWITCH
ENDWHILE
Enter
EnumSelect
EnvCheckCurrentWindow
EnvClipboardText
EnvCommandLine
EnvCurrentViewName
EnvDefaultConnectionName
EnvIsCommandValid
EnvIsMacroPlaying
EnvIsMacroRecording
EnvIsNetworkUser
EnvIsRemote
EnvIsRemoteConnectionActive
EnvIsRemoteConnectionNeeded
EnvIsRemoteConnectionFinished
EnvLastCmdResult

EnvLastError
EnvLastWindowCanceled
EnvNetworkLoginID
EnvPrefArchivePath
EnvPrefCustomViewPath
EnvPrefFullName
EnvPrefMacroPath
EnvPrefPostOfficePath
EnvPrefSavePath
EnvPrivateBif
EnvPublicBif
EnvSharedDLLPath
EnvTextCurrentCharIndex
EnvTextCurrentLineIndex
EnvTextCurrentWord
EnvTextInsertMode
EnvTextLeftChar
EnvTextLineCharCount
EnvTextLineCount
EnvTextRightChar
EnvUserID
EnvVersionDate
EnvVersionName
ERROR
EventNotify
EXISTS

F

FILEERROR
FILEFIND
FILEISEOF
FILEPOSITION
FILEREAD
FILESIZE
FILETRUNCATE
FILEWRITE
FilterApply
FilterCreate
FilterDelete
FilterDlg
FilterExecute
FilterFromFile
FilterSetAttachmentClass
FilterSetAttribute
FilterSetDateAbsolute
FilterSetDateRelative
FilterSetItemType
FilterSetPriority
FilterSetSource
FilterSetText
Find
FindDlg
FindNext

FindPrevious
FocusSet
FolderAddSelections
FolderContract
FolderContractToOneLevel
FolderCreate
FolderCreateDlg
FolderDeepMove
FolderDelete
FolderDeleteDlg
FolderExpand
FolderExpandAll
FolderLinkTo
FolderLinkToDlg
FolderListAddFolder
FolderListApplyToView
FolderListCreate
FolderListCreateFromView
FolderListDelete
FolderListGetCount
FolderListGetName
FolderListSetFolder
FolderMoveTo
FolderRename
FolderRenameDlg
FolderSelect
FolderSelectDlg
FontBold
FontDlg
FontItalic
FontNormal
FontSet
FontUnderline
FOR-ENDFOR
FOREACH-ENDFOR
FORNEXT-ENDFOR
FRACTION
FUNCTION PROTOTYPE
FUNCTION-ENDFUNC

G

GetAddressBookData
GETCURRENTDIRECTORY
GETFILEATTRIBUTES
GETFILEDATEANDTIME
GETNUMBER
GetOfficeData
GETSTRING
GETUNITS
GLOBAL
GO

H

HelpCoaches
HelpContents
HelpContextSensitive
HelpHowDoI
HelpMacros
HelpSearchFor
HelpWhatIs

I

IF-ELSE-ENDIF
IFPLATFORM-ENDIFPLATFORM
IMEOff
IMEOn
IMEWordRegister
ImportCalendar
ImportCalendarDlg
INCLUDE
INDIRECT
InfoUpdate
InhibitInput
Insert
InsertTab
INTEGER
ItemAccept
ItemAcceptWithCommentDlg
ItemAnnotationGetCount
ItemAnnotationSaveAs
ItemArchive
ItemArchiveOpenItem
ItemAttachmentAdd
ItemAttachmentDelete
ItemAttachmentGetClass
ItemAttachmentGetCount
ItemAttachmentGetCurrentIndex
ItemAttachmentGetDisplayName
ItemAttachmentGetName
ItemAttachmentSaveAs
ItemAttachmentUpdate
ItemComplete
ItemCustomReplyDlg
ItemDecline
ItemDeclineOpenItem
ItemDeclineWithCommentDlg
ItemDelegate
ItemDelegateDlg
ItemDelegateOpenItem
ItemDelete
ItemDeleteOpenItem
ItemFolderAltMove
ItemFolderLink
ItemFolderMove

ItemForward
ItemGetAttribute
ItemGetDate
ItemGetMailboxID
ItemGetOutboxMessageID
ItemGetPriority
ItemGetReplyToMessageID
ItemGetSenderID
ItemGetSource
ItemGetText
ItemGetType
ItemInfo
ItemInfoOpenItem
ItemIsValid
ItemListCreate
ItemListCreateFromControl
ItemListDelete
ItemListGetCount
ItemListGetItem
ItemListSort
ItemMarkPrivate
ItemMessageIDFromView
ItemOpen
ItemRead
ItemReadNext
ItemReadPrevious
ItemReply
ItemResend
ItemRoute
ItemSaveInfo
ItemSaveMessage
ItemSaveMessageDlg
ItemSaveView
ItemSaveViewDlg
ItemSend
ItemSetAlarm
ItemSetAttribute
ItemSetDate
ItemSetItemType
ItemSetPriority
ItemSetText
ItemUndelete
ItemUndeleteOpenItem

J

No index entries for this letter.

K

KEYSTRING

L

LABEL
LOCAL

M

MacroFilePlay
MacroPause
MacroPlayDlg
MacroRecordDlg
MacroStop
MainWindowHide
MainWindowShow
MENU
MENULIST
MESSAGEBOX
MMPLAY
MMSPEAK
MMSPEAKCLIPBOARD
MMSTOPSPEECH
MOUSE STRING
MoviePause
MovieResume

N

NEST
NewAppointment
NEWDEFAULT
NewMail
NewNote
NewPhone
NewTask
NEXT
NextPane
NOTFOUND
NTOC
NUMSTR

O

OfficeCloseViews
OfficeMinimize
OleAttachDlg
OleAttachInsertObject
OleAttachPaste
OleConvertToStatic
OleDoVerb
OleInsertObject
OleInsertObjectDlg
OleLinksDlg

OlePasteLink
ONCANCEL
ONCANCEL CALL
ONDDEADVISE CALL
ONERROR CALL
ONERROR
ONNOTFOUND CALL
ONNOTFOUND
OpenCalendar
OPENFILE
OpenInBox
OpenOutBox
OpenTrashWindow

P

PAUSE
PERSIST
PERSISTALL
PosCharNext
PosCharPrevious
PosLineBegin
PosLineDown
PosLineUp
PosScreenDown
PosScreenLeft
PosScreenRight
PosScreenUp
PosTextTop
PosToEndOfLine
PosToEndOfText
PosWordLeft
PosWordRight
PrefAdvanced
PrefAppointment
PrefAppointmentTime
PrefBusySearch
PrefCleanup
PrefDateTimeFormat
PrefDlg
PrefEnvironment
PrefFolderList
PrefLocationOfFiles
PrefMailAndPhone
PrefNote
PrefTask
PrefViewDefaults
PrevPane
Print
PrintCalendar
PrintCalendarDlg
PrintDlg
PrintSetup
PROCEDURE PROTOTYPE

PROCEDURE-ENDPROC
PROMPT
PromptForPassword
PromptSetMode
Proxy
ProxyDlg

Q

QUIT

R

Refresh
REGIONADDLISTITEM
REGIONENABLEWINDOW
REGIONGETCHECK
REGIONGETSELECTEDTEXT
REGIONGETWINDOWTEXT
REGIONISVISIBLE
REGIONMOVEWINDOW
REGIONREMOVELISTITEM
REGIONRESETLIST
REGIONSELECTLISTITEM
REGIONSETCHECK
REGIONSETFOCUS
REGIONSETWINDOWTEXT
REGIONSHOWWINDOW
RemoteConnect
RemoteCreateModemConnection
RemoteCreateNetworkConnection
RemoteDeleteConnection
RemoteDisconnect
RemoteGetDefaultConnectionName
RemoteGetPhoneNumber
RemoteModemCommand
RemotePendingRequestsDlg
RemoteSelectedRetrieveDlg
RemoteSendRetrieveDlg
RemoteSetAddrBookDnloadFilter
RemoteSetDaylightTimeFormula
RemoteSetDaylightTimeDate
RemoteSetDefaultConnection
RemoteSetItemsFolders
RemoteSetPortInfo
RemoteSetPreferences
RemoteSetPublicGroupDnloadFilter
RemoteSetReconcile
RemoteSetRequestItemsFilter
RemoteSetTimeZone
RemoteViewConnectionLog
RENAMEDIRECTORY
RENAMEFILE

REPEAT-UNTIL
Retrieve
RetrieveFileDlg
RETURN
RuleAddActionAccept
RuleAddActionArchive
RuleAddActionDecline
RuleAddActionDelegate
RuleAddActionEmptyItem
RuleAddActionForward
RuleAddActionLinkToFolder
RuleAddActionMarkPrivate
RuleAddActionMoveToFolder
RuleAddActionReply
RuleAddActionSendMail
RuleCreate
RuleDelete
RuleExecute
RuleListDlg
RUN

S

ScrollLeft
ScrollNext
ScrollPrior
ScrollRight
SelectDown
SelectLeft
SelectLeftWord
SelectPageDown
SelectPageUp
SelectRight
SelectRightWord
SelectToBegLine
SelectToBegText
SelectToEndLine
SelectToEndText
SelectUp
SendAppointment
SENDKEYS WAIT
SendMail
SendNote
SendOptions
SendOptionsAdvanced
SendOptionsDlg
SendPhone
SendTask
SETCURRENTDIRECTORY
SETFILEATTRIBUTES
SETFILEDATEANDTIME
ShelfIconArrange
ShelfIconDelete
ShelfIconOpen

ShelfIconPropertiesDlg
ShelfNewIconDlg
SortDlg
SoundAnnotationList
SoundAnnotationPlay
SPEED
Speller
SplitBar
STRLEN
STRNUM
STRPOS
STRUNIT
SUBCHAR
SUBSTR
SWITCH-ENDSWITCH

T

TabNext
TabPrevious
TextSetAuthority
TextSetBC
TextSetCallerName
TextSetCategoryAndPriority
TextSetCC
TextSetCompany
TextSetDateTime
TextSetFrom
TextSetMessage
TextSetPhoneNumber
TextSetPlaceName
TextSetSubject
TextSetTo
Thesaurus
TOLOWER
TOUPPER
Type
TypeChar

U

UNITSTR
UNTIL
USE
UserButtonAction
UserFunction
UtilWPFileRead
UtilWPFileWrite

V

VARERRCHK

ViewHideMenu
ViewMaximize
ViewMinimize
ViewOnTop
ViewOpen
ViewOpenDlg
ViewOpenFile
ViewOpenNamed
ViewRestore
ViewSculpturing
ViewSwitch
ViewSwitchDlg
ViewUseSystemColors

W

WAIT
WHILE-ENDWHILE
WPCharBox

X

No index entries for this letter.

Y

No index entries for this letter.

Z

No index entries for this letter.

Programming Commands Index

A
B
C
D
E
F
G
H
I
J
K
L
M
N
O
P
Q
R
S
T
U
V
W
X
Y
Z
Print
Close

This index contains only programming commands.   For more information on using programming and product commands, see Macro Basics and Creating Macros.

// (Comment)

A

APPACTIVATE
APPEXECUTE

APPEXECUTEEXT
APPLICATION
APPLOCATE
ASSERT
ASSIGN

B

BEEP
BIFFILEPATH
BIFINFO
BIFREAD
BIFWRITE
BREAK

C

CALL
CANCEL
CASE CALL
CASE
CHAIN
CHARLEN
CHARPOS
CLOSEFILE
COACHANIMATE
COACHFILTERADD
COACHFILTERDESTROY
COACHFILTERDISABLE
COACHFILTERENABLE
COACHGETREGIONINFO
COACHMESSAGEBOX
COACHMESSAGEBOXCLOSE
COACHPROMPT
COACHPROMPTCLOSE
COACHREMOVEDIALOGFILTER
COACHSETDIALOGFILTER
COPYFILE
CREATEDIRECTORY
CTON

D

DDEEXECUTE
DDEEXECUTEEXT
DDEINITIATE
DDEPOKE
DDEREQUEST
DDETERMINATE
DDETERMINATEALL
DECLARE
DEFAULTUNITS

DELETEDIRECTORY
DELETEFILE
DIALOGADDCHECKBOX
DIALOGADDCOLORWHEEL
DIALOGADDCOMBOBOX
DIALOGADDCONTROL
DIALOGADDCOUNTER
DIALOGADDEDITBOX
DIALOGADDFILENAMEBOX
DIALOGADDFRAME
DIALOGADDGROUPBOX
DIALOGADDHLINE
DIALOGADDHOTSPOT
DIALOGADDICON
DIALOGADDLISTBOX
DIALOGADDLISTITEM
DIALOGADDPOPUPBUTTON
DIALOGADDPUSHBUTTON
DIALOGADDRADIOBUTTON
DIALOGADDSCROLLBAR
DIALOGADDTEXT
DIALOGADDVIEWER
DIALOGADDVLINE
DIALOGDEFINE
DIALOGDESTROY
DIALOGDISMISS
DIALOGDISPLAY
DIALOGHANDLE
DIALOGSHOW
DIALOGUNDISPLAY
DIMENSIONS
DISCARD
DLLCALL PROTOTYPE
DLLCALL
DLLFREE
DLLLOAD
DOESDIRECTORYEXIST
DOESFILEEXIST

E

ELSE
ENDAPP
ENDFOR
ENDFUNC
ENDFUNCTION
ENDIF
ENDIFPLATFORM
ENDPROC
ENDPROCEDURE
ENDSWITCH
ENDWHILE
ERROR
EXISTS

F

FILEERROR
FILEFIND
FILEISEOF
FILEPOSITION
FILEREAD
FILESIZE
FILETRUNCATE
FILEWRITE
FOR-ENDFOR
FOREACH-ENDFOR
FORNEXT-ENDFOR
FRACTION
FUNCTION PROTOTYPE
FUNCTION-ENDFUNC

G

GETCURRENTDIRECTORY
GETFILEATTRIBUTES
GETFILEDATEANDTIME
GETNUMBER
GETSTRING
GETUNITS
GLOBAL
GO

H

No index entries for this letter.

I

IF-ELSE-ENDIF
IFPLATFORM-ENDIFPLATFORM
INCLUDE
INDIRECT
INTEGER

J

No index entries for this letter.

K

KEYSTRING

L

LABEL
LOCAL

M

MENU
MENULIST
MESSAGEBOX
MMPLAY
MMSPEAK
MMSPEAKCLIPBOARD
MMSTOPSPEECH
MOUSE STRING

N

NEST
NEWDEFAULT
NEXT
NOTFOUND
NTOC
NUMSTR

O

ONCANCEL CALL
ONCANCEL
ONDDEADVISE CALL
ONERROR CALL
ONERROR
ONNOTFOUND
ONNOTFOUND CALL
OPENFILE

P

PAUSE
PERSIST
PERSISTALL
PROCEDURE PROTOTYPE
PROCEDURE-ENDPROC
PROMPT

Q

QUIT

R

REGIONADDLISTITEM
REGIONENABLEWINDOW
REGIONGETCHECK
REGIONGETSELECTEDTEXT
REGIONGETWINDOWTEXT
REGIONISVISIBLE
REGIONMOVEWINDOW
REGIONREMOVELISTITEM
REGIONRESETLIST
REGIONSELECTLISTITEM
REGIONSETCHECK
REGIONSETFOCUS
REGIONSETWINDOWTEXT
REGIONSHOWWINDOW
RENAMEDIRECTORY
RENAMEFILE
REPEAT-UNTIL
RETURN
RUN

S

SENDKEYS WAIT
SETCURRENTDIRECTORY
SETFILEATTRIBUTES
SETFILEDATEANDTIME
SPEED
STRLEN
STRNUM
STRPOS
STRUNIT
SUBCHAR
SUBSTR
SWITCH-ENDSWITCH

T

TOLOWER
TOUPPER

U

UNITSTR
UNTIL
USE

V

VARERRCHK

W

WAIT
WHILE-ENDWHILE

X

No index entries for this letter.

Y

No index entries for this letter.

Z

No index entries for this letter.

Concepts

Macro Basics
Macro concepts, variables, constants, operators, expressions, control statements, conventions, and hints

Macro Facility
WordPerfect's Macro Facility application

Handles and Message IDs
AddressBook, AddressList, ItemList

Creating Macros
Recording, playing, editing, and writing macros

Dialog Boxes
Dialog box controls, modal dialog boxes, modeless dialog boxes, and creating a dialog box

Coach Overview
Writing coaches

Assigning Macros
Assigning macros to a Button Bar

Reserved Words
Words that cannot be user-defined

Additional Help

Glossary Computer language definitions

Print Help Topics Print multiple help topics

GroupWise Help GroupWise 4.1 Help file

Print Topics

Purpose

You can print one or several Help topics.

Steps

To print a single Help topic,

1 Display the topic.

2 Choose the Print button at the top of the Help window.
or
Choose Print Topic from the File menu.

To print several topics,

1 Choose Print Topics from the File menu.

2 Choose from the following sequences of topics:

Product commands A-E
Product commands F-I
Product commands M-W
Programming Commands

3 Select the topics you want to print.

4 Choose Print.

Each topic is printed one at a time.

Creating Macros

Record a Macro
Play a Macro
Edit a Macro
Writing Macros
Macro Errors
Dialog Boxes

Reserved Words

You should not use reserved words as names for variables, arrays, labels, functions, or procedures.

A

ADDRESS
AND
ANSISTRING
APPACTIVATE
APPEXECUTE
APPEXECUTEEXT
APPLOCATE
APPLICATION
ASSERT
ASSERTCANCEL
ASSERTERROR
ASSERTNOTFOUND
ASSIGN

B

BEEP
BOOL
BREAK

C

CALL
CANCEL
CANCELCONDITION
CANCELOFF
CANCELON
CASE
CASEOF
CENTIMETERS
CHAIN
CHARLEN
CHARPOS
CLOSEFILE
COACHANIMATE
COACHFILTERADD
COACHFILTERDESTROY
COACHFILTERDISABLE
COACHFILTERENABLE
COACHGETREGIONINFO
COACHMESSAGEBOX
COACHMESSAGEBOXCLOSE
COACHPROMPT
COACHPROMPTCLOSE
COACHREMOVEDIALOGFILTER
COACHSETDIALOGFILTER
CONTINUE
COPYFILE
CREATEDIRECTORY
CTON

D

DDEEXECUTE
DDEEXECUTEEXT
DDEINITIATE
DDEPOKE
DDEREQUEST
DDETERMINATE
DDETERMINATEALL
DECLARE
DEFAULT
DEFAULTUNITS
DIGIT
DIMENSIONS

DISCARD
DIV
DLLCALL
DLLFREE
DLLLOAD
DOESDIRECTORYEXIST
DOESFILEEXIST
DWORD

E

ELSE
ENDAPP
ENDFOR
ENDFUNC
ENDFUNCTION
ENDIF
ENDIFPLATFORM
ENDPROC
ENDPROCEDURE
ENDPROMPT
ENDSWITCH
ENDWHILE
ERROR
ERRORCONDITION
ERROROFF
ERRORON
ERRORNUMBER
EXISTS

F

FALSE
FILE
FILEERROR
FILEFIND
FILEISEOF
FILEPOSITION
FILEREAD
FILESIZE
FILETRUNCATE
FILEWRITE
FOR
FOREACH
FORNEXT
FRACTION
FUNCTION

G

GETCURRENTDIRECTORY
GETFILEATTRIBUTES

GETFILEDATEANDTIME
GETNUMBER
GETUNITS
GETSTRING
GLOBAL
GO

H

HIWORD

I

IF
IFPLATFORM
INCHES
INCLUDE
INDIRECT
INPUT
INTEGER

L

LABEL
LENGTH
LETTER
LOCAL
LOWORD

M

MENU
MENULIST
MILLIMETERS
MOD

N

NEST
NEXT
NEWDEFAULT
NOT
NOTFOUND
NOTFOUNDCONDITION
NOTFOUNDOFF
NOTFOUNDON
NTOC
NUMSTR

O

OEMSTRING
OFF
ON
ONCANCEL
ONDDEADVISE
ONERROR
ONNOTFOUND
OPENFILE
OR

P

PAUSE
PERSIST
PERSISTALL
POINTS
POSITION
PROCEDURE
PROMPT
PROTOTYPE

Q

QUIT

R

REAL
REGIONADDLISTITEM
REGIONENABLEWINDOW
REGIONGETCHECK
REGIONGETSELECTEDTEXT
REGIONGETWINDOWTEXT
REGIONISVISIBLE
REGIONMOVEWINDOW
REGIONREMOVELISTITEM
REGIONRESETLIST
REGIONSELECTLISTITEM
REGIONSETCHECK
REGIONSETFOCUS
REGIONSETWINDOWTEXT
REGIONSHOWWINDOW
RENAMEDIRECTORY
RENAMEFILE
REPEAT
RETURN
RETURNCANCEL
RETURNERROR
RETURNNOTFOUND

RUN

S

SENDKEYS
SPEED
STRING
STRLEN
STRNUM
STRPOS
STRUNIT
SUBCHAR
SUBSTR
SWITCH

T

TOLOWER
TOUPPER
TRUE

U

UNITSTR
UNTIL
USE

V

VALUE
VARERRCHK
VARERRCHKOFF
VARERRCHKON
VOID

W

WAIT
WHILE
WORD
WPSTRING
WPUNITS
WP1200THS

X

XOR

Macro Basics

This manual discusses the main statement types used to write a macro: assignments, conditions, loops, comments, and commands. See Macro Language Concepts for brief descriptions, and Variables, Constants, Operators, and Expressions, and Macro Control Statements for detailed descriptions and examples.

Macro Language Concepts
Variables, Constants, Operators, and Expressions
Macro Control Statements
Conventions and Macro Hints

Conventions and Macro Hints

Conventions

1 Programming commands, such as SWITCH, are all uppercase to distinguish them from product commands. Type commands in uppercase, lowercase, or mixed case. The compiler is not case sensitive (see Compilers).

2 Product commands, such as AboutDlg or SendMail, are mixed case.

3 System variables, such as EnvPrefMacroPath or EnvCurrentViewName, are mixed case and begin with Env.

4 Italicized words in a syntax definition are user-defined values.

Macro Hints

1 Wrapping does not affect macro execution.

2 Do not insert spaces in a command name unless the space is part of the syntax.

3 Do not use hard returns, tabs, or indents within a character expression (a character string enclosed in double quotation marks). See Character Expressions.

4 You can use spaces, tabs, indents, and hard returns before and after commands, or between parameters, to make a macro easier to read.

5 WordPerfect Smart Quotes in a macro create a compile-time syntax error. To disable, choose Tools, QuickCorrect, then deselect Enable SmartQuotes.

6 Command syntax must be correct for a macro to compile. Common errors include missing or incorrectly used colons (:), semicolons (;), and double quotation marks (").

7 A macro is automatically compiled the first time it plays. You can also compile macros from Macro Facility (see Compilers).

8 Begin macros with an APPLICATION statement to identify, for the compiler, the application used in the macro. You can include more than one APPLICATION statement (see APPLICATION).

Macro Language Concepts

GroupWise's command-based macro language consists of statements that, when compiled, perform tasks. For example, the AboutDlg command displays the About Novell GroupWise 4.1 dialog box. You combine commands in macros to automate routine tasks and simplify large ones. The following topics introduce macros:

Macros
Assignment Statements
Variables
System Variables
Conditional Statements
Loop Statements
Comment Statements
Command Statements
Command Name
Parameters
Product Commands
Programming Commands
Syntax
Subroutines
Compilers

The above topics are presented in detail in Variables, Constants, Operators, and Expressions, and in Macro Control Statements.

Macros

Automate GroupWise tasks, such as scheduling appointments, creating filters, or displaying Calendar views. Macro tasks consist of instructions called statements. Statements can be assignments, conditions, loops, comments, or commands. For example,

```
ViewOpenNamed (ViewName: "Expanded Mail"; ViewType: Mail!)
TextSetMessage (MessageText: "GroupWise is great!"; Append: True)
```

are commands that open an expanded Mail view and type "GroupWise 4.1 is great!" in the message box (the task). Each command represents one instruction or statement.

The simplest macro consists of only one statement. Complex macros have hundreds of statements. The sequence of statements determines how a macro performs its tasks.

Assignment Statements

Assign the value of an expression to a variable.

x := "John Doe"
Result: x equals John Doe

y := 5
Result: y equals 5

z := 3 + 4
Result: z equals the result of 3 + 4

The assignment operator (:= or =) assigns the value of a right operand expression to a left operand variable. You can also assign values to variables using the ASSIGN programming command (see ASSIGN).

Variables

Represent data that can change during macro execution. Data may include any type of expression, but only one at a time. Variable names are user-defined, are not case sensitive, must begin with a letter, can include any other combination of letters or numbers, and are limited to 50 characters. For example, if you assign "MATTW" to variable vTo and "Weekly meeting" to vSubject, the following commands insert a user ID and subject in the To and Subject text boxes:

```
TextSetTo (ToText: vTo; Append: False)
TextSetSubject (SubjectText: vSubject; Append: False)
```

The lowercase v in vTo and vSubject is an optional convention to identify variables. See Variables in Variables, Constants, Operators, and Expressions.

System Variables

Contain current system information such as clipboard text, the current view name, and default directories. System variables are defined by GroupWise and most begin with Env. For example,

```
vMacroPath := EnvPrefMacroPath
```

assigns the path and name of the default macros directory to a variable named vMacroPath. If you change the directory, system variable EnvPrefMacroPath is updated to reflect the change.

Conditional Statements

Execute a statement or group of statements (statement block) when a specified condition is met. If a macro displays a list of options, it executes a statement block depending on which option the user chooses. Conditional statements include CASE, IF, and SWITCH. See Conditional Statements.

Loop Statements

Execute a statement or statement block repeatedly while or until a specified condition is met. The macro then exits the loop and continues to the next statement. Loop statements include FOR, FORNEXT, FOREACH, REPEAT, and WHILE. See Loop Statements.

Comment Statements

Contain notes and other information that do not affect macro execution. Use comment statements to explain the purpose of your macro and describe its components. Comment statements help if you have to modify a macro months after it is written, or they help if someone else has to understand your macro. A comment begins with // and ends with a hard return [HRt]. See // (comment).

Command Statements

Consist of a name, and optionally one or more parameters. Commands represent an instruction or statement. Command names often describe the action, such as SendMail, DeleteWord, EditCut, and EmptyTrash. See Macro Commands Index.

Command Name

Command names often describe the action, such as SendMail, DeleteWord, EditCut, and EmptyTrash. Command names are not case sensitive and usually do not contain spaces. Exceptions include programming commands that call a subroutine, such as CASE CALL or ONCANCEL CALL (see Subroutines).

A macro can use more than one application. Commands to the non-default application require a *product prefix*, which is specified in an APPLICATION statement. In this example,

```
A1.AboutDlg ()
```

A1 (followed by a period) is the product prefix. It identifies, for the compiler, the application to use. See Compilers and APPLICATION.

Parameters

Contain a constant value (data) that is passed to the compiler or between subroutines when the macro plays (see Compilers). In this example,

FocusSet (Place: CC!)

FocusSet is the command name, and CC! is parameter data. This command places the insertion point in a message box. Parameter names, such as Place, are optional (see Macro Facility).

The beginning of a parameter set is marked with an open parenthesis, and the end with a closed parenthesis. Multiple parameters are separated by a semicolon. Some commands, such as AboutDlg and BusySearch, have no parameters or many parameters.

Product Commands

Execute product features. For example,

FontUnderline (State: On!)

turns underline on. See Macro Commands Index.

Programming Commands

Direct the execution of a macro. For example,

```
IF(x = "Y")
   MacroPlayDlg
ELSE
   MacroRecordDlg
ENDIF
```

displays the Play Macro dialog box if x equals Y, or the Record Macro dialog box if x has any other value. IF, ELSE, and ENDIF are programming commands. MacroPlayDlg and MacroRecordDlg are product commands. See Programming Commands Index.

In Help, programming commands appear in uppercase, product commands in mixed case. Product and programming commands are not case sensitive.

Syntax

Refers to rules that govern the form of macro statements and expressions. For example,

```
Type (Text: "John Doe")
```

types John Doe. The next example,

```
Type (Text: "John Doe"
```

lacks a closing parenthesis. The syntax is not correct and produces an error message. See Macro Commands Index for the correct syntax of macro commands and system variables. See Macro Errors for common syntax errors.

Subroutines

Consist of statements that are executed when the subroutine is called by a calling statement. Subroutines include functions, procedures, and labels.

```
CALL(SubExample)

...(other statements)...

LABEL(SubExample)
   ...statement block...
   RETURN
```

Explanation: The calling statement CALL(SubExample) calls (directs macro execution to) the subroutine LABEL(SubExample). RETURN directs macro execution to the statement that follows CALL(SubExample). Statements in a subroutine are accessible to any part of a macro, and can be called any number of times during execution. See Calling Statements and Subroutines.

Compilers

A computer's central processing unit (CPU) executes instructions written in machine (computer) language. Machine language consists of *binary digits* (bits) 0 and 1. For example, the first three letters of the alphabet in binary notation are 1000001, 1000010, and 1000011. The binary result of 4 + 5 is 1001. Since binary digits are hard to work with, English-based programming languages, such as Basic, Pascal, and C were created. Programs are written with an editor or word-processor and saved as *source files*. *Compilers* create *object files*, which is a copy of the source file translated into machine language. Computers can only execute object files.

Macro languages are similar to programming languages. Macros are written with an editor or word-processor and saved as a source file. When Macro Facility compiles a source file, however, it does not create a separate object file. Instead, it saves the *object* in the source file header, where you cannot see it. When you edit a source file, you must recompile it to update the object. GroupWise can only execute (play) a source file with an object. A compiled source file, or macro, contains instructions Macro Facility interprets and sends to the appropriate application for execution when you play the macro (see Play a Macro).

Variables, Constants, Operators, and Expressions

Variables and constants are joined by operators to form expressions. Expressions often compute values that determine how control statements direct macro execution (see Macro Control Statements).

Variables
Constants
Operators
Expressions

Variables

Point to a memory cell where data is stored. The variable name represents the data's address or location. An address (variable) can store only one data item at a time, which can be of any type and can change during macro execution.

! A run-time error occurs if you reference a variable that has not been assigned a value.

Variable names are user-defined, are not case sensitive, must begin with a letter, can include any other combination of letters or numbers, and are limited to 50 characters. The following examples assign a right operand expression to a left operand variable:

x := "John Doe"
Result: x equals John Doe (character string)

vMeasurement:= 5i
Result: vMeasurement equals 5I (measurement expression)

ResultOfOperation := 3 + 4
Result: ResultOfOperation equals 7 (numeric expression)

z := z + 1
Result: z equals the value of z + 1. Since a variable can contain only one value at a time, the original value of z is lost (unless previously assigned to another variable).

x := y > 1
Result: x equals the result of relational expression y > 1: True if y contains a value greater than 1, or False if y contains a value less than or equal to 1. True and False are Boolean data that represent numeric constants 1 and 0.

The following example evaluates the result of y > 1, without assigning the result to a variable. The computer beeps if the value of y is greater than 1 (if expression is true). The first statement after ENDIF is executed if the value of y is less than or equal to 1 (expression is false).

```
IF(y > 1)
     BEEP
ENDIF
```

Variable Programming Commands

INDIRECT
LOCAL
GLOBAL
PERSIST
DECLARE

Special Variables

ErrorNumber
MacroArgs

MacroDialogResult

INDIRECT

Creates a variable out of a combination of character strings and/or numbers (see INDIRECT). You can use INDIRECT wherever you can use a variable.

LOCAL

Creates variables that only pertain to the current macro (see LOCAL).

GLOBAL

Creates variables that pertain to the current macro and macros called by CHAIN or RUN (see CHAIN, GLOBAL, and RUN).

PERSIST

Creates variables that pertain to any Shared Code macro as long as Shared Code is running (see PERSIST and PERSISTALL).

DECLARE

Declares a local array that only pertains to the current macro. An array can have up to 10 dimensions, and up to 32,767 elements per dimension depending on the amount of available memory (see DECLARE).

MacroDialogResult

An implicit variable (one defined by the application) that contains the *Control* value of the button that dismisses a dialog box. See Creating a Dialog Box and DIALOGSHOW.

MacroArgs

A special array variable that contains values passed to it by CHAIN, NEST, or RUN. For example,

```
// Macro: MAIN.WCM
// Include full path if macro not in default macros directory
RUN("TSTMACRO"; {"x"; "y"; "z"})

// Macro: TSTMACRO.WCM
// Compile, then play MAIN.WCM
vElements = DIMENSIONS(MacroArgs[]; 0)
IF(vElements != 0)
   FORNEXT(x; 1; vElements; 1)
      MESSAGEBOX(z; "Element Values";
         "MacroArgs[" + x + "] = " + MacroArgs[x])
   ENDFOR
ELSE
   BEEP
   MESSAGEBOX(z; "Error"; "No values passed"; IconExclamation!)
ENDIF
```

Result: x y z

ErrorNumber

A special variable that contains the error value of a Cancel, Error, or Not Found condition. See ASSERT.

Constants

A data object with a value that does not change during macro execution.

```
SWITCH(x)
   CASEOF 1: CALL(Start)
   CASEOF 2: CALL(Stop)
ENDSWITCH
```

Explanation: This SWITCH statement contains two CASEOF statements, followed by numeric constants 1 and 2. If variable x equals 1, a subroutine named Start is called. If x equals 2, a subroutine named Stop is called.

See Also

Conditional Statements
SWITCH

Operators

Macros support the following operators:

Unary Operators
Binary Operators
Arithmetic Operators
Relational Operators
Logical Operators
Bitwise Operators

See Also

Operator Precedence

Unary Operators

A symbol or word that represents an operation on only one operand or expression.

Operator	Action
+	Multiplies an operand by +1
-	Multiplies an operand by -1
NOT	Inverts the result of relational and logical expressions

Unary plus/minus (+/-) examples
NOT example
NOT example in an IF statement
NOT example in shorthand notation

Unary plus/minus (+/-) examples

+5
Result: +5 is the result of 5 * +1.

-10
Result: -10 is the result of 10 * -1.

NOT example

```
x := 5
y := (x < 10)
z := NOT(x < 10)
```

Explanation: True is assigned to y (x is less than 10). False is assigned to z (the result of expression x < 10 is inverted).

NOT example in an IF statement

```
x := 5
z := (x < 10)
IF(NOT(z))
   BEEP
ELSE
   QUIT
ENDIF
```

Explanation: If the inverted result of expression x < 10 is True, beep. Since False is the result of NOT(z), there is no beep, and the ELSE statement is executed (QUIT ends the macro).

NOT example in shorthand notation

```
x := 5
IF(NOT(x < 10))
   BEEP
ELSE
   QUIT
ENDIF
```

Explanation: If the inverted result of expression x < 10 equals True, beep. Since the result of NOT(x < 10) equals False, there is no beep, and the ELSE statement is executed (QUIT ends the macro).

Previous example without NOT operator

```
x := 5
IF(x > 10)
   BEEP
ELSE
   QUIT
ENDIF
```

Explanation: If the result of expression x > 10 equals True, beep. Since the result equals False (the value of x is less than 10), the ELSE statement is executed (QUIT ends the macro).

Binary Operators

A symbol that represents an operation on two operands or expressions. For example,

```
x := 3 + 4
```

contains a plus operator (+) which adds the operands 3 and 4. The assignment operator (:=) assigns the result of the arithmetic expression 3 + 4 to variable x.

The binary operators are:

Arithmetic Operators
Relational Operators
Logical Operators
Bitwise Operators

Arithmetic Operators

A symbol or word that represents a mathematical operation on two operands.

Operator	Action
*	Multiplication
/	Division
_	Subtraction
+	Addition
%	Floating point modulus division (returns floating point division remainder)
MOD	Integer modulus division (returns integer division remainder)
DIV	Integer division (returns integer portion of integer division)

Floating point (%) examples
MOD examples
DIV examples

See Also

Arithmetic Expressions

Floating point (%) examples

x := 10.1 % 3
Result: x equals 1.1

x := 9 % 3
Result: x equals 0

MOD examples

x := 10 MOD 3
Result: x equals 1.

x := 10.1 MOD 3
Result: error (cannot use MOD on real numbers).

DIV examples

x := 10 DIV 3
Result: x equals 3.

x := 9 DIV 3
Result: x equals 3.

x := 9.1 DIV 3.5
Result: error (cannot use DIV on real numbers).

Relational Operators

A symbol that represents a relational operation on two operands. The result of the operation equals True or False.

Operator	Action
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to
=	Equal to
<>	Not equal to
!=	Not equal to

Greater than (>) and less than (<) examples
Greater (>=) and less (<=) than or equal to examples
Not equal (<>) example

See Also

Relational Expressions

Greater than (>) and less than (<) examples

x := 10
z := (x > 5)
Result: z equals True (x is greater than 5).

x := 10
z := (x < 5)
Result: z equals False (x is not less than 5).

Not equal (<>) example

x := 10
z := (x <> 5)
Result: z equals True (x is not equal to 5).

Greater (>=) and less (<=) than or equal to examples

```
x := 10
IF(x >= 10)
   BEEP
ELSE
   QUIT
ENDIF
```

Explanation: The result of expression x >= 10 equals True (x equals 10), and the computer beeps. The ELSE statement is skipped (QUIT does not end macro).

```
x := 20
IF(x <= 10)
   BEEP
ELSE
   QUIT
ENDIF
```

Explanation: The result of expression x <= 10 equals False (x is greater than 10), and the computer does not beep. The ELSE statement is executed (QUIT ends macro).

Logical Operators

Words that represent a logical relationship between conditions, or that invert a condition. The condition is the result of a relational expression. The operators are listed in order of precedence.

Operator	Action
NOT	NOT is a unary operator that inverts the result of a relational expression.
AND	Combines two relational expressions. Each expression must be true for the logical experssion to be true.
XOR	Combines two relational expressions. Only one expression can be true for the logical expression to be true. If both are true or both are false, the logical expression is false. Operator is called exclusive OR.
OR	Combines two relational expressions. Only one expression need be true for the logical expression to be true. Operator is called Inclusive OR.

NOT example
AND example
XOR example
XOR example in shorthand notation
OR example

See Also

Logical Expressions

NOT example

```
x := 8
IF((x < 10) AND NOT (x = 5))
   BEEP
ENDIF
```

Explanation: Logical expression is true (x is less than 10 AND NOT x is equal to 5, or x is not equal to 5). The computer beeps.

AND example

x := 1
y := 2
z := ((x = 1) AND (y = 2))

Result: z equals True. The logical AND expression is true because the relational expressions x = 1 and y = 2 are true.

XOR example

x := 1
y := 2
z := ((x = 0) XOR (y = 2))
Result: z equals True (only one relational expression is true)

x := 1
y := 2
z := ((x = 1) XOR (y = 2))
Result: z equals False (both relational expressions are true)

x := 1
y := 2
z := ((x = 0) XOR (y = 1))
Result: z equals False (both relational expressions are false)

XOR example in shorthand notation

```
x := 1
y := 2
IF((x = 0) XOR (y = 2))
   BEEP
ENDIF
```

Explanation: The logical XOR expression is true because relational expression x = 0 is false and relational expression y = 2 is true. The computer beeps.

OR example

```
x := 1
y := 2
z := ((x = 1) OR (y = 5))
```

Explanation: The logical OR expression is true because relational expression x = 1 is true (z equals True).

Bitwise Operators

A symbol that represents a bitwise operation on two integer operands. The operators are listed in order of precedence.

Operator	Action
~	Bitwise unary NOT (one's complement) toggles a binary value (1 to 0, and 0 to 1).
&	Bitwise AND results in 1 if both operand bits are 1, and 0 if not.
|	Bitwise inclusive OR results in 1 if both operands are 1 or operands do not match. Results in 0 if both operands are 0.
^	Bitwise exclusive OR (XOR) results in 0 if operands match, and 1 if not.
<<	Shift bits left.
>>	Shift bits right.

Unary NOT (~) example
AND (&) example
Inclusive OR (|) example
Exclusive OR (^) example
Shift left (<<) example
Shift right (>>) example

See Also

Bitwise Expressions

Unary NOT (~) example

	DEC	BIN
~	-15	1111111111110001
	14	0000000000001110

Explanation: Bitwise unary NOT toggles bits from 1 to 0, and 0 to 1.

AND (&) example

```
    DEC   BIN
    1000  1111101000
  & 31    0000011111
    ____  __________
    8     0000001000
```

Explanation: Bitwise AND turns bits off (result is 0) if one operand is 0. Bits are left on (result is 1) if both operands are 1.

Inclusive OR (|) example

```
    DEC   BIN
    1000  1111101000
  | 27    0000011011
    ____  __________
    1019  1111111011
```

Explanation: Bitwise OR turns bits on (result is 1) if one operand is 1. Bits are left off (result is 0) if both operands are 0.

Exclusive OR (^) example

```
    DEC   BIN
    1000  1111101000
  ^ 40    0000101000
    ____  __________
    960       1111000000
```

Explanation: Bitwise XOR turns bits on (result is 1) if the operands are different. It turns bits off (result is 0) if the operands are the same.

Shift left (<<) example

```
     DEC    BIN
     500        0111110100
<<     1    0000000001

     ____   __________
     1000   1111101000
```

Explanation: Shift bits one position left, and insert one 0 bit at the right end of the binary numeral. Multiplies the original value by 2.

Shift right (>>) example

```
    DEC   BIN
    1000  1111101000
>>     1  0000000001

    ____  __________
    500       0111110100
```

Explanation: Shift bits one position right, and insert one 0 bit at the left end of the binary numeral. Divides the original value by 2.

Operator Precedence

The following table shows operator precedence, which is applied to expressions with two or more operators.

Order	Operators
1	parentheses (), unary minus (-), unary plus (+), bitwise not (~), logical not (NOT)
2	multiply (*), divide (/), modulus division (% or MOD), integer division (DIV)
3	add (+), substract (-)
4	shift left (<<), shift right (>>)
5	less than (<), less than or equal to (<=), greater than (>), greater than or equal to (>=), not equal (<>), equal (=)
6	bitwise and (&), bitwise or (|), bitwise xor (^)
7	logical and (AND), logical xor (XOR)
8	logical or (OR)

The following rules apply to operators:

- Operators with the same precedence are evaluted from left to right.
- Operators inside parentheses are evaluated before operators outside parentheses.
- Operators inside nested parentheses are evaluated from the innermost parentheses out.

Operator Precedence Examples

Operator Precedence Examples

x := ((50 * 5 + 50) * 3 + 100)
Result: x equals 1000 (50 * 5 = 250, 250 + 50 = 300, 300 * 3 = 900, 900 + 100 = 1000)

x := ((50 * (5 + 50)) * 3 + 100)
Result: x equals 8350 (5 + 50 = 55, 55 * 50 = 2750, 2750 * 3 = 8250, 8250 + 100 = 8350)

x := ((50 * 5 + 50) * (3 + 100))
Result: x equals 30900 (50 * 5 = 250, 250 + 50 = 300, 300 * 103 = 30900)

Expressions

Macros support the following expressions:

Numeric Expressions
Measurement Expressions
Character Expressions
Arithmetic Expressions
Relational Expressions
Logical Expressions
Bitwise Expressions

See Also

Radix Base Conversions

Numeric Expressions

Numeric constants or variables, or a combination of the two joined by a numeric operator. A numeric operator is a word or symbol that operates on numeric data (see Unary Operators and Binary Operators).

Given that x equals 3, the following examples are valid numeric expressions:

5
Explanation: Numeric constant.

x
Explanation: Variable containing a numeric value of 3.

x * 5
Explanation: Expression (multiply x and 5). The expression x * 5 is also an arithmetic expression.

+5
Explanation: Unary plus constant.

-(x + 10)
Explanation: Unary minus expression (negate the result of x plus 10)

Measurement Expressions

Constants or variables containing a measurement value, or combination of the two joined by a numeric operator.

Given that z equals 4i (inches), the following examples are valid measurement expressions:

5c
Explanation: Constant (5 centimeters).

z
Explanation: Variable that contains a measurement value of 4 inches.

z * 10i
Explanation: Expression (multiply z and 10 inches).

-z
Explanation: Unary minus (negative 4 inches).

The valid units of measure are:

Unit of Measure	Measurement Type
"	inches
i	inches
c	centimeters
m	millimeters
p	points (72 per inch)
w	WP units (1200 per inch)

Character Expressions

A character constant (letter, digit, or keyboard symbol) or variable, or combination of the two concatenated by the plus operator (+) or compared by a relational operator such as > (greater than). Character constants and character constants assigned to variables must be enclosed in double quotation marks.

Enclose a character constant in single quotation marks to specify an ASCII numeric value (see CTON). For example,

x := 'A'
Result: x equals 65

x := 'A' + 'B'
Result: x equals 131 (65 + 66)

A character string with double quotation marks is enclosed with two sets of double quotation marks. For example,

x := "His name is ""John"" Doe"
Result: x equals His name is "John" Doe

x := """John Doe"""
Result: x equals "John Doe" (outside quotation marks are always required)

The following examples are valid character expressions (notice the space after Joe and before Jr.).

"John Doe"
Explanation: Character string.

z := "Joe " + "Doe"
Explanation: Expression assigned to variable z (z equals John Doe).

x := z + ", Jr."
Explanation: Expression assigned to variable x (x equals John Doe, Jr.).

If you concatenate a character string and a number, the number is converted to a character string. If you concatenate a numeric character and a number, the numeric character is converted to a number and the two are added.

x := "A" + 1
Result: x equals A1 (character string)

x := "1" + 1
Result: x equals 2 (number)

x := ("A" + (1 + 3))
Result: x equals A4 (the result of a mathematical operation (1+3=4) converted to a character string before being concatenated to A)

Compare the previous example with the following examples (operation precedence is left to right):

x := ("A" + 1 + 3)
Result: x equals A13 (character string--numbers converted and not added).

x := (1 + 3 + "A")
Result: x equals 4A (character string--numbers added and then converted to a character string).

Arithmetic Expressions

A statement that represents an arithmetic operation, or one that contains two operands joined by an arithmetic operator. The operation result is a numeric value.

x := 1 + 2
Result: x equals 3

x := 3 * 3
Result: x equals 9

x := "2" * 3 * 4
Result: x equals 24 (2 converted to a number then multiplied)

x := "A" * 2
Result: error (cannot multiply letters and numbers)

See Also

Arithmetic Operators

Relational Expressions

A statement that represents a relational operation, or one that contains two operands joined by a relational operator. The operation result equals True or False.

Given that x equals 5,

z := (x > 6)
Result: z equals False (5 is less than 6)

z := (x >= 5)
Result: z equals True (5 equals 5)

z := ("Ab" > "Bb")
Result: z equals False (Ab is less than, or comes before, Bb)

z := ("Ab" <> "Bb")
Result: z equals True (Ab is not equal to Bb)

Given that x equals "A", y equals "B", and z equals "a", the following expressions return True or False in variable w:

w := (x > y)
Result: w equals False (uppercase A is less than, or comes before, uppercase B)

w := (x > z)
Result: w equals True (uppercase A is greater than, or comes after, lowercase a)

See Also

Relational Operators

Logical Expressions

A statement that represents a logical operation, or one that contains two relational expressions joined by a logical operator. The operation result equals True or False.

Given that x equals 10, y equals 5, and z equals 20, the following expressions return True or False in variable w:

w := ((x > y) AND (y < z))
Result: w equals True (both relational expressions are true)

w := ((x < y) AND (y < z))
Result: w equals False (first relational expression is false)

w := NOT(y > z)
Result: w equals True (5 is not greater than 20)

w := ((x > 5) AND (y < 20) AND (z = 20))
Result: w equals True (all relational expressions are true)

w := (((x < 5) AND (y > 20)) OR (z = 20))
Result: w equals True (z = 20 is true)

w := (((x < 5) AND (y > 20)) OR NOT (z = 20))
Result: w equals False (all relational expressions are false)

See Also

Logical Operators

Bitwise Expressions

A statement that represents a bitwise operation, or one that contains two operands joined by a bitwise operator.

Bitwise NOT (~) example
Bitwise AND (&) example
Bitwise inclusive OR (|) example
Bitwise XOR (^) example
Bitwise shift left (<<) example
Bitwise shift right (>>) example

See Also

Bitwise Operators

Bitwise NOT (~) example

x := ~(-15)
Result: x equals 14 (one's complement)

x := ~(-15) + 1
Result: x equals 15 (two's complement)

Bitwise AND (&) example

x := 65535 & 535
Result: x equals 535

Previous example in binary notation

```
    1111111111111111 (binary 65535)
  & 0000001000010111 (AND binary 535)
    ________________
    0000001000010111 (result is binary 535)
```

Explanation: Bitwise AND operation result is 1 if both operands are 1, or 0 if not.

Bitwise inclusive OR (|) example

x := 65535 | 535
Result: x equals 65,535

Previous example in binary notation

```
    1111111111111111 (binary 65535)
  | 0000001000010111 (OR binary 535)
    ________________
    1111111111111111 (result is binary 65535)
```

Explanation: Bitwise OR operation result is 1 if one operand is 1, or 0 if both operands are 0.

Bitwise XOR (^) example

x := 65535 ^ 535
Result: x equals 65,000

Previous example in binary notation

```
    1111111111111111 (binary 65535)
  ^ 0000001000010111 (XOR binary 535)
    ________________
    1111110111101000 (result is binary 65,000)
```

Explanation: Bitwise XOR operation result is 0 if both operands are 0 or 1, or 1 if operands do not match.

Bitwise shift left (<<) example

x := 65535 << 1
Result: x equals 131,070

Previous example in binary notation

```
     1111111111111111   (binary 65535)

<<  ‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾
    1111111111111110   (result is binary 131,070)
```

Explanation: Bitwise shift left operator shifts bits one position left, and inserts one 0 bit at the right end of the binary numeral. The result is the original value multiplied by 2.

Bitwise shift right (>>) example

x := 65535 >> 1
Result: x equals 32,767

Previous example in binary notation

```
     1111111111111111   (binary 65535)

<<  0111111111111111   (result is binary 32,767)
```

Explanation: Bitwise shift right operator shift bits one position right, and inserts one 0 bit at the left end of the binary numeral. The result is the original value divided by 2.

Radix Base Conversions

The radix is the base of a number system. The conversion choices are,

h or x Hexadecimal (radix is 16)
o Octal (radix is 8)
b Binary (radix is 2)

For example,

x := 1Ah
Result: x equals 26

x := 1111b
Result: x equals 15

x := 44o
Result: x equals 36

All radix bases must begin with a number. Precede non-numeric hex digits with 0. For example,

x := 0Ah
Result: x equals 10

Macro Control Statements

Control statements alter the sequential execution of macro commands. For example,

```
TextSetSubject (SubjectText: "Meeting Thursday"; Append: False)
TextSetSubject (SubjectText: "Meeting Friday"; Append: False)
```

inserts "Meeting Thursday" in a Subject text box, and then "Meeting Friday" in the same text box. The second command overrides the first. A control statement can select which MarginLeft to execute depending on a condition being true.

```
SWITCH(Test)
    CASEOF 1: TextSetSubject ("Meeting Thursday"; False)
    CASEOF 2: TextSetSubject ("Meeting Friday"; False)
    DEFAULT: TextSetSubject ("Meeting Monday"; False)
ENDSWITCH
```

Explanation: If variable *Test* equals 1, insert Meeting Thursday. If *Test* equals 2, insert Meeting Friday. If *Test* equals any value except 1 or 2, insert Meeting Monday.

Macros support the following control statements:

Conditional Statements
Loop Statements
Calling Statements
Subroutines

See Also

Macro File Libraries
SWITCH

Conditional Statements

Execute a statement or statement block when an expression is true, or a variable matches a constant.

Macros support the following conditional statements:

CASE
IF-ELSE-ENDIF
SWITCH-ENDSWITCH

CASE

Executes a LABEL statement when *Test* (user-defined variable) matches a constant value.

```
CASE(Test; {1; Start; 2; Next}; Other)

...(other statements)...

LABEL(Start)
   ...statement block...

LABEL(Next)
   ...statement block...

LABEL(Other)
   ...statement block...
```

Explanation: If *Test* matches 1, call LABEL(Start). If *Test* matches 2, call LABEL(Next). If there is no match, call LABEL(Other). CASE CALL is a similar statement which expects a RETURN after a LABEL statement. See CASE and CASE CALL.

IF-ELSE-ENDIF

Executes a statement or statement block when an expression is true.

```
IF(x = 5)
   ...statement block...
ELSE
   ...statement block...
ENDIF
```

Explanation: Execute the first statement block if the expression x = 5 is true (if x equals 5). If not true, execute the second statement block. ELSE is optional.

```
IF(Expression)
   ...statement block...
ENDIF
```

Explanation: If *Expression* is true, execute statement block. If not true, execute the first statement after ENDIF. *Expression* must evaluate to True or False. See Relational Expressions and IF.

SWITCH-ENDSWITCH

Executes a statement or statement block when *Test* matches *Constant*.

```
SWITCH(Test)
   CASEOF Constant1:
        ...statement block...
   CASEOF Constant2:
        ...statement block...
   CASEOF Constant3:
        ...statement block...
   DEFAULT:
        ...statement block...
ENDSWITCH
```

Explanation: If *Test* matches *Constant1*, the statement block after CASEOF *Constant1* is executed. Statement blocks can include calling a subroutine (see Subroutines). If CONTINUE follows a statement block, the next statement block is automatically executed. See SWITCH.

Loop Statements

Execute a statement or statement block a specified number of times, until an expression is true, or while an expression is true.

Macros support the following loop statements:

FOR-ENDFOR
REPEAT-UNTIL
WHILE-ENDWHILE

FOR-ENDFOR

Executes a statement or statement block a specified number of times. The general form is,

```
FOR(ControlVariable; InitialValue; TerminateExp; IncrementExp)
   ...statement block...
ENDFOR
```

Explanation: *InitialValue* initializes *ControlVariable*. *TerminateExp* tests the value of *ControlVariable*. *IncrementExp* increases the value of *ControlVariable* until *TerminateExp* is true and the loop ends.

```
FOR(x; 1; x < 5; x + 1)
   ...statement block...
ENDFOR
```

Explanation: x is initialized to 1, statement block executes while x is less than 5, x is incremented by 1 at the end of each loop. Similar loop statements are FOREACH-ENDFOR and FORNEXT-ENDFOR. See Relational Expressions, FOR, FOREACH, and FORNEXT.

REPEAT-UNTIL

Executes a statement or statement block until an expression is true. All REPEAT statements execute at least once because the expression is checked at the end of the loop.

```
REPEAT
   ...statement block...
UNTIL(x >= 10)
```

Explanation: Execute *statement block* until the expression x >= 10 is true (until x is greater than or equal to 10). The value of x must change in the loop to make the expression true or the loop never ends. For example,

```
REPEAT
   ...statement block...
   x := x + 1
UNTIL(x >= 10)
```

Explanation: Increment x by 1 (x := x + 1) at the end of each loop until x is greater than or equal to 10. See Relational Expressions and REPEAT.

WHILE-ENDWHILE

Executes a statement or statement block while an expression is true. WHILE statements never execute unless the expression is true because the expression is checked at the start of the loop. For example,

```
WHILE(x <= 10)
   ...statement block...
ENDWHILE
```

Explanation: Execute statement block while the expression x <= 10 is true (while x is less than or equal to 10). If x is greater than 10, the loop does not execute. The value of x must change in the loop to make the expression true or the loop never ends. For example,

```
WHILE(x <= 10)
 ...statement block...
   x := x + 1
ENDWHILE
```

Explanation: Increment x by 1 (x := x + 1) at the end of each loop while x is less than or equal to 10. See Relational Expressions and WHILE.

Calling Statements

Call a subroutine which consists of one or more statements (see Subroutines).

Macros support the following calling statements:

CALL
GO
Subroutine Name

CALL

Has one parameter which is the name of a subroutine to call. RETURN directs macro execution to the statement that follows CALL (see CALL and RUN).

```
CALL(ExSub)

   ...other statements...

LABEL(ExSub)
   ...statement block...
   RETURN
```

Explanation: CALL(ExSub) directs macro execution to LABEL(ExSub), where statement block is executed. RETURN directs macro execution to the first statement after CALL(ExSub). See CALL.

GO

Has one parameter which is the name of a subroutine to jump to. Macro execution continues from the point of the subroutine and does not return (statements between GO and the subroutine do not execute). RETURN ends a macro or directs macro execution to the statement that follows a RUN command (see GO and RUN).

```
GO(ExSub)

   ...other statements...

LABEL(ExSub)
   ...statement block...
   RETURN
```

Explanation: GO(ExSub) directs macro execution to LABEL(ExSub), where statement block is executed. RETURN ends the macro.

Subroutine Name

Performs the same action as a CALL statement. For example,

```
CALL InitializeVariables(Parameter1; Parameter2)
InitializeVariables(Parameter1; Parameter2)
```

both call a function or procedure named InitializeVariables (see Subroutines, LABEL, FUNCTION, and PROCEDURE). If the second example calls a function, you can assign a return value to a variable with a statement such as

```
x := InitializeVariables(Parameter1; Parameter2)
```

See RETURN.

Subroutines

Consist of one or more statements that execute when the subroutine is called by CALL, GO, or the name of the subroutine.

Macros support the following subroutines:

Labels
Functions
Procedures

Labels

Have one parameter which is the name of the subroutine. Label names must begin with a character, consist of one or more letters or numbers, are limited to 30 characters, and have an optional trailing @ sign. Labels generally include one or more statements followed by RETURN or QUIT. LABEL statements execute the same as other macro statements, and do not have to be called. See LABEL.

Functions

Can have one or more parameters which receive a value from a calling statement (see Calling Statements), and return a value. A function without parameters performs like a LABEL, except it does not execute unless called. Function names must begin with a character, consist of one or more letters or numbers, are limited to 30 characters, and have an optional trailing @ sign. The following rules apply to functions:

1 Functions begin with the word FUNCTION, and end with ENDFUNC or ENDFUNCTION.
2 Functions must be called to execute.
3 Calling statements consist of a function name, which can be followed by one or more parameter values that pass to the function. If there are no parameters, empty parentheses must follow the function name.
4 The number of parameters in a calling statement must equal the number of function parameters.
5 A function cannot be defined inside another subroutine.
6 Functions can be placed anywhere in a macro or macro library file (see USE, INCLUDE).
).
7 Functions can be called from another subroutine, or a function can call itself (functions are recursive).
8 Function variables are private to the function. A variable with the same name as a function variable can be used elsewhere in the macro without conflict.
9 Functions can include LABEL statements which are not visible outside the function. A LABEL statement inside a function should *not* have the same name as a function or procedure.
10 Functions accept RETURN statements with no parameters (return 0); or a value contained in a variable that is the result of a function; or an enumerated type that asserts a Cancel, Error, or Not Found condition; or an enumerated type that asserts a Cancel, Error, or Not Found condition, and a value contained in a variable that is the result of a function operation. See RETURN.
11 Function names cannot be reserved words (see Reserved Words).

Procedures

Can have no parameters, or one or more parameters which receive a value from a calling statement (see Calling Statements). A procedure cannot return a value. A procedure without parameters performs like LABEL, except it does not execute unless called. Procedure names must begin with a character, consist of one or more letters or numbers, are limited to 30 characters, and have an optional trailing @ sign. The following rules apply to procedures:

1 Procedures begin with the word PROCEDURE and end with ENDPROC or ENDPROCEDURE.
2 Procedures must be called to execute.
3 Calling statements consist of a procedure name, which can be followed by one or more parameter values that are passed to the procedure. If there are no parameters, empty parentheses must follow the procedure name.
4 The number of parameters in a calling statement must equal the number of procedure parameters.
5 A procedure cannot be defined inside another subroutine.
6 Procedures can be placed anywhere in a macro or macro library file (see USE, INCLUDE).
7 Procedures can be called from another subroutine, or a procedure can call itself (procedures are recursive).
8 Procedure variables are private to the procedure. A variable with the same name as a procedure variable can be used elsewhere in the macro without conflict.
9 Procedures can include LABEL statements which are not visible outside the procedure. A LABEL statement inside a procedure should *not* have the same name as a procedure or function.
10 Procedures accept RETURN statements with no parameters (directs macro execution to the statement that follows a procedure's caller); or one parameter (an enumerated type that asserts a Cancel, Error, or Not Found condition). See RETURN.
11 Function names cannot be reserved words (see Reserved Words).

Macro File Libraries

Contain functions and/or procedures that can be called from another macro, and must be compiled.

INCLUDE
Specifies a macro file with executable statements, functions, and/or procedures that can be called from another macro. Playing the main macro below

```
//INCLUDE macro: INC.WCM
APPLICATION (A1; "WPOffice"; Default; "US")
    Type("2")

    PROCEDURE TestProc(x)
        Type(x)
    ENDPROC

//Main macro: PARENT.WCM
APPLICATION (A1; "WPOffice"; Default; "US")
    Type("1")
    INCLUDE("C:\OFFICE40\MACROS\INC.WCM")
    Type("3")
    CALL TestProc(4)
    QUIT
```

displays the following:

```
1234
```

Explanation: The main macro types 1. The INCLUDE macro types 2 (INC.WCM). The main macro types 3. The main macro calls TestProc procedure in the INCLUDE macro, which types 4.

USE
Specifies a macro that contains functions and/or procedures. USE usually precedes calling statements to a macro file library. The following example contains two functions:

```
FUNCTION Add(x)
    x := x + 50
    RETURN(x)
ENDFUNC

FUNCTION Subtract(x)
    x:= x - 25
    RETURN(x)
ENDFUNC
```

Explanation: Function Add receives a value in variable x (see Calling Statements). 50 is added to x and returned to the functions caller. Function Subtract receives a value in a variable named x. 25 is substracted from x and returned to the functions caller. If this example was saved and compiled as LIBRARY.WCM, the following example includes statements that call the Add and Subtract functions:

```
APPLICATION(A1; "WPOffice"; Default; "US")
USE("C:\OFWIN40\MACROS\LIBRARY.WCM")
z := Add(50)
z := Subtract(z)
IF(z = 75)
     BEEP // computer beeps because z equals 75
ENDIF
```

Explanation: After function Add is called, 100 is returned in variable z. After function Subtract is called, 75 is returned in variable z. The computer beeps because the expression z = 75 is true.

The previous example with function calling statements in shorthand notation:

```
APPLICATION(A1; "WPOffice"; Default; "US")
USE("C:\WPWIN60\MACROS\LIBRARY.WCM")
z := Add(Subtract(50))
IF(z = 75)
     BEEP // computer beeps because z equals 75
ENDIF
```

Record a Macro

Recording a macro saves a series of keystrokes and/or mouse actions to play (repeat) later. Recorded macros automate routine tasks, such as typing reminders in a message box, displaying a dialog box for user input, or sending an e-mail item.

Steps

1 Choose Tools, Macro, Record (or press Ctrl+F10) to display the Record Macro dialog box.

2 Type a filename in the edit box. The default extension .wcm is automatically added when you choose Record unless you type a different extension. Type a path if different from the default macros directory. To insert a filename, select a name from the list box.

   To change the default extension, start WordPerfect Macro Facility. From the main menu, choose Settings, Options, then type a file extension in the Extension edit box.

3 Choose Record to dismiss the dialog box. The insertion point changes to a circle with a slash to indicate that keystrokes and/or mouse actions are being recorded.

4 Perform a series of keystrokes and/or mouse actions. For example, choose Send, New Mail to open the default Mail view.

5 Choose Tools, Macro, Stop to stop recording. The macro is saved in the default macros directory unless you specified a different path (step 2).

Play a Macro

After recording, playing a macro accomplishes in one step what required several steps to achieve during the recording session.

Steps

1. Choose Tools, Macro, Play (or press Alt+F10) to display the Play Macro dialog box.

2. Type a filename, including path and extension if necessary, then choose Play. To insert a filename, select a name from the list box.

3. Choose Play to dismiss the dialog box and play the macro. Macros automatically compile the first time they are played.

Edit a Macro

Macros are saved as document files, with a compiled macro object hidden at the beginning of each file (see Compilers). You can write, edit, and save a macro as you would any other text document. You can edit macros in a message box, WordPerfect (Windows or DOS), or a text editor such as Windows' Notepad.

Steps (WordPerfect for Windows 6.0a)

1. Choose Tools, Macro, Edit to display the Edit Macro dialog box.

2. Type a filename, including path and extension if necessary, then choose Edit. To insert a macro filename, choose the Directory button to display the Select File dialog box, then double-click the desired filename.

3. Choose Edit to dismiss the dialog box, display the Macro Feature Bar, and open the macro.

4. Edit the macro by deleting commands or typing new ones.

Macro Errors

You can introduce errors when you edit a macro (see Syntax). If you type ViewOpenName instead of ViewOpenNamed, a compile-time syntax error displays a syntax error message box.

The compiler does not recognize ViewOpenName, and guesses that it is a function or procedure calling statement (see Calling Statements and Subroutines). When it reaches the end of the file and does not find a corresponding subroutine, it displays a Syntax Error message. Common syntax errors are,

- Missing semicolons between parameters.
- Missing parentheses.
- Missing double quotation marks (see Character Expressions).
- Missing command in a conditional or loop statement (see Conditional Statements and Loop Statements).
- Misspelled macro command names.
- Undefined calling statements (see Calling Statements).

The compiler identifies syntax errors and suggests solutions for the writer to verify.

Run-time error messages identify errors that occur while a macro is playing and where the errors occur.

Writing Macros

A recorded macro includes only product commands. When you play a recorded macro, each action is executed in recorded order (see Record a Macro).

A written macro can also include non-recordable statements that specify how a macro performs a task. The writer specifies the conditions for executing assignments, loops, and programming commands (see Macro Language Concepts). Writing a macro provides greater flexibility to determine function than recording a macro.

Dialog Boxes

Users interact with applications through menus and dialog boxes. Dialog boxes can display more than one option for user input. For example, in the Busy Search Settings dialog box, default settings are displayed in edit boxes next to button controls. Other edit boxes are left blank for the user to fill in. To display the Busy Search Settings dialog box, choose Send, Busy Search from the Main Window or an Appointment view (an elipsis to the right of a menu option indicates a dialog box is available).

To change settings, enter new values in the edit boxes, or select a button control (see DIALOGADDPOPUPBUTTON). Edit boxes, check boxes, and buttons are Windows controls for user input. Controls available for a user-defined dialog box are described below.

Dialog Box Controls
Creating a Dialog Box with Macro Commands
Creating a Dialog Box with Dialog Editor
Recording a Dialog Box
Modal Dialog Boxes
Modeless Dialog Boxes

Dialog Box Controls

Are input or output windows where the user interacts with a dialog box and its parent application. Macros supports the following dialog box controls:

Check Boxes
Color Wheels
Combination Boxes
Counter Buttons
Edit Boxes
Filename Edit Boxes
Frames
Group Boxes
Horizontal Lines
Hot Spots
Icons
List Boxes
Popup Buttons
Push Buttons
Radio Buttons
Scroll Bars
Static Text Controls
Viewers
Vertical Lines

Check Boxes

Display one or more options. Use a callback function to activate user-defined responses. See DIALOGADDCHECKBOX.

Color Wheels

Display colors to select. See DIALOGADDCOLORWHEEL.

Combination Boxes

Display an edit box and a list box. Enter text such as a filename in the edit box, or double-click a list item to insert. See DIALOGADDCOMBOBOX.

Counter Buttons

Display an edit box and a counter button. Enter a number in the edit box, or click the counter button to insert a number. See DIALOGADDCOUNTER.

Edit Boxes

Receive text input. There are different styles of edit controls, including single and multiple line. See DIALOGADDEDITBOX.

Filename Edit Boxes

Display an edit control and a button control. Enter a filename in the edit control, or click the button to display the Select File dialog box. See DIALOGADDFILENAMEBOX.

Frames

Group items in a dialog box. Accept no input. See DIALOGADDFRAME.

Group Boxes

Group items in a dialog box with a titled frame. Accept no input. See DIALOGADDGROUPBOX.

Horizontal Lines

Separate items in a dialog box. Accept no input. See DIALOGADDHLINE.

Hot Spots

Are invisible controls that close a dialog box when the user clicks a defined area. Redefine the response with a callback function. See DIALOGADDHOTSPOT.

Icons

Display graphic representatinos. Accept no input, unless used in a callback function with DIALOGADDHOTSPOT. See DIALOGADDHOTSPOT, DIALOGADDICON, DIALOGSHOW.

List Boxes

Display a list of options to choose from. There are different styles of list boxes, including single and multicolumn. See DIALOGADDLISTBOX.

Popup Buttons

Display the name of a menu item on a push button. Click the button to display the menu. See DIALOGADDPOPUPBUTTON.

Push Buttons

Activate user-defined responses when chosen. See DIALOGADDPUSHBUTTON.

Radio Buttons

Display mutually-exclusive options. Use a callback function to activate user-defined responses. See DIALOGADDRADIOBUTTON and DIALOGSHOW.

Scroll Bars

Scroll through documents, or activate user-defined responses with a callback function. See DIALOGADDSCROLLBAR and DIALOGSHOW.

Static Text Controls

Send text to a dialog box. Accept no input. See DIALOGADDTEXT.

Viewers

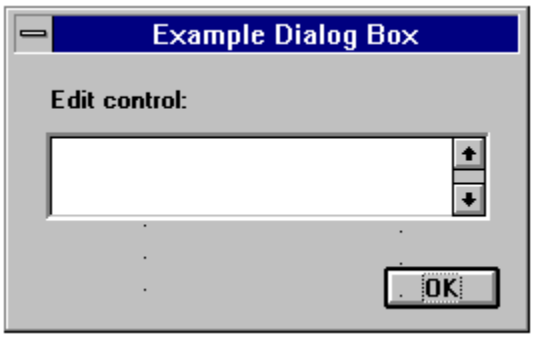
Display read-only text files. See DIALOGADDVIEWER.

Vertical Lines

Separate items in a dialog box. Accept no input. See DIALOGADDVLINE.

Creating a Dialog Box with Macro Commands

To create a dialog box with macro commands, define it, add controls, and display it. You should dismiss the dialog box when the macro ends. The commands presented below create the following dialog box:



Click the example button above to display the complete macro.

Define
Specify the size and style of the dialog box, and the controls it will contain. The following command,

DIALOGDEFINE(Dialog: 1000; Left: 50; Top: 50; Width: 150; Height: 100; Style: OK! | Percent!; Caption: "Example Dialog Box")

defines (creates in memory) a dialog box with the following specifications:

Dialog: The number 1000 was chosen to identify the dialog box. A name such as "MainDialog" could also have been used. The number or name must be unique. It is used again to add controls, display, and then destroy the dialog box when the macro ends.

Left, Top, Width, Height: The dialog box is displayed in the center of the screen by setting the left and top parameters to 50 and using the Percent! style. Dialog boxes are positioned and sized in dialog units. A vertical unit equals 1/8 the font height, and a horizontal unit equals 1/4 the font width.

Style: The Percent! formula is,

(*screen width or height* **-** *dialog box width or height*) * *percentage*

It equals the number of dialog units from the left side of the screen to the left side of the dialog box, or the number of dialog units from the top of the screen to the top of the dialog box. The dialog box has an OK push button control.

Caption: The dialog box title is "Example Dialog Box".

Parameter names: Parameter names are optional. For example,

DIALOGDEFINE(1000; 50; 50; 200; 125; OK! | Percent!; "Example Dialog Box")

defines the same dialog box as the command above.

Add Controls

The following commands,

DIALOGADDTEXT(Dialog: 1000; Control: 100; Left: 10; Top: 10; Width: 50; Height: 15; Style: Left!; Text: "Edit control:")

DIALOGADDEDITBOX(Dialog: 1000; Control: 101; Left: 10; Top: 25; Width: 125; Height: 25; Style: Left! | VScroll! | Multiline!; vReturn; 1000)

add a static control and an edit control to the dialog box with the following specifications (see Dialog Box Controls):

Dialog: 1000 for both controls, which equals the *Dialog* parameter of the parent dialog box (see *Define* above).

Control: The controls have unique Control numbers (100 and 101), which could also be names such as "Control 1" and "Control 2". Old-style dialog boxes used Control values to give a control the initial input focus (see Creating a Dialog Box with Dialog Editor).

The value of the *Control* used to dismiss the dialog box is returned in the implicit variable MacroDialogResult (see DIALOGDEFINE): 1 for OK, 2 for Cancel, 2 for Close (system menu box), 2 if you double-click the system menu box, 2 if you press Alt+F4, or the value of the *Control* parameter of a user-defined push button or hot spot (see DIALOGADDPUSHBUTTON and DIALOGADDHOTSPOT).

Left, Top, Width, Height: The controls are displayed 10 dialog units from the left side of the dialog box, and 10 and 25 units from the top. The widths are 50 and 175 dialog units, and the heights are 15 and 50 dialog units. A vertical dialog unit equals 1/8 the font height, and a horizontal dialog unit equals 1/4 the font width.

Style: The static text is left justified. The edit box is multi-line with a vertical scroll bar. Combine styles with bitwise OR operator (|) (see Bitwise Operators).

Text: The static control text is "Edit Control".

MacroVar, LimitText: Text typed into the edit box is assigned to variable vReturn when the dialog box is dismissed by OK or Close on the system menu. The maximum number of characters the edit control accepts is 1000.

Display

The following command,

DIALOGSHOW("1000"; "WPOffice.GroupWise")

displays dialog box 1000 and gives the input focus to control 101 (edit control has the insertion point).

Dismiss

The following command,

DIALOGDISMISS("1000"; "OKBttn")

dismisses dialog box 1000, and clears the value of the implicit variable MacroDialogResult. If you need this value, assign it to a another variable before executing DIALOGDISMISS. If you do not destroy a dialog box before the macro ends, memory conflicts may occur.

Creating a Dialog Box with Dialog Editor

With Dialog Editor you can visually create and edit dialogs for GroupWise macros. Open Dialog Editor from the WordPerfect Macro Facility or from within WordPerfect 6.1 for Windows.

Steps

To open the Macro Dialog Editor from the Macro Facility,

1 Open the Macro Facility (run ...\WPC20\MFWIN20.EXE from Windows Program Manager).
2 Choose Dialog from the Macros menu.
3 Specify a macro filename, then choose OK.
4 Choose Create, type a name for the dialog, then choose OK.
or
Select the dialog you want, then choose Edit.

To open the Macro Dialog Editor from within WordPerfect,

1 Choose Macro from the Tools menu, then choose Edit.
2 Specify a macro filename, then choose OK to display the Macro feature bar.
3 Click Dialog Editor on the macro feature bar.
4 Choose Create, type a name for the dialog, then choose OK.
or
Select the dialog you want, then choose Edit.

A file that contains only dialog definitions does not have to be compiled. The following example shows how to use Dialog Editor dialogs in a GroupWise macro:

```
APPLICATION(A1; "WPOffice"; Default; "US")
// For convenience, save the file that contains the dialog
// definitions under the same name as your macro
// with a .DLG extension.

// USE statement for including dialog definitions
USE("C:\WPOFFICE\MACROS\DLGTEST.DLG")

// Display dialog contained in definition file named "Dialog1"
DIALOGSHOW("Dialog1"; "WPOffice.GroupWise")

   ...statement block...

DIALOGDISMISS("Dialog1"; "CancelBttn")
QUIT
```

 For Example...


```
// Example Dialog Box
// Define dialog box
DIALOGDEFINE(Dialog: 1000; Left: 50; Top: 50; Width: 150; Height: 100; Style: OK! | Percent!;
Caption: "Example Dialog Box")

   //Static text control
   DIALOGADDTEXT(Dialog: 1000; Control: 100; Left: 10; Top: 10; Width: 50; Height: 15;
   Style: Left!; Text: "Edit control:")

   // Edit box control
   DIALOGADDEDITBOX(Dialog: 1000; Control: 101; Left: 10; Top: 25; Width: 125; Height: 25;
   Style: Left! | VScroll! | Multiline!; vReturn; 1000)

// Display dialog box
DIALOGSHOW("1000"; "WPOffice.GroupWise")
// Destroy dialog box
DIALOGDISMISS("1000"; "OKBttn")
```

Modal Dialog Boxes

Require the user to complete an action before input to the parent application is allowed. For example, the Busy Search Settings dialog box requires the user to choose OK or Cancel before control returns to GroupWise. Macro dialog boxes are modal by default (see DIALOGDEFINE).

Modeless Dialog Boxes

Do not require the user to complete an action before input to the parent application is allowed. For example, you can perform a spell check from the Spell Checker dialog box (choose Tools, Speller), then click a view and edit it while the Spell Checker dialog box remains on the screen. Click the dialog box to perform another spell check. For creating a modeless dialog box, see DIALOGDEFINE.

Recording a Dialog Box

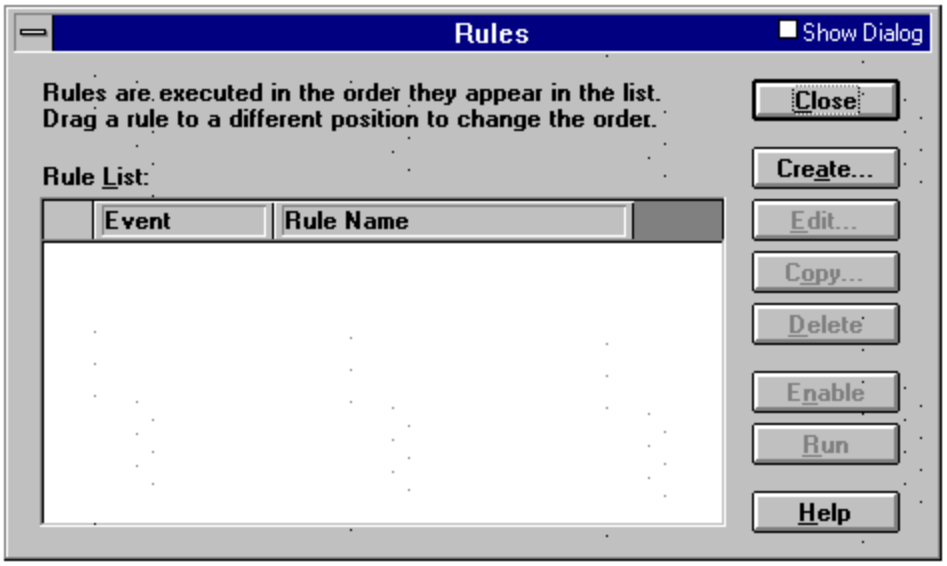
Steps

1 Choose Tools, Macro, Record (or press Ctrl+F10) to display the Record Macro dialog box.

2 Type a filename in the edit box. Type a path if different from the default macros directory. To insert a filename, select a name from the list box. See Record a Macro.

3 Choose Record to dismiss the dialog box. The insertion point changes to a circle with a slash to indicate that keystrokes and/or mouse actions are being recorded.

4 Open a dialog box. For example, choose Tools, Rules to open the Rules dialog box.



5 Select the Show Dialog check box in the upper right corner.

6 Close the dialog box.

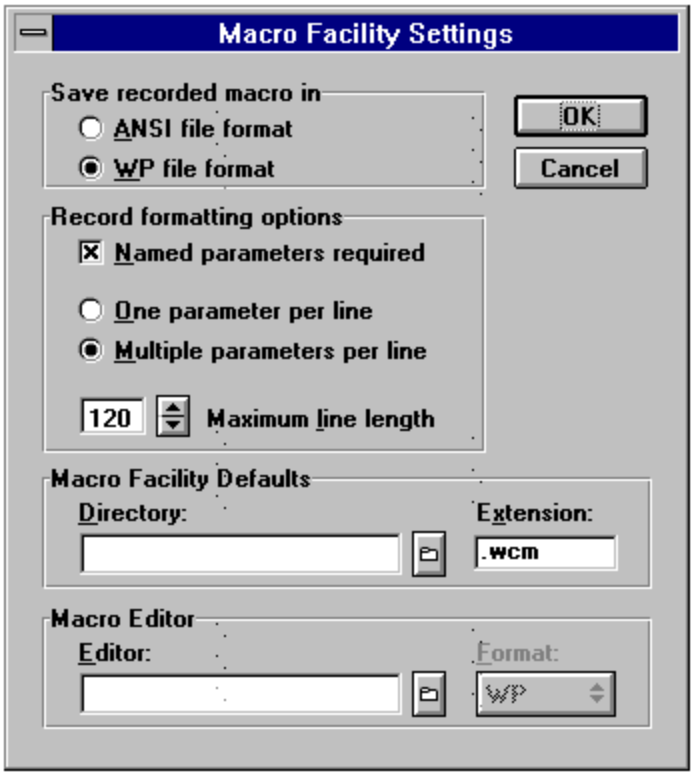
7 Choose Tools, Macro, Stop to stop recording. The macro is saved in the default macros directory unless you specified a different path (step 2).

Macro Facility

Macro Facility is an application that ships with Novell GroupWise 4.1, that records, plays, compiles, and converts macros created for Shared Code applications (Shared Code refers to programming code used by WordPerfect Windows applications).

Macro Facility starts automatically when you record or play a macro from a Shared Code application. If you play a macro for an application that is not running, Macro Facility automatically starts the application.

Set Macro Facility options, including the default macro directory, editor, and extension in the Macro Facility Settings dialog box. To display the dialog box, choose Settings, then Options:



See Also

Compilers
Command-Based Macro Language
Compiler Errors
Run-time Errors

Command-Based Macro Language

A command-based macro language records keystrokes and/or mouse actions as commands. It does not record the actual keystrokes. For example, to display the default Mail view, choose Send, New Mail. If you record these steps, your macro contains a product command named ViewOpen. Example:

```
APPLICATION (A1; "WPOffice"; Default; "US")
ViewOpen (Mail!; YES!)
```

Compiler Errors

The compiler stops and displays a message when it discovers a macro syntax error (see Syntax). The message contains information about the error and its location (the compiler makes a best guess and may not always be accurate). Choose Cancel Compilation to dismiss the message box, or Continue Compilation to test for other errors. You must correct all errors before a macro compiles. See Macro Errors for common syntax errors.

Run-time Errors

Run-time is the same as execution time. Run-time errors occur while the macro is executing (playing). Referencing a variable that has not been assigned a value causes a run-time error. See Macro Errors.

Assigning Macros

Assign a macro to a Button Bar.

Steps

1 Choose View, Button Bar Preferences (if grayed, choose Button Bar and try again).

2 Choose Create, enter a Button Bar name.

or ...

Select a Button Bar definition, then choose Edit.

3 Select Play a Macro, choose Add Macro, type the path and filename of the macro you want to add, then choose Select.

To play a Button Bar macro, choose the button the macro is assigned to.

Glossary

A

Active
ANSI Character Set
Application
Argument
ASCII Character Set

B

BIF (Binary Initialization File)
Bitmap
Button

C

Character Expression
Check Box
Choose

Click
Clipboard
Codes
Color Wheel
Command
Command Button
Compile
Conditional Statement
Control Menu
Control-Menu Box
Counter
CUA Keyboard
Current Cell, Current File, Current Table
Current Directory

D

DDE (Dynamic Data Exchange)
Default
Desktop
Dialog Box
Dialog Units
Directory
DOS
Double-Click
Drag
Driver

E

Edit Box
Enumerated Type
Extension

F

File
File Format
Filename

G

Group

I

INI File
Incrementing Button
Input Focus
Insertion Point

K

Kerning

L

Link
List Box
Loop

M

Macro
Macro Command
Measurement Expression
Memory
Menu
Message Box

N

Numeric Equivalent
Numeric Expression

O

OLE (Object Linking and Embedding)
Operator

P

Parameter
Path
Pixel
Point Size
Popup List
Port
Printer Driver
Product Prefix
Prompt

R

Radio Button
Relational Expression
Reveal Codes
Root Directory

Run-Time

S

Scroll
Scroll Bar
Select
Selection Cursor
Sizing Handle
Shared Code
Sort Key
Spreadsheet Import
Spreadsheet Link
Status Bar
Syntax

T

Text Box (Dialog Box)
Text Box (Graphics)
Text File
Title Bar
Toggle

U

User-defined Dialog Box

V

Value Set Member
Variable

W

Wildcard
Window
Window Handle

Active

An application, window, or dialog box that is currently in use. Windows highlights the caption bar or dialog frame.

ANSI Character Set
The 256 characters of the American National Standards Institute. Used by WordPerfect for Windows.

Application
A software program, such as a word processing or spreadsheet program.

Argument
A variable, constant, or expression required by a command or function.

ASCII Character Set

The American Standard Code for Information Interchange is one of the standard formats for representing characters. The ASCII character set contains 128 characters, which are the first 128 characters of the ANSI character set.

BIF (Binary Initialization File)
A file that contains run-time options for a WordPerfect application. BIF replaces the standard text initialization files (INI) commonly used by Windows applications.

Bitmap
A graphics file format that represents images as a pattern of pixels (dots).

Button
A graphic representation of a command or option.

Character Expression

One or more characters enclosed in quotation marks, which identifies the characters as text, not a variable.

Check Box

A graphic representation where the user turns an option on or off by selecting or deselecting the box. An X in the box indicates the option is on.

Choose
To click a menu item or dialog box option that initiates an immediate action. See Select.

Click
To press and release the mouse button.

Clipboard
An area of memory, also called a buffer, where text and commands can be stored for further action. Clipboard contents are erased when you exit Windows.

Codes

Formatting commands that specify computer and printer functions. Reveal Codes displays codes.

Color Wheel
A dialog box graphic where the user selects a color.

Command
An instruction given to a computer.

Command Button

A button in a dialog box which carries out an action OK, Exit, or Cancel. A thick, dark border indicates the default button.

Compile

To translate program code into machine language and check for syntax errors. The program automatically compiles macros the first time they are played. Each time a macro is edited and saved, it must be recompiled before it can be played.

Conditional Statement

A statement created with macro programming commands such as IF and WHILE. A conditional statement contains expressions such as text, numbers, or variables to be evaluated. Macro execution can be affected by the "condition" of these expressions at the time they are evaluated.

Control Menu

The menu to open, close, maximize, minimize, or restore a window or dialog box. To display a control menu, click the control-menu box or press Alt+Spacebar

Control-Menu Box

A small rectangular button in the upper left corner of a window or dialog box. Click a control-menu box to display the control menu; double-click to close the window or dialog box. Several windows can be open at the same time, each with its own control-menu box.

Counter
A dialog box element where the user specifies values with incrementer and decrementer buttons.

CUA Keyboard

Common User Access (CUA) standard keyboard. The WordPerfect for Windows default keyboard is CUA compliant.

Current Cell, Current File, Current Table

The cell, document, or table where the insertion point is.

Current Directory
The directory where a file will be saved if you do not specify another.

DDE (Dynamic Data Exchange)

A connection between a WordPerfect document and a file created by another Windows application that supports DDE. You can transfer information from one file to the other through the link. If both files are open, transferred information can be updated automatically when information changes in the source file.

Default

A setting, value, or response provided by the application unless an alternative is specified. You can change some WordPerfect defaults for all future documents in Preferences. Other options change a default for the active document only.

Desktop
The screen background and environment for all Windows applications.

Dialog Box

A window that displays warnings and messages, and options for the user to select. Dialog boxes have a title bar and a control menu, but not a menu bar. They can be moved to different positions on the screen. Most dialog boxes must be closed before you can work in the document window, but a few (such as Edit Button Bar) allow you to move between the window and the dialog box.

Dialog Units

A vertical unit that equals 1/8 the font height, or a horizontal unit that equals 1/4 the font width. Dialog boxes are positioned and sized in dialog units.

Directory

A group of files or directories under one identifying title.

The *root* directory is the directory on a disk drive to which all other directories are related.

A *default* directory is where files are saved to or opened and retrieved from if you do not specify a pathname. Not all features in WordPerfect have the same default directory; many defaults can be customized.

A *directory* tree is a diagram of how directories and subdirectories relate to each other.

DOS

Disk Operating System. Software that directs the flow of data between disk drives and your computer. An application needs an operating system to function.

Double-Click
To click the mouse button twice in rapid succession.

Drag
Press and hold down the left mouse button while moving the mouse.

Driver
A set of commands that runs a peripheral device such as a printer or monitor.

Edit Box
An element in a dialog box where a user can type text.

Enumerated Type

The parameters for some product commands accept only certain predefined words known as "enumerated types." Enumerated types end with exclamation points and have numeric equivalents. For example, DisplayMode parameter accepts only Text!, Graphics!, or FullPage!.

Extension
The optional three-letter identifier that may be added to any filename.

File
A document or other type of information saved under an identifying name.

File Format

The pattern in which a file is organized. Each application creates documents in a unique format not accessible by other applications. Those source documents must be converted to the new application’s format.

Filename

The name given to a file on disk. The name may be up to eight characters long, with an optional three letter extension, such as MYLETTER.ANN.

Group
A set of related options in a dialog box, often with its own subtitle.

INI File
A file that stores setup and startup information for Windows applications.

Incrementing Button
A button that lets you specify an amount by clicking the mouse instead of typing numbers.

Input Focus

The window or control receiving the keyboard or mouse action has input focus. Controls include radio buttons, push buttons, and edit boxes.

Insertion Point

The blinking vertical bar in the current document window or dialog box. It indicates where text will appear when you type.

Kerning
Increasing or decreasing space between a pair of characters.

Link

See DDE and Spreadsheet Link.

List Box
A dialog control that displays a list of mutually-exclusive items.

Loop
A set of programming commands that repeat.

Macro
A series of commands and keystrokes written and/or recorded in its own file to be compiled, then replayed as needed.

Macro Command

An instruction for WordPerfect to carry out. The instruction can be in the form of a programming command or a product command. Many macro commands must be used together with other macro commands to be successful in giving a complete instruction.

Measurement Expression
A number followed by a unit of measure (", i, c, m, p, w).

Memory
The temporary data storage area for a computer or printer.

Menu
A list of commands that can be applied to the active window or application.

Message Box

A dialog box that displays information, a warning, or error message, or asks for confirmation before completing a command.

Numeric Equivalent
The number that represents an enumerated type.

Numeric Expression

A number on which mathematical operations may be performed. Numeric expressions are not enclosed in quotation marks.

OLE (Object Linking and Embedding)
A feature that copies information from one document to another (embedding), through a "live" link. When the original document changes, the embedded copy reflects the changes.

Operator

A symbol or word that performs a function on one or more expressions. Operators compare expressions, link words together, and perform mathematical functions.

Parameter
Contains data to pass to the compiler or pass between subroutines.

Path

The location of a certain file or directory in a computer disk drive or on a network. A path includes the drive, the root directory, and any subdirectory names. Each name is separated by a backslash (\). For example, C:\WPWIN60\LETTER refers to the LETTER subdirectory in the WPWIN60 directory on the C drive.

Pixel
The smallest unit of display on a computer screen.

Point Size
The unit of measure for font sizes. One point equals 1/72".

Popup List

A list of options displayed when a popup button is clicked. Most popup buttons are marked by double triangles and display mutually exclusive options. The button itself shows the selected option. Other popup lists, marked by single triangles, show the feature name rather than the selected option. On the ruler bar, popup list buttons have no triangular markings. Some display the selected option and others display the feature name.

Port
A connection device between a computer and another component such as a printer or modem.

Printer Driver

Software commands that enable an application to communicate with a printer. WordPerfect can use WordPerfect printer drivers and Windows printer drivers. WordPerfect printer drivers are identified by a .PRS extension.

Product Prefix
A two-character expression that specifies a product for a macro command.

Prompt

A message box that displays information for the user.

Radio Button
Represents one item in a list of mutually exclusive options.

Relational Expression
An expression that evaluates parameters with only two possible states: TRUE and FALSE.

Reveal Codes
A feature that displays text and formatting codes in the document window.

Root Directory
See Directory.

Run-Time

Execution time. Run-time errors occur during macro execution. Run-time options are application start-up settings, such as the macros default directory. See BIF.

Scroll

To move through a document or list box by using the scroll bar.

Scroll Bar

The vertical or horizontal bar in a document and in some list boxes that allows you to move through a document or list by clicking the scroll arrows or dragging the scroll box (thumb).

Select

To identify a file, directory, graphics box, or area of text that will be affected by subsequent choices; to identify a dialog box option to be applied to a file, directory, graphics box, or area of text.

Selection Cursor
The highlighted text, dotted rectangle, or insertion point that indicates where the next keystroke or mouse action will apply in a dialog box.

Sizing Handle

The small solid squares that appear on the borders of a selected graphics box. Drag handles to size the box and its contents.

Shared Code
Programming code used by WordPerfect Windows applications.

Sort Key
Words, fields, or phrases that define sort criteria.

Spreadsheet Import

Copies information from a spreadsheet file into a WordPerfect document one time only. You cannot update a spreadsheet import like you can update a spreadsheet link.

Spreadsheet Link

A connection to transfer information between a spreadsheet file and a WordPerfect document. You can update transferred information to reflect changes made in the spreadsheet file. See DDE.

Status Bar

The line at the bottom of the WordPerfect window that lists the current font and the position of the insertion point. The status bar also displays some WordPerfect messages and features.

Syntax

The order in which parts of a command are organized. Macro commands require a specific syntax to function properly.

Text Box (Dialog Box)
The area in a dialog box where you type text or where WordPerfect types text for you. The insertion point must be in the text box before you can begin to type.

Text Box (Graphics)
A type of graphics box for placing and rotating text.

Text File

A file saved in ASCII (DOS) or ANSI (Windows) file format. It contains text, spaces, and returns, but no formatting codes.

Title Bar
The area of a window or dialog box that displays its title.

Toggle

To switch between on or off; or, a feature or option that is turned on or off with the same keystroke or command. Toggle commands usually do not require parameters.

User-defined Dialog Box
A custom dialog box created with DIALOG programming commands that display options for user input.

Value Set Member

Words specified for acceptance by certain parameters. Each value set member ends with an exclamation point (!). Every value set member has a numeric equivalent.

Variable
One or more characters that represent data that can change during macro execution.

Wildcard

Characters that represent variables in a word, file, or directory search. A question mark (?) represents a single character. An asterisk (*) represents zero or more characters in succession.

Window

Each window has a title bar, a menu bar, and a status bar, and may contain a scroll bar, Button Bar, and Ruler. The Equation Editor window has separate editing and display areas called *panes*.

Window Handle
A unique identifier for a window or control.

Coach Overview

A Coach takes the user through some of the product's most common tasks step-by-step, providing instructions and prompts as needed. The Coach can either show the user how to perform the desired task or it can tell the user what to do and then monitor the user's actions to make sure no mistakes are made.

The Coach Support Routines allow Coaches to be written with macros for products that use Shared Code 2.x.

The support routines provide three types of functions:

Prompting

CoachMessageBox displays options for the user to select. CoachPrompt displays helps and hints.

Animating

CoachAnimate shows the user how to perform certain actions. The Coach can: 1) move the mouse pointer, 2) send mouse clicks, and 3) send keystrokes to the application.

Filtering

CoachFilterAdd defines a filter to monitor mouse movements, keystrokes, and tokens. CoachFilterEnable activates the filter to compare defined events with current events. For example, a filter may wait for a left mouse click in the OK button of a particular dialog, then move to the next step. If the user makes a mistake, the Coach might display a message box or prompt.

CoachFilterDisable deactivates a filter and CoachFilterDestroy destroys a filter.

There are three types of filters 1) mouse, 2) keyboard, and 3) command. Mouse and keyboard filters are *interface filters*.

Named Region

A named region is a specific window or item on the screen, such as "GroupWise.EditWindow" and "GroupWise.OpenDlg.OpenBttn". See Named Region in MOUSE STRING.

Writing the Coach

A Coach generally follows these steps:

1 Pause keyboard and mouse input until a Coach enables it.

2 Define filter.

3 Instruct the user how to perform a task with prompts, message boxes, and animation.

4 Enable filter.

5 Wait for user input.

6 Respond to user input with a callback procedure.

7 Disable filter.

8 Destroy filter.

9 Turn on keyboard and mouse input.

Define all filters at the beginning of a Coach macro, then enable each one as it is needed. Repeat steps 2 through 8 as many times as necessary.

Quitting the Coach

Press Alt+Esc to quit the Coach.

Coach Dialog Box

Coaches have a .WCH extension and are stored in the macro directory.

When the user chooses Coach from the Help menu, the descriptive names of all .WCH files in the macros directory are displayed in the Coach dialog box. When a Coach name is selected, the information from the Abstract field of the document summary is displayed at the bottom of the Coach dialog box. If the Coach does not have a descriptive name, the filename is displayed.

DDE Conversations

Use one of the following Application Name/Topic pairs to establish a DDE conversation with GroupWise 4.1 for Windows. The first pair is recommended. The other pairs are for backward compatibility with WordPerfect Office 4.0 and 4.0a. See DDEINITIATE.

After establishing a DDE conversation, you can send any supported command to GroupWise, including commands that return information to the DDE client.

Application Name	Topic
"GroupWise"	"Command"
"OFWin"	"Command"
"WPOffice40"	"Command"
"WPOffice40"	"GetOfficeData"
"WPOffice40"	"GetAddressBookData"

Handles and Message IDs

AddressBookSearchBegin returns an AddressBook handle.

AddressListCreate and AddressListCreateFromGroup return an AddressList handle.

FilterCreate returns a Filter handle.

FolderListCreate and FolderListCreateFromView return a FolderList handle.

ItemMessageIDFromView and ItemGetReplyToMessageID return message IDs.

AboutDlg

Purpose

Displays the About Novell GroupWise 4.1 dialog box, which contains product, network, and Post Office information.

Syntax

AboutDlg ()

Route...
Choose Help, About GroupWise.

AddressBookCompare

Purpose
Returns True if two address strings point to the same person or resource, False if not. Not recordable.

Syntax
boolean AddressBookCompare (Address1: *string*; Address2: *string*)

Return Value
boolean True/False.

Parameters
Address1: *string*
First address string.

Address2: *string*
Second address string.

See Also
AddressBookResolve

AddressBookDeletePersonalGroup

Purpose

Deletes a personal group from the Address Book. Not recordable.

Syntax

AddressBookDeletePersonalGroup (GroupName: *string*; UserID: *string*)

Parameters

GroupName: *string*
Name of a personal group.

UserID: *string* (optional)
User ID of the mail box that contains personal group Address Book information. Use this parameter to delete a personal group from a proxy mailbox.

AddressBookDlg

Purpose

Displays the Address Book dialog box, which contains user names, user IDs, groups, and resources.

Syntax

AddressBookDlg ()

See Also

AddressListEdit

Route...
Choose Send, Address Book.

AddressBookGetEntry

Purpose
Returns the contents of an Address Book field. Not recordable.

Syntax
string AddressBookGetEntry (ABField: *enum*; FullAddressText: *string*; UserID: *string*)

Return Value
string For example, UsersFirstName! may return Bob, John, or Mary.

Parameters
ABField: *enum*
Address Book field. The entry returns in a variable. If a field does not exist in a specified address section (see AddressListIsInSection), a run-time error occurs.

- PersonalGroupsID!
- PublicGroupsDescription!
- PublicGroupsDomain!
- PublicGroupsHost!
- PublicGroupsID!
- ResourcesDescription!
- ResourcesDomain!
- ResourcesFID!
- ResourcesHost!
- ResourcesID!
- ResourcesOwner!
- ResourcesType!
- UsersAccountID!
- UsersDept!
- UsersDomain!
- UsersFID!
- UsersFaxNum!
- UsersFirstName!
- UsersHost!
- UsersID!
- UsersLastName!
- UsersNetID!
- UsersPhone!
- UsersTitle!
- UsersUD1!
- UsersUD10!
- UsersUD2!
- UsersUD3!
- UsersUD4!
- UsersUD5!
- UsersUD6!
- UsersUD7!
- UsersUD8!
- UsersUD9!

FullAddressText: *string*
The full address string of a person or resource. The command fails if FullAddressText is

not unique. Use AddressBookResolve (command) to determine if FullAddressText is unique.

UserID: *string* (optional)
User ID of the mail box that contains the Address Book information. Use this parameter to search in a proxy mailbox.

AddressBookGetField

Purpose
Returns the name of an Address Book field. Not recordable.

Syntax
string AddressBookGetField (ABField: *enum*)

Return Value
string For example, PublicGroupsDomain! returns the name Domain.

Parameters
ABField: *enum*
Address Book field.
- PersonalGroupsID!
- PublicGroupsDescription!
- PublicGroupsDomain!
- PublicGroupsHost!
- PublicGroupsID!
- ResourcesDescription!
- ResourcesDomain!
- ResourcesFID!
- ResourcesHost!
- ResourcesID!
- ResourcesOwner!
- ResourcesType!
- UsersAccountID!
- UsersDept!
- UsersDomain!
- UsersFID!
- UsersFaxNum!
- UsersFirstName!
- UsersHost!
- UsersID!
- UsersLastName!
- UsersNetID!
- UsersPhone!
- UsersTitle!
- UsersUD1!
- UsersUD10!
- UsersUD2!
- UsersUD3!
- UsersUD4!
- UsersUD5!
- UsersUD6!
- UsersUD7!
- UsersUD8!
- UsersUD9!

AddressBookGetFullName

Purpose
Returns the full name associated with a user ID in an Address Book. Not recordable.

Syntax
string AddressBookGetFullName (FullAddressText: *string*; UserID: *string*)

Return Value
string Full name.

Parameters
FullAddressText: *string*
User ID.

UserID: *string* (optional)
User ID of the mail box that contains the Address Book information. Use this parameter to search a proxy mailbox.

AddressBookIsGroupMember

Purpose
Determines if a personal group name is in another personal group. Not recordable.

Syntax
boolean AddressBookIsGroupMember (GroupName: *string*; UserID: *string*)

Return Value
boolean True if a group name is in another group, False if not.

Parameters
GroupName: *string*
Name of a personal group.

UserID: *string* (optional)
User ID of the mail box that contains the Address Book information. Use this parameter to search a proxy mailbox.

AddressBookResolve

Purpose
Returns a list of valid, unambiguous (unique) addresses. Not recordable.

Syntax
string AddressBookResolve (Addresses: *string*; RemoveAmbiguousAddresses: *enum*; RemoveInvalidAddresses: *enum*; UserID: *string*)

Return Value
string Post office, domain, and user ID of valid and unambiguous addresses.

Parameters
Addresses: *string*
Separate addresses with commas and enclose in double quotation marks.

RemoveAmbiguousAddresses: *enum*
Fail!
Prompt!
Yes!

Fail! causes the command to fail if there are ambiguous addresses. Prompt! displays a dialog box where the user selects an address to include. Yes! removes all ambiguous addresses.

RemoveInvalidAddresses: *enum*
Fail!
Prompt!
Yes!

Fail! causes the command to fail if there are invalid addresses. Prompt! displays a dialog box where the user removes invalid addresses from the address list. Yes! removes all invalid addresses.

UserID: *string* (optional)
User ID of the mail box that contains the Address Book information. Use this parameter to search in a proxy mailbox.

AddressBookResolveFullName

Purpose
Returns the full address of a name or resource. Not recordable.

Syntax
string AddressBookResolveFullName (FullName: *string*; UserID: *string*)

Return Value
string Full address (Domain.PostOfice.ID).

Parameters
FullName: *string*
User or resource.

UserID: *string* (optional)
User ID of the mail box that contains the Address Book information. Use this parameter to search in a proxy mailbox.

AddressBookSearch

Purpose
Searches for a character string in a specified Address Book field. Not recordable.

Syntax
string AddressBookSearch (Handle: *numeric*; ABField: *enum*; SearchString: *string*; SearchDir: *enum*; Match: *enum*)

Return Value
string First address (user ID, resource ID, or group name) to match the search criteria.

Parameters
Handle: *numeric*
Address Book search handle, returned by AddressBookSearchBegin
.

ABField: *enum*
Address Book field. If search is successful, the field's contents are returned. If a field does not exist in specified address section (see AddressListIsInSection), a run-time error occurs.

PersonalGroupsID!
PublicGroupsDescription!
PublicGroupsDomain!
PublicGroupsHost!
PublicGroupsID!
ResourcesDescription!
ResourcesDomain!
ResourcesFID!
ResourcesHost!
ResourcesID!
ResourcesOwner!
ResourcesType!
UsersAccountID!
UsersDept!
UsersDomain!
UsersFID!
UsersFaxNum!
UsersFirstName!
UsersHost!
UsersID!
UsersLastName!
UsersNetID!
UsersPhone!
UsersTitle!
UsersUD1!
UsersUD10!
UsersUD2!
UsersUD3!
UsersUD4!
UsersUD5!
UsersUD6!

UsersUD7!
UsersUD8!
UsersUD9!

SearchString: *string*

SearchDir: *enum*
Backward!
Forward!

Match: *enum*
SearchString in relation to the field contents.
BeginText!
FullText!
SubText!

FullText! requires an exact match.

AddressBookSearchBegin

Purpose
Returns an Address Book search handle. Not recordable.

Delete the search handle when it is no longer needed, or system resources will be lost. See AddressBookSearchEnd.

Syntax
numeric AddressBookSearchBegin (UserID: *string*)

Return Value
numeric Search handle if successful, 0 if not.

Parameters
UserID: *string*

AddressBookSearchEnd

Purpose
Deletes an Address Book search handle from memory. Not recordable.

Syntax
AddressBookSearchEnd (Handle: *numeric*)

Parameters
Handle: *numeric*
Address Book search handle.

See Also
AddressBookSearchBegin

AddressBookSearchNext

Purpose
Searches for the next occurrence of the character string specified by AddressBookSearch. Not recordable.

Syntax
string AddressBookSearchNext (Handle: *numeric*)

Return Value
string Next address (user ID, resource ID, or group name) to match the search criteria.

Parameters
Handle: *numeric*
Address Book search handle, returned by AddressBookSearchBegin.

AddressBookSearchPrevious

Purpose
Searches backward for the next occurrence of the character string specified by AddressBookSearch. Not recordable.

Syntax
string AddressBookSearchPrevious (Handle: *numeric*)

Return Value
string Previous address (user ID, resource ID, or group name) to match the search criteria.

Parameters
Handle: *numeric*
Address Book search handle, returned by AddressBookSearchBegin.

AddressInfoDlg

Purpose

Displays the Address Book information of a person or resource. Not recordable.

Syntax

AddressInfoDlg (FullAddressText: *string*; UserID: *string*)

Parameters

FullAddressText: *string*
Full address of a person or resource (Domain.PostOfice.UserID). Command fails if FullAddressText is not unique. AddressBookResolve determines if an address is unique.

UserID: *string*
User ID of the mail box that contains the Address Book information. Use this parameter to search in a proxy mailbox.

AddressListAdd

Purpose

Adds a user ID to To, CC, and BC address lists. Not recordable.

A run-time error occurs for incorrect or non-existent user IDs.

Syntax

AddressListAdd (Handle: *numeric*; ToList: *string*; CCList: *string*; BCList: *string*)

Parameters

Handle: *numeric*
Address list handle, returned by AddressListCreate or AddressListCreateFromGroup.

ToList: *string* (optional)
User ID.

CCList: *string* (optional)
User ID.

BCList: *string* (optional)
User ID.

AddressListCreate

Purpose
Creates an address list. Not recordable.


Delete the address list handle when it is no longer needed, or system resources will be lost. See AddressListDelete.

Syntax
numeric AddressListCreate (UserID: *string*)

Return Value
numeric Address list handle, or 0 if an error occurs.

Parameters
UserID: *string* (optional)
User ID of the mail box where the address list is being built. Use this parameter to create an address list in a proxy mailbox.

See Also
AddressListCreateFromGroup

AddressListCreateFromGroup

Purpose
Creates an address list from a public or personal group. Default: personal group. To specify a public group, use the full address string. Not recordable.

Delete the address list handle when it is no longer needed, or system resources will be lost. See AddressListDelete.

Syntax
numeric AddressListCreateFromGroup (GroupName: *string*; UserID: *string*)

Return Value
numeric Address list handle, or 0 if an error occurs.

Parameters
GroupName: *string*
Public or personal group name.

UserID: *string* (optional)
User ID of the mail box that contains the Address Book information. Use this parameter to search a proxy mailbox.

See Also
AddressListCreate
AddressListSaveAsPersonalGroup

AddressListDelete

Purpose
Deletes an address list handle from memory. Not recordable.


Delete the address list handle when it is no longer needed, or system resources will be lost.

Syntax
AddressListDelete (Handle: *numeric*)

Parameters
Handle: *numeric*
Address list handle.

See Also
AddressListCreate
AddressListCreateFromGroup

AddressListEdit

Purpose

Displays the Address Book dialog box for editing an address list. Not recordable.

The Address Book dialog box is initialized with the contents of the address list. Changes to the TO, CC, and BC dialog lists replace the corresponding contents of the address list.

Syntax

AddressListEdit (Handle: *numeric*)

Parameters

Handle: *numeric*
Address list handle, returned by AddressListCreate or AddressListCreateFromGroup.

See Also

AddressBookDlg
AddressListDelete

AddressListGetAddress

Purpose
Returns the full address string of a specified address. Not recordable.

Syntax
string AddressListGetAddress (Handle: *numeric*; Index: *numeric*; FromWhichList: *enum*)

Return Value
string A user, resource, or group address (Domain.PostOffice.ID).

Parameters
Handle: *numeric*
Address list handle, returned by AddressListCreate or AddressListCreateFromGroup.

Index: *numeric*
Address index. The first address is 0, the second is 1, and so forth. If the index is out of range, a run-time error occurs (see AddressListGetCount).

FromWhichList: *enum*
Address list to apply Index to.
BCList!
CCList!
Combined!
ToList!

Combined! concatenates TO, CC, and BC in that order.

See Also
AddressListDelete
AddressListGetAddress

AddressListGetCount

Purpose
Returns the number of addresses in a list. Not recordable.

Syntax
numeric AddressListGetCount (Handle: *numeric*; FromWhichList: *enum*)

Return Value
numeric Number of addresses.

Parameters
Handle: *numeric*
Address list handle, returned by AddressListCreate or AddressListCreateFromGroup.

FromWhichList: *enum*
Address list to apply Index to.
BCList!
CCList!
Combined!
ToList!

Combined! concatenates TO, CC, and BC in that order.

AddressListGetEntry

Purpose
Returns the contents of an address list entry. Not recordable.

Syntax
string AddressListGetEntry (Handle: *numeric*; Index: *numeric*; ABField: *enum*; FromWhichList: *enum*)

Return Value
string Address list entry.

Parameters
Handle: *numeric*
Address list handle, returned by AddressListCreate or AddressListCreateFromGroup.

Index: *numeric*
Address index. The first address is 0, the second is 1, and so forth. If the index is out of range, a run-time error occurs.

ABField: *enum*
Address Book field. The field's entry returns in a variable. If a field does not exist in a section of the specified address (see AddressListIsInSection), a run-time error occurs.

PersonalGroupsID!
PublicGroupsDescription!
PublicGroupsDomain!
PublicGroupsHost!
PublicGroupsID!
ResourcesDescription!
ResourcesDomain!
ResourcesFID!
ResourcesHost!
ResourcesID!
ResourcesOwner!
ResourcesType!
UsersAccountID!
UsersDept!
UsersDomain!
UsersFID!
UsersFaxNum!
UsersFirstName!
UsersHost!
UsersID!
UsersLastName!
UsersNetID!
UsersPhone!
UsersTitle!
UsersUD1!
UsersUD10!
UsersUD2!
UsersUD3!
UsersUD4!

UsersUD5!
UsersUD6!
UsersUD7!
UsersUD8!
UsersUD9!

UsersUD*x* are defined by the System Administrator.

FromWhichList: *enum*
Address list to apply Index to.
BCList!
CCList!
Combined!
ToList!

Combined! concatenates TO, CC, and BC in that order.

See Also
AddressListDelete

AddressListGetField

Purpose
Returns an address list field name. Not recordable.

Syntax
string AddressListGetField (Handle: *numeric*; Index: *numeric*; ABField: *enum*; FromWhichList: *enum*)

Return Value
string Returns an Address Book field name (see ABField parameter).

Parameters
Handle: *numeric*
Address list handle, returned by AddressListCreate or AddressListCreateFromGroup.

Index: *numeric*
Address index. The first address is 0, the second is 1, and so forth. If the index is out of range, a run-time error occurs.

ABField: *enum*
Address Book field. The field's name returns in a variable. If a field does not exist in the section of a specified address (see AddressListIsInSection), a run-time error occurs.
- PersonalGroupsID!
- PublicGroupsDescription!
- PublicGroupsDomain!
- PublicGroupsHost!
- PublicGroupsID!
- ResourcesDescription!
- ResourcesDomain!
- ResourcesFID!
- ResourcesHost!
- ResourcesID!
- ResourcesOwner!
- ResourcesType!
- UsersAccountID!
- UsersDept!
- UsersDomain!
- UsersFID!
- UsersFaxNum!
- UsersFirstName!
- UsersHost!
- UsersID!
- UsersLastName!
- UsersNetID!
- UsersPhone!
- UsersTitle!
- UsersUD1!
- UsersUD10!
- UsersUD2!
- UsersUD3!
- UsersUD4!

UsersUD5!
UsersUD6!
UsersUD7!
UsersUD8!
UsersUD9!

UsersUD*x* are defined by the System Administrator.

FromWhichList: *enum*
Address list to apply Index to.
BCList!
CCList!
Combined!
ToList!

Combined! concatenates TO, CC, and BC in that order.

AddressListGetFullName

Purpose
Returns the full name associated with a user ID in an address list. Not recordable.

Syntax
string AddressListGetFullName (Handle: *numeric*; Index: *numeric*; FromWhichList: *enum*)

Return Value
string Full name.

Parameters
Handle: *numeric*
Address list handle, returned by AddressListCreate or AddressListCreateFromGroup.

Index: *numeric*
Address index. The first address is 0, the second is 1, and so forth. If the index is out of range, a run-time error occurs.

FromWhichList: *enum*
Address list to apply Index to.
BCList!
CCList!
Combined!
ToList!

Combined! concatenates TO, CC, and BC in that order.

AddressListIsInSection

Purpose
Determines whether an address is in a specified address section. Not recordable.

Syntax
boolean AddressListIsInSection (Handle: *numeric*; Index: *numeric*; Section: *enum*; FromWhichList: *enum*)

Return Value
boolean Returns True if an address is in a specified section, False if not (see Section parameter).

Parameters
Handle: *numeric*
Address list handle, returned by AddressListCreate or AddressListCreateFromGroup.

Index: *numeric*
Address index. The first address is 0, the second is 1, and so forth. If the index is out of range, a run-time error occurs.

Section: *enum*
The section to search in.
- ExternalAddress!
- PersonalGroups!
- PublicGroups!
- Resources!
- Users!

ExternalAddress! is for other mail systems.

FromWhichList: *enum*
Address list to apply Index to.
- BCList!
- CCList!
- Combined!
- ToList!

Combined! concatenates TO, CC, and BC in that order.

AddressListSaveAsPersonalGroup

Purpose
Creates a personal group from an address list. Not recordable.

Syntax
AddressListSaveAsPersonalGroup (Handle: *numeric*; GroupName: *string*)

Parameters
Handle: *numeric*
Address list handle, returned by AddressListCreate or AddressListCreateFromGroup.

GroupName: *string*
Name of personal group.

See Also
AddressBookDeletePersonalGroup
AddressListAdd
AddressListDelete

AlarmSet

Purpose

Sets an alarm that notifies a user of an upcoming appointment and, optionally, launches a program. Not recordable.

The appointment must have the input focus and must be future.

Syntax

AlarmSet (HoursBefore: *numeric*; MinutesBefore: *numeric*; ProgramToLaunch: *string*)

Parameters

HoursBefore: *numeric*

MinutesBefore: *numeric*

ProgramToLaunch: *string* (optional)

Route...
Select or open a future appointment, choose Actions, Set Alarm, select options.

AlarmSetDlg

Purpose

Displays the Set Alarm dialog box. The appointment must have the input focus and must be future.

Syntax

AlarmSetDlg ()

Route...
Select or open an appointment, choose Actions, Set Alarm.

AppAllowClose

Purpose

Specifies whether a user can exit GroupWise. This command is normally called by a DDE client or cross-application macro. Not recordable.

GroupWise cannot be closed if a macro is playing or if AppAllowClose is No! (see Allow parameter).

Syntax

AppAllowClose (Allow: *enum*)

Parameters

Allow: *enum*

No!
Yes!

If Allow is No!, you must set it to Yes! before ending the DDE conversation or cross-application macro. Otherwise, you can exit GroupWise only through the Windows Task List.

AttachmentAdd

Purpose
Attaches a file to an item.

Syntax
AttachmentAdd (FileName: *string*; DeleteWhenDone: *enum*; AttachmentDisplayName: *string*)

Parameters
FileName: *string*

DeleteWhenDone: *enum* (optional)
Not recordable.
No!
Yes!

AttachmentDisplayName: *string* (optional)
Named displayed in the Attach List box. Not recordable.

See Also
AttachmentDelete
AttachmentOpen
AttachmentView

Route...
Open an item view, choose File, Attach File, select a file.

AttachmentAddDlg

Purpose

Displays the Add Attachment dialog box to attach a file to an item.

Syntax

AttachmentAddDlg ()

See Also

AttachmentAdd
AttachmentListDlg

Route...
Open an item view, choose File, Attach File.

AttachmentDelete

Purpose
Deletes an attachment before the item is sent.

Syntax
AttachmentDelete (AttachmentIndex: *numeric*)

Parameters
AttachmentIndex: *numeric*
Index number (zero-based).

See Also
AttachmentAdd
AttachmentOpen
AttachmentView

Route...
Open an item view that has attachments, choose File, Attachments, select a file, Delete.

AttachmentListDlg

Purpose

Displays the Attachments dialog box to add attachments to a new item.

Syntax

AttachmentListDlg ()

See Also

AttachmentAdd
AttachmentAddDlg
AttachmentSaveAs

Route...
Open an item view, choose File, Attachments.

AttachmentOpen

Purpose
Opens an attachment.

Syntax
AttachmentOpen (AttachmentIndex: *numeric*)

Parameters
AttachmentIndex: *numeric*
Index number (zero-based).

See Also
AttachmentAdd
AttachmentAddDlg
AttachmentSaveAs

Route...
Open an item view that has attachments, choose File, Attachments, select a file, Open.

AttachmentSaveAs

Purpose
Saves an attachment with a new filename. Not recordable.

Syntax
AttachmentSaveAs (NewFilename: *string*; AttachmentIndex: *numeric*)

Parameters
NewFilename: *string*

AttachmentIndex: *numeric*
Index number (zero-based).

AttachmentSaveAsDlg

Purpose

Displays the Save As dialog box to save an attachment with a new filename.

Syntax

AttachmentSaveAsDlg ()

Route...
Open an item view, choose File, Attachments, select an attachment, View, File, Save As.

AttachmentView

Purpose
Displays an attachment.

Syntax
AttachmentView (TypeToView: *enum*; AttachmentIndex: *numeric*)

Parameters
TypeToView: *enum*
Attachment!
Message!

AttachmentIndex: *numeric* (optional)
Index number (zero-based).

See Also
AttachmentAdd
AttachmentDelete
AttachmentOpen
AttachmentSaveAs

Route...
Open an item, choose File, View Attachment.

or

Choose File, Attachments, select an attachment, View.

BusySearch

Purpose

Searches one or more user's schedules, and displays the result in the Choose Appointment Time dialog box.

Enclose multiple IDs in double quotation marks, separated by commas (see TO, CC, and BC parameters).

Syntax

BusySearch (To: *string*; CC: *string*; BC: *string*; StartDay: *numeric*; StartMonth: *numeric*; StartYear: *numeric*; StartMinute: *numeric*; StartHour: *numeric*; DurationMinutes: *numeric*; DurationHours: *numeric*; AppointmentInformation: *enum*; NumberOfDays: *numeric*; Sunday: *enum*; Monday: *enum*; Tuesday: *enum*; Wednesday: *enum*; Thursday: *enum*; Friday: *enum*; Saturday: *enum*; DisplayBeginMinute: *numeric*; DisplayBeginHour: *numeric*; DisplayEndMinute: *numeric*; DisplayEndHour: *numeric*)

Parameters

To: *string*
User IDs to include in search.

CC: *string* (optional)
User IDs to receive a carbon copy.

BC: *string* (optional)
User IDs to receive a blind copy.

StartDay: *numeric* (optional)
Appointment day.

StartMonth: *numeric* (optional)
Appointment month.

StartYear: *numeric* (optional)
Appointment year.

StartMinute: *numeric* (optional)
Minute to start appointment.

StartHour: *numeric* (optional)

DurationMinutes: *numeric* (optional)

DurationHours: *numeric* (optional)

AppointmentInformation: *enum* (optional)
Show appointment information such as From, Place, and Subject, depending on access rights.

Hide!
Show!

NumberOfDays: *numeric* (optional)
Number of days to search.

Sunday: *enum* (optional)
Include Sunday in search.
Yes!
No!

Monday: *enum* (optional)
Yes!
No!

Tuesday: *enum* (optional)
Yes!
No!

Wednesday: *enum* (optional)
Yes!
No!

Thursday: *enum* (optional)
Yes!
No!

Friday: *enum* (optional)
Yes!
No!

Saturday: *enum* (optional)
Yes!
No!

DisplayBeginMinute: *numeric*
Minute to start display on.

DisplayBeginHour: *numeric*
Hour to start display on.

DisplayEndMinute: *numeric*
Minute to end display on.

DisplayEndHour: *numeric*
Hour to end display on.

See Also
BusySearch

Route...
Open an Appointment view, choose Send, Busy Search, select options.

BusySearchDlg

Purpose

Displays the Busy Search Settings dialog box to search one or more user's schedules.

Syntax

BusySearchDlg ()

Route...
From the Main Window, or an In Box, Out Box, or Calendar view, choose Send, Busy Search.

ButtonBarCopy

Purpose
Copies a Button Bar definition to a Button Bar file. Not recordable.

Syntax
ButtonBarCopy (SrcFilename: *string*; DestFilename: *string*; BtnBarName: *string*)

Parameters
SrcFilename: *string*
Source file.

DestFilename: *string*
Destination file.

BtnBarName: *string*
Button Bar definition.

ButtonBarOptions

Purpose

Specifies Button Bar style and location (backward compatibility only). Use ButtonBarSetLocation and ButtonBarSetStyle. Not recordable.

Syntax

ButtonBarOptions (Style: *enum*; Location: *enum*)

Parameters

Style: *enum*

- PictureAndText!
- PictureOnly!
- TextOnly!

Location: *enum*

- BBarOnBottom!
- BBarOnLeft!
- BBarOnPalette!
- BBarOnRight!
- BBarOnTop!

ButtonBarOptionsDlg

Purpose

Displays the Button Bar Options dialog box. This command exists for backward compatibility. Use ButtonBarPreferences. Not recordable.

Syntax

ButtonBarOptionsDlg ()

ButtonBarPreferences

Purpose

Displays the Button Bar Preferences dialog box to select, create, delete, or edit a Button Bar. Not recordable.

Syntax

ButtonBarPreferences ()

ButtonBarSave

Purpose

Saves the current Button Bar. This command exists for backward compatibility. Use ButtonBarPreferences. Not recordable.

Syntax

ButtonBarSave (Filename: *string*)

Parameters

Filename: *string*
Button Bar name.

ButtonBarSelect

Purpose
Selects a Button Bar.

Syntax
ButtonBarSelect (Filename: *string*)

Parameters
Filename: *string*
Button Bar name. All Button Bar definitions are saved in one file.

Route...
Choose View, Button Bar Preferences, select a Button Bar, choose Select.

ButtonBarSelectDlg

Purpose

Displays the Button Bar Preferences dialog box. This command exists for backward compatibility. Use ButtonBarPreferences. Not recordable.

Syntax

ButtonBarSelectDlg ()

ButtonBarSetFont

Purpose
Selects the Button Bar font and font size.

Syntax
ButtonBarSetFont (FontSize: *numeric*; FontName: *string*; CharSet: *numeric*)

Parameters
FontSize: *numeric*

FontName: *string*

CharSet: *numeric*
Used in Asian versions of GroupWise.

Route...
Choose View, Button Bar Preferences, Options, select a font and font size.

ButtonBarSetLocation

Purpose
Sets the Button Bar location.

Syntax
ButtonBarSetLocation (Location: *enum*)

Parameters
Location: *enum*
BBarOnBottom!
BBarOnLeft!
BBarOnPalette!
BBarOnRight!
BBarOnTop!

Route...
Choose View, Button Bar Preferences, Options, select location.

ButtonBarSetRows

Purpose
Specifies the maximum number of rows or columns to display on the Button Bar.

Syntax
ButtonBarSetRows (Rows: *numeric*)

Parameters
Rows: *numeric*
Maximum number of rows or columns.

Route...
Choose View, Button Bar Preferences, Options, specify the number of rows or columns.

ButtonBarSetStyle

Purpose
Specifies a Button Bar style.

Syntax
ButtonBarSetStyle (Style: *enum*)

Parameters
Style: *enum*
PictureAndText!
PictureOnly!
TextOnly!

Route...
Choose View, Button Bar Preferences, Options, select a style.

ButtonBarShow

Purpose
Turns the Button Bar on or off.

Syntax
ButtonBarShow (State: *enum*)

Parameters
State: *enum* (optional)
If State not specified, acts as a toggle.
Off!
On!

Route...
Choose View, Button Bar.

CheckboxSelect

Purpose
Selects or deselects a check box in a Phone message view.

Syntax
CheckboxSelect (StandardCheckBox: *enum*; UserDefinedBoxName: *string*; State: *enum*)

Parameters
StandardCheckBox: *enum* (optional)
Phone message check box.
- PhoneCalled!
- PhoneCameToSeeYou!
- PhonePleaseCall!
- PhoneReturnedYourCall!
- PhoneUrgent!
- PhoneWantsToSeeYou!
- PhoneWillCallAgain!

UserDefinedBoxName: *string* (optional)

State: *enum* (required)
- Off!
- On!

See Also
SendPhone

Route...
Select or deselect a check box, or press the Space Bar while the check box has the input focus.

CloseWindow

Purpose
Closes the current view.

Syntax
CloseWindow ()

See Also
MainWindowHide

Route...
Click the system menu box, choose Close.

CustomCommandDelete

Purpose
Deletes a custom command. Not recordable.

Syntax
CustomCommandDelete (CmdName: *string*)

Parameters
CmdName: *string*
Custom command name. See CustomCommandInstall.

CustomCommandExecute

Purpose

Executes a user-defined command, defined in the Binary Initialization File (BIF). See CustomCommandInstall.

Syntax

CustomCommandExecute (KeyName: *string*)

Parameters

KeyName: *string*
Custom command name.

Route...
Install GroupWise View Designer on the Tools menu (see CustomCommandInstall), choose Tools, View Designer.

CustomCommandInstall

Purpose

Installs a custom command as a GroupWise menu item. Exit and restart GroupWise to see the new item. Not recordable.

Syntax

CustomCommandInstall (CmdName: *string*; DLLName: *string*; CommandLine: *string*; OnMenu: *enum*; EventMap: *string*; MenuEnabled: *numeric*; MenuName: *string*; MenuItem: *string*; MenuPosition: *numeric*; IsMenuSeperator: *enum*; LongPrompt: *string*; HelpFile: *string*; HelpContext: *numeric*; HelpKeyword: *string*; BtnBarName: *string*; BtnBarFile: *string*; Description: *string*)

Parameters

CmdName: *string*
User-defined name, inserted in the BIF under WPOffice (group), Custom Commands (section).

DLLName: *string* (optional)
DLL that launches a command (see CommandLine parameter). Optional if MenuSeparator is TRUE. APPEXEC.DLL is a DLL that ships with GroupWise, or a custom DLL.

CommandLine: *string*
Command to install, such as an application (.EXE) or macro (.WCM). Macro example: "MFWIN20.EXE /M-" + *macropath* + *macroname*.

OnMenu: *enum* (optional)
Command appears on a menu.
FALSE
TRUE

EventMap: *string* (optional)
Array of characters (16) that specifies when a command is executed. Required if OnMenu is FALSE. Each element in the array has a value of 1 or 0. 1 calls the DLL (see DLLName parameter) when an event occurs, 0 does not. The events for array elements 0-2 are startup, exit, and new message arrival (in that order). Elements 3-15 are reserved for future use and must be 0. For example, the EventMap string "1000000000000000" executes a command when GroupWise starts (event equals startup).

MenuEnabled: *numeric* (optional)

0	Always enabled
1	Enabled on item views
2	Enabled when one or more items are selected

MenuName: *string* (optional)
Menu where MenuItem is inserted. Required if OnMenu is TRUE.

MenuItem: *string* (optional)
Name that appears on the menu. Required if OnMenu is TRUE and MenuSeperator is FALSE.

MenuPosition: *numeric* (optional)
If no position is specified, item appears as the last menu item.

IsMenuSeperator: *enum* (optional)
Command is a menu separator.
FALSE
TRUE

The following example adds a menu separator at the second position on the Tools menu (menu position is zero-based):

```
CustomCommandInstall(CmdName: "Separator_2"; OnMenu: TRUE;
    MenuEnabled: 0; MenuName: "Tools"; MenuPosition: 1;
    IsMenuSeparator: TRUE)
```

LongPrompt: *string* (optional)
Prompt that appears in the Title bar when you select a menu item.

HelpFile: *string* (optional)
Name of a Windows Help File. After adding a custom command to a menu, press Shift+F1 and select the custom command menu item. The Help topic defined by HelpContext or HelpKeyword is displayed.

HelpContext: *numeric* (optional)
Context-number defined in the MAP section of a Windows .HPJ file. Required if Help File is specified.

HelpKeyword: *string* (optional)
Word that appears in the K footnote of a Windows Help File. Required if Help File is specified.

BtnBarName: *string* (optional)
Button display name when you add a custom command (menu item) to the Button Bar. In the Main Window (after adding a custom command to a menu), choose View, Button Bar Preferences, select a Button Bar, Edit, Activate a Feature, select Other Menu Items under Feature Categories, select a custom command, OK. The name specified by this parameter appears on the Button.

BtnBarFile: *string* (optional)
Graphic that displays if you add a custom command to a Button Bar (see BtnBarName parameter).

Description: *string* (optional)
Notes that appear in the BIF under WPOffice (group), Custom Commands (section), Item Value.

CustomMessageCompose

Purpose

Executes a user-defined custom message, defined in the Binary Initialization File (BIF). See CustomMessageInstall.

Syntax

CustomMessageCompose (MessageClase: *string*)

Parameters

MessageClase: *string*
Custom command message.

Route...
If you install a custom message on the Tools menu (see CustomMessageInstall), choose Tools, name of custom message.

CustomMessageDelete

Purpose
Deletes a custom message. Not recordable.

Syntax
CustomMessageDelete (MessageClass: *string*)

Parameters
MessageClass: *string*
Custom message name. See CustomMessageInstall.

CustomMessageInstall

Purpose

Installs a custom message, where an entry is made in the Binary Initialization File (BIF). Exit and restart GroupWise to make sure the custom message is installed properly. Not recordable.

A custom message can be any item type, such as a Mail, Task, or Appointment item. The view name of each custom message is stored in the BIF under the Custom Messages section. If the view name is not in the BIF, or the associated DLL is not found (see DLLName parameter), GroupWise handles custom messages as regular items. Custom message DLLs initiate actions when certain events occur (see EventMap parameter).

Syntax

CustomMessageInstall (MessageClass: *string*; DLLName: *string*; EventMap: *string*; IsIPC: *enum*; CommandLine: *string*; MenuName: *string*; MenuItem: *string*; MenuPosition: *numeric*; LongPrompt: *string*; HelpFile: *string*; HelpContext: *numeric*; HelpKeyword: *string*; SmallIconFile: *string*; SmallIconIndex: *numeric*; LargeIconFile: *string*; LargeIconIndex: *numeric*; BtnBarName: *string*; BtnBarFile: *string*; Description: *string*)

Parameters

MessageClass: *string*
User-defined name, inserted in the BIF under WPOffice (group), Custom Messages (section).

DLLName: *string*
DLL that launches a command (see CommandLine parameter). Optional if MenuSeparator is TRUE. APPEXEC.DLL is a DLL that ships with GroupWise, or a custom DLL.

EventMap: *string* (optional)
Array of characters that specifies how an event is handled. Each element in the array represents an event and is assigned a value of 0 (GroupWise handles the event as a normal view), 1 (DLL handles the event), or 2 (disable the event). Default: 0. From left to right, the events are:

Compose
Open
Reply
ReplyToAll
Forward
Print
Save
Delivery
Accept
Complete
Delegate
Decline
Delete
Info
Archive
Undelete
Resend

The first eight events are recognized by custom message handlers written for Microsoft Windows for Workgroups 3.11. The remaining events are GroupWise specific and, except for Resend, have a corresponding Item*xxxxx*() command that can be sent to GroupWise. This allows the custom message handler (see DLLName parameter) to do preprocessing on an event before GroupWise actually handles it.

Resend. A custom message handler must extract pertinent information from the original message and then create a new one. Users can resend In Box items if they originated the item. The message ID passed to the message handler references the In Box item.

For example, a DLL would handle the first eight events and GroupWise the last eight in the following EventMap string: "1111111100000000". GroupWise handles all unspecified events as normal views.

IsIPC: *enum*
Custom message is an Inter-process communication message. IPC messages do not appear in the In Box, but are automatically passed to an associated DLL. If an IPC view name is not in the BIF, or a DLL is not found (see DLLName parameter), IPC messages appear in the In Box for manual processing by the user.
No!
Yes!

If No!, the custom message is an Inter-personal message. IPM messages appear in In Boxes, and may be handled (displayed) by a DLL rather than by GroupWise.

CommandLine: *string*
Command to install, such as an application (.EXE) or macro (.WCM). Macro example: "MFWIN20.EXE /M-" + *macropath* + *macroname*.

MenuName: *string* (optional)
Menu where MenuItem is inserted.

MenuItem: *string* (optional)
Name that appears on the menu.

MenuPosition: *numeric* (optional)
If no position is specified, item appears as the last menu item.

LongPrompt: *string* (optional)
Prompt that appears in the Title bar when you select a menu item.

HelpFile: *string* (optional)
Name of a Windows Help File. After adding a custom command to a menu, press Shift+F1 and select the custom message menu item. The Help topic defined by HelpContext or HelpKeyword is displayed.

HelpContext: *numeric* (optional)
Context-number defined in the MAP section of a Windows .HPJ file. Required if HelpFile is specified.

HelpKeyword: *string* (optional)
Word that appears in the K footnote of a Windows Help File. Required if HelpFile is specified.

SmallIconFile: *string*
Upper left hand 16x16 quadrant (pixel width and height) is extracted from a 32x32 icon, and used for the 16x16 image in an item list. If no icon is installed, the default is used.

SmallIconIndex: *numeric*
Icon to extract from any .EXE or .DLL file (index is zero-based).

LargeIconFile: *string*
32x32 icon used for the 16x16 image in an item list (see SmallIconFile parameter).

LargeIconIndex: *numeric*
Icon to extract from any .EXE or .DLL file (index is zero-based).

BtnBarName: *string* (optional)
Button display name when you add a custom message (menu item) to the Button Bar. In the Main Window (after adding a custom message to a menu), choose View, Button Bar Preferences, select a Button Bar, Edit, Activate a Feature, select Other Menu Items under Feature Categories, select a custom message, OK. The name specified by this parameter appears on the Button.

BtnBarFile: *string* (optional)
Graphic that displays if you add a custom message to a Button Bar (see BtnBarName parameter).

Description: *string* (optional)
Notes that appear in the BIF under WPOffice (group), Message Commands (section), Item Value.

DBToggle

Purpose

Toggles between current and archived items. Not recordable.

Syntax

DBToggle ()

DBUseArchive

Purpose

Lists archived In Box, Out Box, and Trash items.

Syntax

DBUseArchive ()

Route...
Open an In Box, Out Box, or Trash view, choose File, Open Archive.

DBUsePrimary

Purpose

Lists current In Box, Out Box, and Trash items.

Syntax

DBUsePrimary ()

Route...
Open an In Box, Out Box, or Trash, choose File, Open Archive.

DateAbsoluteGoTo

Purpose

Changes the calendar date to a specified day, month, and year. An error message is displayed if the day, month, and year are out of range. Not recordable.

Syntax

DateAbsoluteGoTo (Day: *numeric*; Month: *numeric*; Year: *numeric*)

Parameters

Day: *numeric*

Month: *numeric*

Year: *numeric*

See Also

DateParse
DateRelativeGoTo

DateDifferenceDlg

F

Purpose
Displays the Date Difference dialog box.

Syntax
DateDifferenceDlg ()

See Also
DateRelativeGoTo

Route...
In a Calendar view, choose View, Date Difference.

DateParse

Purpose

Parses a date character string into numeric day, month, and year equivalents, and assigns them to output variables. Not recordable.

Syntax

DateParse (DateString: *string*; Day: *variable*; Month: *variable*; Year: *variable*; DayOfWeek: *variable*)

Parameters

DateString: *string*
For example: "Jun 17, 1995" returns 17, 6, and 1995 in the Day, Month, and Year variables, and 6 in the DayOfWeek variable. A date string of numbers parses to the default date format.

Day: *variable*

Month: *variable*

Year: *variable*

DayOfWeek: *variable*
Sunday = 0, Monday = 1, and so forth.

See Also

DateAbsoluteGoTo

DateParseDay

Purpose
Parses a date string and returns the day of the month. Not recordable.

Syntax
numeric DateParseDay (DateString: *string*)

Return Value
numeric Day of the month.

Parameters
DateString: *string*
For example: "Jun 17, 1995" returns 17.

DateParseDayOfWeek

Purpose
Parses a date string and returns the day of the week. Not recordable.

Syntax
numeric DateParseDayOfWeek (DateString: *string*)

Return Value
numeric Day of the week.

Parameters
DateString: *string*
For example: "Jun 17, 1995" returns 6.

DateParseDurationHours

Purpose
Parses a date string and returns the number of hours. Not recordable.

Syntax
numeric DateParseDurationHours (DateString: *string*)

Return Value
numeric Number of hours.

Parameters
DateString: *string*
For example: "2.5 days" returns 60; "60 minutes" returns 1.

DateParseDurationMinutes

Purpose
Parses a date string and returns the number of minutes. Not recordable.

Syntax
numeric DateParseDurationMinutes (DateString: *string*)

Return Value
numeric Number of minutes.

Parameters
DateString: *string*
For example: "1.5 hours" returns 90.

DateParseHours

Purpose
Parses a time string and returns the number of hours. Not recordable.

Syntax
numeric DateParseHours (DateString: *string*)

Return Value
numeric Number of hours.

Parameters
DateString: *string*
For example: "10:30" returns 10.

DateParseMinutes

Purpose
Parses a time string and returns the number of minutes. Not recordable.

Syntax
numeric DateParseMinutes (DateString: *string*)

Return Value
numeric Number of minutes.

Parameters
DateString: *string*
For example: "10:30" returns 30.

DateParseMonth

Purpose
Parses a date string and returns the month. Not recordable.

Syntax
numeric DateParseMonth (DateString: *string*)

Return Value
numeric Number of month.

Parameters
DateString: *string*
For example: "Jun 17, 1995" returns 6.

DateParseYear

Purpose
Parses a date string and returns the year. Not recordable.

Syntax
numeric DateParseYear (DateString: *string*)

Return Value
numeric Year.

Parameters
DateString: *string*
For example: "Jun 17, 1995" returns 1995.

DateRelativeGoTo

Purpose

Sets the calendar date forward (positive) or backward (negative) relative to the current date. If no parameters are specified, the date is set to the current date.

Syntax

DateRelativeGoTo (RelativeDay: *numeric*; RelativeMonth: *numeric*; RelativeYear: *numeric*)

Parameters

RelativeDay: *numeric* (optional)
Number of days.

RelativeMonth: *numeric* (optional)
Number of months.

RelativeYear: *numeric* (optional)
Number of years.

See Also

DateAbsoluteGoTo
DateDifferenceDlg

Route...
Open a Calendar view, choose View, Go to Date, specify a date.

DateRelativeGoToDlg

Purpose
Displays the Go to Date dialog box.

Syntax
DateRelativeGoToDlg ()

See Also
DateRelativeGoTo

Route...
Open a Calendar view, choose View, Go to Date.

DateSetAutodateMode

Purpose
Displays the Auto-Date dialog box to schedule recurring appointments, tasks, or notes.

Syntax
DateSetAutodateMode ()

Route...
Open a Schedule, Assign Task, or Write Note view, choose Send, Auto-Date.

DateSetStartDateMode

Purpose
Displays the Set Date dialog box to set appointment, task, or note dates.

Syntax
DateSetStartDateMode ()

Route...
Open a Schedule, Assign Task, or Write Note view, choose Send, Set Date.

DateSwitchToAutoDate

Purpose

Switches from Start Date to Auto-Date to schedule recurring appointments, tasks, and/or notes. Not recordable.

Syntax

DateSwitchToAutoDate ()

DateSwitchToDuration

Purpose

Switches from End Date to Duration to display the length of an appointment. Not recordable.

Syntax

DateSwitchToDuration ()

DateSwitchToEndDate

Purpose

Switches from Duration to End Date to display the length of an appointment. Not recordable.

Syntax

DateSwitchToEndDate ()

DateSwitchToStartDate

Purpose

Switches from Auto-Date to Start Date to schedule appointments, tasks, and/or notes. Not recordable.

Syntax

DateSwitchToStartDate ()

DayColumnAdd

Purpose

Adds a new day column to the Week Calendar view (up to six days).

Syntax

DayColumnAdd ()

Route...
Open the Week Calendar view, click the plus button in the upper left corner.

DayColumnDelete

F

Purpose
Deletes a day column from the Week calendar view (down to one day).

Syntax
DayColumnDelete ()

See Also
Delete

Route...
Open the Week Calendar view, click the minus button in the upper left corner.

Delete

Purpose

Deletes selected text, selected items, or a character at the insertion point.

Syntax

Delete ()

Route...
Select an item, press Delete.

DeleteCharPrevious

Purpose
Deletes selected text, or one character to the left of the insertion point.

Syntax
DeleteCharPrevious ()

Route...
Press Backspace.

DeleteWord

Purpose

Deletes the word at the insertion point. Not recordable.

Syntax

DeleteWord ()

DeleteWordLeft

Purpose

Deletes the word at the insertion point, to the left of the insertion point (if it is after the last character), or to the right of the insertion point (if it is before the first character).

Syntax

DeleteWordLeft ()

Route...
Press Ctrl+Backspace.

DeleteWordRight

Purpose
Deletes all text right of the insertion point to the end of the line.

Syntax
DeleteWordRight ()

Route...
Press Ctrl+Delete.

EditCopy

Purpose
Copies selected text to the clipboard, replacing existing clipboard text.

Syntax
EditCopy ()

See Also
EditCut
OlePasteLink

Route...
Select text, choose Edit, Copy.

EditCut

Purpose
Removes selected text and copies it to the clipboard, replacing existing clipboard text.

Syntax
EditCut ()

Route...
Select text, choose Edit, Cut.

EditPaste

Purpose
Inserts clipboard text at the insertion point.

Syntax
EditPaste()

Route...
Place the insertion point in an Edit box, choose Edit, Paste.

EmptySelectedItems

Purpose

Removes selected items from Trash and erases them from the Post Office database.

Syntax

EmptySelectedItems ()

Route...
Open Trash, select item(s) to remove, choose Edit, Empty Selected Items.

EmptyTrash

Purpose

Removes all items from Trash and erases them from the Post Office database.

Syntax

EmptyTrash ()

Route...
Open Trash, choose Edit, Empty Trash.

Enter

Purpose
Inserts a hard return at the insertion point, or opens a selected item.

Syntax
Enter ()

Route...
Select an item, press Enter.

EnumSelect

Purpose

Selects an enumeration from a user-defined control such as a popup button.

Syntax

EnumSelect (Text: *string*; ControlName: *string*)

Parameters

Text: *string*
Enumeration to select, such as a list item on a popup button control.

ControlName: *string*
Name of a user-defined control. For example, open GroupWise View Designer and create a popup button control. Then choose Edit, Item Properties. Type the ControlName in the List Name edit box.

Route...
Open a customized view, select an enumeration from a custom control.

EnvCheckCurrentWindow

Purpose
Returns True if the current window type matches a specified window type, False if not. Not recordable.

Syntax
boolean EnvCheckCurrentWindow (WindowType: *enum*)

Return Value
boolean True/False.

Parameters
WindowType: *enum*
- AppointmentMessage!
- AttachmentViewer!
- Calendar!
- Inbox!
- Info!
- ItemView!
- MailMessage!
- MainWindow!
- ModelessDialog!
- NoteMessage!
- Outbox!
- PhoneMessage!
- TaskMessage!
- Trash!

EnvClipboardText

Purpose
Returns the contents (text) of the clipboard. Not recordable.

An error occurs if the clipboard contains a graphic.

Syntax
string EnvClipboardText ()

Return Value
string Clipboard text.

EnvCommandLine

Purpose

Returns the command line string passed to GroupWise at startup. Not recordable.

Startup options are specified in the Windows Program Manager (see Startup Options in GroupWise Help).

Syntax

string EnvCommandLine ()

Return Value

string Command line string.

EnvCurrentViewName

Purpose
Returns the name of the active view. Not recordable.

Syntax
string EnvCurrentViewName ()

Return Value
string Name of active view.

EnvDefaultConnectionName

Purpose
Returns the default Remote connection name. Not recordable.

Syntax
string EnvDefaultConnectionName ()

Return Value
string Connection name.

EnvIsCommandValid

Purpose
Returns True if a command is valid, False if not. Not recordable.

Syntax
boolean EnvIsCommandValid (CommandID: *numeric*)

Return Value
boolean True/False

Parameters
CommandID: *numeric*
See *GroupWise Software Developer's Kit* for command IDs.

EnvIsMacroPlaying

Purpose

Returns True if a macro is playing or is paused, False if not. Not recordable.

This command is normally called by a DDE client or cross-application macro.

Syntax

boolean EnvIsMacroPlaying ()

Return Value

boolean True/False.

EnvIsMacroRecording

Purpose

Returns True if a macro is being recorded, False if not. Not recordable.

This command is normally called by a DDE client or cross-application macro.

Syntax

boolean EnvIsMacroRecording ()

Return Value

boolean True/False.

EnvIsNetworkUser

Purpose

Returns True if the current user is logged into the network, False if not. Not recordable.

This command detects whether the /la startup option is used (see Startup Options in GroupWise Help).

Syntax

boolean EnvIsNetworkUser ()

Return Value

boolean True/False.

EnvIsRemote

Purpose
Returns True if GroupWise is running as a remote client, False if not. Not recordable.

Syntax
boolean EnvIsRemote ()

Return Value
boolean True/False.

EnvIsRemoteConnectionActive

Purpose
Returns True if the remote connection is active, False if not. Not recordable.

Syntax
boolean EnvIsRemoteConnectionActive ()

Return Value
boolean True/False.

EnvIsRemoteConnectionFinished

Purpose
Returns True if the updating of the remote mailbox is complete and there are no outstanding remote requests, False if not. Not recordable.

Syntax
boolean EnvIsRemoteConnectionFinished ()

Return Value
boolean True/False.

EnvIsRemoteConnectionNeeded

Purpose

Returns True if a remote connection is needed, False if not. Not recordable.

A connection is needed when items are in the Pending Requests to Master Mailbox dialog box. Open GroupWise Remote 4.1, choose Remote, Pending Requests.

Syntax

boolean EnvIsRemoteConnectionNeeded ()

Return Value

boolean True/False.

EnvLastCmdResult

Purpose
Returns the value returned by the last DDE-issued command. Not recordable.

Syntax
string EnvLastCmdResult ()

Return Value
string Value returned by the last DDE-issued command.

EnvLastError

Purpose

Returns information about an error condition that causes a DDE or other transaction to fail. Not recordable.

Use this command as part of a DDE request transaction. Possible error strings are:

10E1 Bad Parameter x (x is the index of the bad parameter)
10E2 Failed
10E3 Command is invalid in the current GroupWise state
10E4 Not Found
10E5 Syntax Error
10E7 BC does not apply to personal items
10E8 Not supported for encapsulated item attachments
10E9 Not supported for OLE attachments
10EA Message ID references an Out Box message that does not exist yet
10EB You cannot set an alarm for a time that has already expired
10EC The In Box and Out Box source attributes of the In Box filter cannot be changed
10ED The source attributes of the Out Box filter cannot be changed

xxxx GroupWise Error (other GroupWise error)

Syntax

string EnvLastError ()

Return Value

string Error condition.

EnvLastWindowCanceled

Purpose
Returns True if the last displayed window was canceled, False if not. Not recordable.

Syntax
boolean EnvLastWindowCanceled ()

Return Value
boolean True/False.

EnvNetworkLoginID

Purpose
Returns the network login ID of the current user. Not recordable.

Syntax
string EnvNetworkLoginID ()

Return Value
string Network login ID.

See Also
EnvIsNetworkUser

EnvPrefArchivePath

Purpose
Returns the default path of archive database files. Not recordable.

Syntax
string EnvPrefArchivePath ()

Return Value
string Archive database path.

EnvPrefCustomViewPath

Purpose
Returns the default path of custom view files. Not recordable.

Syntax
string EnvPrefCustomViewPath ()

Return Value
string Custom view path.

EnvPrefFullName

Purpose
Returns a user's full name. Not recordable.

Syntax
string EnvPrefFullName ()

Return Value
string User name.

EnvPrefMacroPath

Purpose
Returns the default macro path. Not recordable.

Syntax
string EnvPrefMacroPath ()

Return Value
string Macro path.

EnvPrefPostOfficePath

Purpose
Returns the default Post Office path. Not recordable.

Syntax
string EnvPrefPostOfficePath ()

Return Value
string Post Office path.

EnvPrefSavePath

Purpose
Returns the default path for item (*.MLM) files and attachments. Not recordable.

Syntax
string EnvPrefSavePath ()

Return Value
string Item and attachment path.

EnvPrivateBif

Purpose
Returns the default path and name of the private Binary Initialization File (BIF). Not recordable.

Syntax
string EnvPrivateBif ()

Return Value
string Private BIF path.

EnvPublicBif

Purpose
Returns the default path and name of the public Binary Initialization File (BIF). Not recordable.

Syntax
string EnvPublicBif ()

Return Value
string Public BIF path.

EnvSharedDLLPath

Purpose
Returns the default path for Shared Code DLL files. Not recordable.

Syntax
string EnvSharedDLLPath ()

Return Value
string Shared Code path.

EnvTextCurrentCharIndex

Purpose
Returns the number of characters from the beginning of a line to the insertion point (index is zero-based). Not recordable.

Syntax
numeric EnvTextCurrentCharIndex ()

Return Value
numeric Number of characters.

EnvTextCurrentLineIndex

Purpose
Returns the current line number (index is zero-based). Not recordable.

Syntax
numeric EnvTextCurrentLineIndex ()

Return Value
numeric Number of current line.

EnvTextCurrentWord

Purpose

Returns the text at the insertion point. Text is separated by spaces, hard returns, and punctuation such as a period or semicolon. Not recordable.

Syntax

string EnvTextCurrentWord ()

Return Value

string Text at the insertion point.

EnvTextInsertMode

Purpose
Returns True if Insert mode is on, False if off. Not recordable.

Syntax
boolean EnvTextInsertMode ()

Return Value
boolean True/False.

EnvTextLeftChar

Purpose
Returns the character left of the insertion point, or an empty string if the insertion point is at the beginning of the line. Not recordable.

Syntax
string EnvTextLeftChar ()

Return Value
string Character.

EnvTextLineCharCount

Purpose
Returns the number of characters in a specified line of text, including HRt and SRt codes. Not recordable.

Syntax
numeric EnvTextLineCharCount (Index: *Numeric*)

Return Value
numeric Number of characters.

Parameters
Index: *Numeric* (optional)
A line number (zero-based). Default is current line of text.

EnvTextLineCount

Purpose
Returns the number of lines in the current message box (must have input focus), 1 if empty. Not recordable.

Syntax
numeric EnvTextLineCount ()

Return Value
numeric Number of lines.

EnvTextRightChar

Purpose

Returns the character to the right of the insertion point, or an empty string if the insertion point is at the end of the line. Not recordable.

Syntax

string EnvTextRightChar ()

Return Value

string Character.

EnvUserID

Purpose
Returns the current user ID. Not recordable.

Syntax
string EnvUserID ()

Return Value
string User ID.

EnvVersionDate

Purpose
Returns the GroupWise version date. Not recordable.

Syntax
string EnvVersionDate ()

Return Value
string GroupWise version date.

EnvVersionName

Purpose
Returns the GroupWise version name. Not recordable.

Syntax
string EnvVersionName ()

Return Value
string GroupWise version.

EventNotify

Purpose

Signals a third-party DLL when certain events happen in GroupWise 4.1. This token is only called from a DLL. Not recordable.

The first event signals a third-party DLL that GroupWise 4.1 is running. The second and third events signal that a custom command or message failed. The fourth event signals that a context menu (right mouse click menu) is about to be displayed. See AppEvent parameter.

Syntax

EventNotify (AppEvent: *enum*; Command: *string*)

Parameters

AppEvent: *enum*

AppInitialized!
BadCustomCommand!
BadCustomMessage!
ContextMenu!

Command: *string*

For AppInitialized!: "WPOF: App Initialized".
For BadCustomCommand!: "WPOF: Custom command improperly defined".
For BadCustomMessage!: "WPOF: Custom message improperly defined".
For ContextMenu!, a string representation of the decimal ID (MENU_ID) followed by a space and the decimal equivalent of the menu handle (HMENU) for the context menu that is about to be displayed. This allows the third-party DLL to add menu items to the context menu before it is displayed. EventNotify supports the following menu IDs:

20 OLE Object
21 Shelf Icon or Quick Start Icon
22 Attachment control (composing message, attachment selected)
23 Attachment control (reading message)
24 Soun item or control
25 Item list control (In Box, Out Box, Calendar views)
26 Movie control
27 (Reserved)
28 Main Window Icons (In Box, Out Box, Calendar, Mail, Appointment, and so forth)
29 In Box view
30 Out Box view
31 Trash view
32 (Reserved)
33 (Reserved)
34 (Reserved)
35 Item list (preempted by local menu ID 25, but can be accessed in the control space to the left of the Column Manager)
36 Basic item (may be overridden by more specific view menus)
37 Basic item view (may be overridden by more specific view menus)
38 Folder list control (In Box, Out Box, Trash, Calendar views)
39 Button Bar
40 CC and BC controls (composing message)
41 Main Window shelf (no shelf icon selected)

42 Attachment control (composing message, no attachment selected)
43 Trash item list (preempted by local menu ID 31, but can be accessed in the control space to the left of the Column Manager)
44 (Reserved)
45 (Reserved)
46 Start Date/Auto-Date control (composing message)
47 End Date/Duration control (composing message)
48 To control (composing message)
49 Main Window's Trash icon.

FilterApply

Purpose
Applies a filter to a specified list or control. Not recordable.

Syntax
FilterApply (Handle: *numeric*; ApplyTo: *enum*; ControlName: *string*)

Parameters
Handle: *numeric*
Unique filter identifier, returned by FilterCreate.

ApplyTo: *enum*
AppointmentList!
ItemList!
NoteList!
SelectedControl!
TaskList!
View!
WeekControl!

ItemList! includes all items. WeekControl! includes Week calendar view items.

ControlName: *string*
Name of a user-defined control.

FilterCreate

Purpose
Creates a new filter. Not recordable.


Delete the filter handle when it is no longer needed, or system resources will be lost. See FilterDelete.

Syntax
numeric FilterCreate (FilterType: *enum*)

Return Value
numeric Filter handle (a number greater than zero, or zero if an error occurs).

Parameters
FilterType: *enum*
Calendar!
Empty!
Inbox!
Outbox!

Calendar!, Inbox!, and Outbox! automatically apply defined filter criteria. For example, since Calendar! only displays appointment, note, and task items, you do not have to filter mail items (see FilterSetItemType). Empty! requires you to set all filter criteria, including type, source, and attributes as needed.

FilterDelete

Purpose
Deletes a filter handle from memory. Not recordable.


Delete the filter handle when it is no longer needed, or system resources will be lost. See FilterCreate.

Syntax
FilterDelete (Handle: *numeric*)

Parameters
Handle: *numeric*
Filter handle.

FilterDlg

Purpose

Displays the Filter, Filter In Box, or Filter Out Box dialog box, depending on the view displayed.

Syntax

FilterDlg ()

Route...
Open an In Box, Out Box, Trash, or Calendar view, and choose View, Filter.

FilterExecute

Purpose

Specifies a filter to apply to a list or control.

Syntax

FilterExecute (AttachmentFile: *enum*; AttachmentMessage: *enum*; AttachmentMovie: *enum*; AttachmentSound: *enum*; AttribAccepted: *enum*; AttribCompleted: *enum*; AttribOpened: *enum*; AttribPrivate: *enum*; AttribRead: *enum*; AttribReturnReceiptReq: *enum*; AttribReplyRequested: *enum*; AttribRouted: *enum*; DateType: *enum*; DateRangeType: *enum*; StartDate: *enum*; StartDateOffset: *numeric*; EndDate: *enum*; EndDateOffset: *numeric*; ItemTypeAppointment: *enum*; ItemTypeMail: *enum*; ItemTypeNote: *enum*; ItemTypePhone: *enum*; ItemTypeTask: *enum*; PriorityHigh: *enum*; PriorityLow: *enum*; PriorityNormal: *enum*; SourceInbox: *enum*; SourceOutbox: *enum*; SourcePersonal: *enum*; AuthorityText: *string*; CallerName: *string*; CCText: *string*; CompanyText: *string*; FromText: *string*; MessageText: *string*; CallerPhone: *string*; PlaceName: *string*; SubjectText: *string*; CategoryText: *string*; PriorityText: *string*; ToText: *string*; ViewName: *string*; ApplyTo: *enum*; ControlName: *string*)

Parameters

AttachmentFile: *enum*
- No!
- Yes!

AttachmentMessage: *enum*
- No!
- Yes!

AttachmentMovie: *enum*
- No!
- Yes!

AttachmentSound: *enum*
- No!
- Yes!

AttribAccepted: *enum*
- DontCare!
- No!
- Yes!

AttribCompleted: *enum*
- DontCare!
- No!
- Yes!

AttribOpened: *enum*
- DontCare!
- No!
- Yes!

AttribPrivate: *enum*

DontCare!
No!
Yes!

AttribRead: *enum*
DontCare!
No!
Yes!

AttribReturnReceiptReq: *enum*
DontCare!
No!
Yes!

AttribReplyRequested: *enum*
DontCare!
No!
Yes!

AttribRouted: *enum*
DontCare!
No!
Yes!

DateType: *enum*
AssignedDate!
BeginDateTime!
CreateDate!
DueDate!
Duration!
EndDateTime!

DateRangeType: *enum*
After!
Anytime!
Before!
Between!
On!

StartDate: *enum*
BeginningOfMonth!
BeginningOfWeek!
BeginningOfYear!
EndOfMonth!
EndOfWeek!
EndOfYear!
Today!
Tomorrow!
Yesterday!

StartDateOffset: *numeric*

EndDate: *enum*
BeginningOfMonth!
BeginningOfWeek!

BeginningOfYear!
EndOfMonth!
EndOfWeek!
EndOfYear!
Today!
Tomorrow!
Yesterday!

EndDateOffset: *numeric*

ItemTypeAppointment: *enum*
No!
Yes!

ItemTypeMail: *enum*
No!
Yes!

ItemTypeNote: *enum*
No!
Yes!

ItemTypePhone: *enum*
No!
Yes!

ItemTypeTask: *enum*
No!
Yes!

PriorityHigh: *enum*
No!
Yes!

PriorityLow: *enum*
No!
Yes!

PriorityNormal: *enum*
No!
Yes!

SourceInbox: *enum*
No!
Yes!

SourceOutbox: *enum*
No!
Yes!

SourcePersonal: *enum*
No!
Yes!

AuthorityText: *string*

CallerName: *string*

CCText: *string*

CompanyText: *string*

FromText: *string*

MessageText: *string*

CallerPhone: *string*

PlaceName: *string*

SubjectText: *string*

CategoryText: *string*

PriorityText: *string*

ToText: *string*

ViewName: *string*

ApplyTo: *enum*
- AppointmentList!
- ItemList!
- NoteList!
- SelectedControl!
- TaskList!
- View!
- WeekControl!

ControlName: *string*

Route...
Open an item list, choose View, Filter, select options.

FilterFromFile

Purpose
Retrieves filter information saved to a file. Not recordable.

Syntax
FilterFromFile (FileName: *string*)

Parameters
FileName: *string*
Name of filter information file.

FilterSetAttachmentClass

Purpose

Filters items according to the contents of the Attach list box. Not recordable.

Syntax

FilterSetAttachmentClass (Handle: *numeric*; FilterAttachmentClass: *enum*; IsRequired: *enum*)

Parameters

Handle: *numeric*
Unique filter identifier, returned by FilterCreate.

FilterAttachmentClass: *enum*
Attachment type to filter, or not filter, depending on the IsRequired parameter setting. Use multiple commands to filter more than one attachment type.

- AttachClassFile!
- AttachClassMovie!
- AttachClassSound!

IsRequired: *enum*

- DontCare!
- No!
- Yes!

No! filters out items that have attachments. Yes! filters out items that do not have attachments.

FilterSetAttribute

Purpose
Filters items by specified attributes. Not recordable.

Syntax
FilterSetAttribute (Handle: *numeric*; Attribute: *enum*; IsRequired: *enum*)

Parameters
Handle: *numeric*
Unique filter identifier, returned by FilterCreate.

Attribute: *enum*
Attribute to filter, or not filter, depending on the IsRequired parameter setting. Use multiple commands to filter more than one attribute type.
- Accepted!
- Completed!
- CustomView!
- Delegated!
- Deleted!
- Opened!
- Personal!
- Private!
- Read!
- ReplyRequested!
- ReturnReceiptRequested!
- Routed!

IsRequired: *enum*
- DontCare!
- No!
- Yes!

No! filters out items that have specified attributes. Yes! filters out items that do not have specified attributes.

FilterSetDateAbsolute

Purpose
Filters items according to an absolute date or date range. Not recordable.

Syntax
FilterSetDateAbsolute (Handle: *numeric*; DateType: *enum*; DateRangeType: *enum*; StartDay: *numeric*; StartMonth: *numeric*; StartYear: *numeric*; EndDay: *numeric*; EndMonth: *numeric*; EndYear: *numeric*)

Parameters
Handle: *numeric*
Unique filter identifier, returned by FilterCreate.

DateType: *enum*
- AssignedDate!
- BeginDateTime!
- CreateDate!
- DueDate!
- Duration!
- EndDateTime!

DateRangeType: *enum*
- After!
- Anytime!
- Before!
- Between!
- On!

StartDay: *numeric* (optional)

StartMonth: *numeric* (optional)
Example: January = 1.

StartYear: *numeric* (optional)
Requires a 4-digit date. Example: 1994.

EndDay: *numeric* (optional)

EndMonth: *numeric* (optional)
Example: December = 12.

EndYear: *numeric* (optional)
Requires a 4-digit date. Example: 1994.

See Also
FilterSetDateRelative

FilterSetDateRelative

Purpose
Filters items according to a relative date and date type. Not recordable.

Syntax
FilterSetDateRelative (Handle: *numeric*; DateType: *enum*; StartDate: *enum*; StartDateOffset: *numeric*; EndDate: *enum*; EndDateOffset: *numeric*)

Parameters
Handle: *numeric*
Unique filter identifier, returned by FilterCreate.

DateType: *enum*
AssignedDate!
BeginDateTime!
CreateDate!
DueDate!
Duration!
EndDateTime!

StartDate: *enum*
BeginningOfMonth!
BeginningOfWeek!
BeginningOfYear!
EndOfMonth!
EndOfWeek!
EndOfYear!
Today!
Tomorrow!
Yesterday!

Set BeginningOfWeek! day in Environment preferences. From the Main Window, choose File, Preferences, Environment, select a First Day of the Week option.

StartDateOffset: *numeric* (optional)
Number of offset days (positive or negative value). For example, the following are equivalent commands that filter out dates before and after last week:

FilterSetDateRelative (Handle: hFilter; DateType: CreateDate!; StartDate: BeginningOfWeek!; StartDateOffset: -7; EndDate: BeginngingOfWeek; EndDateOffset: -1)

or

FilterSetDateRelative (Handle: hFilter; DateType: CreateDate!; StartDate: BeginningOfWeek; StartDateOffset: -7; EndDate: EndOfWeek; EndDateOffset: -7)

EndDate: *enum* (optional)
BeginningOfMonth!
BeginningOfWeek!
BeginningOfYear!

EndOfMonth!
EndOfWeek!
EndOfYear!
Today!
Tomorrow!
Yesterday!

EndDateOffset: *numeric* (optional)
See StartDateOffset above. The following complete example filters out all Out Box items created before the current date:

```
Application(A1; "WPOffice"; Default; "US")
ViewOpen(ViewType: Outbox!)
hFilter = FilterCreate(FilterType: OutBox!)
FilterSetDateRelative(Handle: hFilter; DateType: CreateDate!;
    StartDate: Today!)
FilterApply(Handle: hFilter; ApplyTo: ItemList!)
FilterDelete(Handle: hFilter)
```

See Also
FilterSetDateAbsolute

FilterSetItemType

Purpose
Filters items according to a specified item type. Not recordable.

Syntax
FilterSetItemType (Handle: *numeric*; ItemType: *enum*; WillPassFilter: *enum*)

Parameters
Handle: *numeric*
Unique filter identifier, returned by FilterCreate.

ItemType: *enum*
Item type to filter, or not filter, depending on the WillPassFilter setting. Use multiple commands to filter more than one item type.

Appointment!
Mail!
Note!
Phone!
Task!

The following example allows Appointment! and Task! items to pass through an In Box filter:

```
Application(A1; "WPOffice"; Default; "US")
ViewOpen(ViewType: Inbox!)
hFilter = FilterCreate(Inbox!)
FilterSetItemType(Handle: hFilter; ItemType: Appointment!;
    WillPassFilter: Yes!)
FilterSetItemType(Handle: hFilter; ItemType: Task!;
    WillPassFilter: Yes!)
FilterApply(Handle: hFilter; ApplyTo: ItemList!)
FilterDelete(Handle: hFilter)
```

WillPassFilter: *enum*
No!
Yes!

No! filters out the specified item type. Yes! passes the specified item type through the filter, but no others.

FilterSetPriority

Purpose
Filters items according to priority. Not recordable.

Syntax
FilterSetPriority (Handle: *numeric*; Priority: *enum*; WillPassFilter: *enum*)

Parameters
Handle: *numeric*
Unique filter handle, returned by FilterCreate.

Priority: *enum*
Priority to filter, or not filter, depending on the WillPassFilter setting. Use multiple commands to filter more than one priority.
- High!
- Low!
- Normal!

WillPassFilter: *enum*
- No!
- Yes!

No! filters items with the specified priority. Yes! passes items with the specified priority through the filter, but no others.

FilterSetSource

Purpose

Filters items according to their source. You must specify a source if ItemCreate created an empty filter. Not recordable.

Syntax

FilterSetSource (Handle: *numeric*; Source: *enum*; WillPassFilter: *enum*)

Parameters

Handle: *numeric*
Unique filter identifier, returned by FilterCreate.

Source: *enum*
- Inbox!
- Outbox!
- Personal!

WillPassFilter: *enum*
- No!
- Yes!

No! filters out items from the specified source. Yes! passes items with the specified source through the filter, but no others.

FilterSetText

Purpose
Filters items according to the text in a text field. Not recordable.

Syntax
FilterSetText (Handle: *numeric*; Field: *enum*; FieldText: *string*; Match: *enum*)

Parameters
Handle: *numeric*
Unique filter identifier, returned by FilterCreate.

Field: *enum*
Text field to filter. Use multiple commands to filter more than one field type.
- Authority!
- BC!
- CC!
- Caller!
- Company!
- From!
- Message!
- Phone!
- Place!
- Subject!
- TaskCategory!
- TaskPriority!
- To!
- ViewName!

FieldText: *string*
Text to match.

Match: *enum*
Position of the text in a character string.
- BeginText!
- FullText!
- SubText!

BeginText! matches a string from the first character to the length of the search string; FullText! matches an exact string and is case sensitive; SubText! matches any substring. Default: SubText!.

Find

Purpose

Finds a character string in the From, Subject, or Message edit boxes.

A Not Found condition occurs if you use FromSearchString, SubjectSearchString, and MessageSearchString, and one or more does not find a match. When you search only one edit box, you must include the parameter name. For example,

```
Find(FromSearchString: "John"; SearchDir: Forward!; FromMatch: SubText!)
```

searches forward from the insertion point for John in a From edit box.

Syntax

Find (FromSearchString: *string*; SubjectSearchString: *string*; MessageSearchString: *string*; SearchDir: *enum*; FromMatch: *enum*; SubjectMatch: *enum*; MessageMatch: *enum*)

Parameters

FromSearchString: *string* (optional)
From edit box character string.

SubjectSearchString: *string* (optional)
Subject edit box character string.

MessageSearchString: *string* (optional)
Message edit box character string.

SearchDir: *enum* (optional)
Search forward or backward from the insertion point. Default: Forward!.
- Forward!
- Backward!

FromMatch: *enum* (optional)
From edit box match criteria.
- BeginText!
- FullText!
- SubText!

BeginText! matches a string from the first character to the length of the search string; FullText! matches an exact string and is case sensitive; SubText! matches any substring. Default: SubText!.

SubjectMatch: *enum* (optional)
Match criteria for the Subject edit box.
- BeginText!
- FullText!
- SubText!

BeginText! matches a string from the first character to the length of the search string; FullText! matches an exact string and is case sensitive; SubText! matches any substring. Default: SubText!.

MessageMatch: *enum* (optional)
Match criteria for the Message edit box.
BeginText!
FullText!
SubText!

BeginText! matches a string from the first character to the length of the search string; FullText! matches an exact string and is case sensitive; SubText! matches any substring. Default: SubText!.

See Also

FindNext
FindPrevious

Route...
Open a view, choose Edit, Find, type search text in the From, Subject, or Message edit boxes.

FindDlg

Purpose

Opens the Find dialog box to search the Message edit box of the current view, or displays an error message if the Message edit box does not have the input focus.

Syntax

FindDlg ()

Route...
Select a list item or open an item view, choose Edit, Find.

FindNext

Purpose
Finds the next occurrence of a character string (see Find).

Syntax
FindNext ()

Route...
Select a list item or open an item view, choose Edit, Find Next.

FindPrevious

Purpose

Finds the previous occurrence of a character string (see Find).

Syntax

FindPrevious ()

Route...
Select a list item or open an item view, choose Edit, Find Previous.

FocusSet

Purpose

Gives the input focus to a specified Windows control.

A dashed line inside a push button, or surrounding the text of a check box or radio button, identifies the control that has the input focus. A flashing insertion point identifies an edit control that has the input focus.

Syntax

FocusSet (Place: *enum*; ControlName: *string*)

Parameters

Place: *enum*

Control to receive the input focus.

- AcceptBtn!
- AddressBtn!
- AttachBtn!
- Attachments!
- Authority!
- BC!
- BusyBtn!
- CC!
- Called!
- Caller!
- CameToSeeYou!
- CancelOrCloseBtn!
- Company!
- ControlName!
- DayAppt!
- DayNote!
- DayTask!
- DeclineBtn!
- DelegateBtn!
- DeleteBtn!
- FolderList!
- ForwardBtn!
- From!
- InfoBtn!
- ItemList!
- Message!
- OkBtn!
- Phone!
- Place!
- PleaseCall!
- PrintBtn!
- Priority!
- ReplyBtn!
- ReturnedYourCall!
- SaveBtn!
- Security!
- SendBtn!

SmallMonth!
Subject!
To!
Urgent!
UserDefinedCheckBox!
UserDefinedEnum!
WantsToSeeYou!
WeekAppt!
WeekNote!
WeekTask!
WillCall!

ControlName: *string*
Name of a user-defined control.

Route...
Tab to a control such as an edit box or push button.

FolderAddSelections

Purpose

Adds up to 50 folders to a list of one or more selected folders.

Use multiple commands to make selections without deleting a previous selection.

Syntax

FolderAddSelections (FolderName00: *string*; ...FolderName49: *string*)

Parameters

FolderName00: *string*
Include full folder path. Separate multiple folders with a semicolon.

See Also

FolderSelect

Route...
Select 50 folders, press Shift+Up Arrow or Shift+Down Arrow to select another folder. FolderSelect records when less than 50 folders are selected.

FolderContract

Purpose
Contracts a specified folder (hides all subfolders).

Syntax
FolderContract (FolderName: *string*)

Parameters
FolderName: *string*
Include full folder path.

See Also
FolderExpand
FolderExpandAll

Route...
Select a folder, press Shift+Minus.

FolderContractToOneLevel

Purpose

Contracts a specified folder to one level (hides subfolders). There must be two levels of subfolders.

Syntax

FolderContractToOneLevel (FolderName: *string*)

Parameters

FolderName: *string*
Include full folder path.

Route...
Select a folder, press Ctrl+Minus.

FolderCreate

Purpose
Creates a new folder.

Syntax
FolderCreate (FolderName: *string*)

Parameters
FolderName: *string*
Include full folder path.

Route...
Select a folder, choose File, Folders, Create, type a new folder name.

FolderCreateDlg

Purpose

Displays the Folder Name dialog box to create a new folder. The command fails if a folder item is selected instead of a folder. Not recordable.

Syntax

FolderCreateDlg (FolderName: *string*)

Parameters

FolderName: *string* (optional)
Include full folder path.

FolderDeepMove

Purpose
Deletes all links and moves selected items to one or more folders (up to 50).

Syntax
FolderDeepMove (FolderName00: *string*; ... FolderName49: *string*)

Parameters
FolderName00: *string*; ... FolderName49: *string*
Include full folder path. Separate multiple folders with a semicolon.

See Also
FolderLinkTo
FolderMoveTo

Route...
Select a folder item, choose File, Move/Link to Folders, select one or more folders, select Delete Old Links, choose Move.

FolderDelete

Purpose
Deletes one or more folders (up to 50) and their contents, or only the contents.

Syntax
FolderDelete (DeleteType: *enum*; FolderName00: *string*; ... FolderName49: *string*)

Parameters
DeleteType: *enum*
ItemsAndFolders!
ItemsOnly!

FolderName00: *string*; ... FolderName49: *string*
Include full folder path. Separate multiple folders with a semicolon.

See Also
FolderRename
FolderSelect

Route...
Select one or more folders, choose Edit, Delete, select an option.

FolderDeleteDlg

Purpose

Displays the Delete Folder(s) dialog box. The command fails if a folder item is selected instead of a folder. Not recordable.

Syntax

FolderDeleteDlg ()

See Also

FolderDelete

FolderExpand

Purpose
Displays the subfolders of a contracted folder.

Syntax
FolderExpand (FolderName: *string*)

Parameters
FolderName: *string*
Include full folder path.

See Also
FolderContract
FolderContractToOneLevel
FolderExpandAll

Route...
Select a folder, press Shift+Plus.

FolderExpandAll

Purpose
Displays all subfolders of a contracted folder.

Syntax
FolderExpandAllLevels (FolderName: *string*)

Parameters
FolderName: *string*
Include full folder path.

Route...
Select a folder, press Shift+Ctrl+Plus.

FolderLinkTo

F

Purpose

Links an item to one or more folders (up to 50). Linked items appear in more than one folder.

Syntax

FolderLinkTo (FolderName00: *string*; ... FolderName49: *string*)

Parameters

FolderName00: *string*; ... FolderName49: *string*
Include full folder path. Separate multiple folders with a semicolon.

See Also

FolderDeepMove
FolderMoveTo

Route...
Select a folder item, choose File, Move/Link to Folders, select one or more folders, choose Link.

FolderLinkToDlg

Purpose

Displays the In Box Selections, Out Box Selections, or Move/Link Selections to Folder dialog box, depending on the folder list.

Syntax

FolderLinkToDlg ()

See Also

FolderLinkTo

Route...
Select a folder item, choose File, Move/Link to Folders.

FolderListAddFolder

Purpose

Adds one or more folders to a folder list. Not recordable.

To add the root folder to an empty folder list, use FolderListAddFolder:

```
hList = FolderListCreate(Contents: Empty!)
FolderListAddFolder (FolderListHandle: hList; FolderName: "Root Folder Name")
```

Syntax

FolderListAddFolder (FolderListHandle: *numeric*; FolderName: *string*)

Parameters

FolderListHandle: *numeric*
Folder list handle, returned by FolderListCreate or FolderListCreateFromView.

FolderName: *string*
Include full folder path.

FolderListApplyToView

Purpose
Applies a folder list to the active view. Not recordable.

Syntax
FolderListApplyToView (FolderListHandle: *numeric*)

Parameters
FolderListHandle: *numeric*
Folder list handle, returned by FolderListCreate or FolderListCreateFromView.

FolderListCreate

Purpose
Creates a folder list. Not recordable.

F Delete a folder list handle when it is no longer needed, or system resources will be lost. See FolderListDelete.

Syntax
numeric FolderListCreate (Contents: *enum*; UserID: *string*)

Return Value
numeric Folder list handle if successful, 0 if not.

Parameters
Contents: *enum*
Create a folder list including all folders, no folders, or only the root folder.
All!
Empty!
Root!

If you create an Empty! folder list, call FolderListAddFolder before passing the folder list handle to ItemListCreate.

UserID: *string* (optional)
A user ID (root folder).

See Also
FolderListCreateFromView
ItemListCreate

FolderListCreateFromView

Purpose
Creates a folder list from one or more selected folders (active view). Not recordable.


Delete a folder list handle when it is no longer needed, or system resources will be lost. See FolderListDelete.

Syntax
numeric FolderListCreateFromView ()

Return Value
numeric Folder list handle if successful, 0 if not.

See Also
FolderListCreate

FolderListDelete

Purpose

Deletes a folder list handle from memory. Not recordable.

The folder list handle is returned by FolderListCreate or FolderListCreateFromView.

Syntax

FolderListDelete (FolderListHandle: *numeric*)

Delete the folder list handle after it is used to create an item list, or when it is no longer needed, or system resources will be lost.

FolderListGetCount

Purpose
Returns the number of folders in a folder list. Not recordable.

Syntax
numeric FolderListGetCount (FolderListHandle: *numeric*)

Return Value
numeric Number of folders.

Parameters
FolderListHandle: *numeric*
Folder list handle, returned by FolderListCreate or FolderListCreateFromView.

FolderListGetName

Purpose
Returns a folder name from a folder list. Not recordable.

Syntax
string FolderListGetName (FolderListHandle: *numeric*; Index: *numeric*; FullyQualified: *enum*)

Return Value
string Folder name.

Parameters
FolderListHandle: *numeric*
Folder list handle, retuned by FolderListCreate or FolderListCreateFromView.

Index: *numeric*
Index number (first folder equals 0).

FullyQualified: *enum*
Include full folder path.
No!
Yes!

FolderListSetFolder

Purpose

Replaces the contents of a folder list with one or more specified folders (creates a new list). Not recordable.

Syntax

FolderListSetFolder (FolderListHandle: *numeric*; FolderName: *string*)

Parameters

FolderListHandle: *numeric*
Handle of the folder list that has contents to replace. The handle is returned by FolderListCreate or FolderListCreateFromView.

FolderName: *string*
Folder name to add.

See Also

EnvUserID

FolderMoveTo

Purpose
Moves an item to one or more folders and removes it from the original folder.

Syntax
FolderMoveTo (FolderName00: *string*; ... FolderName49: *string*)

Parameters
FolderName00: *string*; ... FolderName49: *string*
Include full folder path. Separate multiple folders with a semicolon.

See Also
FolderDeepMove
FolderLinkTo
FolderSelect

Route...
Select a folder item, choose File, Move/Link to Folders, select one or more folders, choose Move.

FolderRename

Purpose
Renames a folder.

Syntax
FolderRename (FullFolderName: *string*; NewName: *string*)

Parameters
FullFolderName: *string*
Folder to rename. Include full path.

NewName: *string*
New folder name only. Do not include full path.

Route...
Select a folder, choose File, Folders, Rename, type a new folder name.

FolderRenameDlg

Purpose

Displays the Rename Folder dialog box. The command fails if a folder item is selected instead of a folder. Not recordable.

Syntax

FolderRenameDlg (FolderName: *string*)

Parameters

FolderName: *string* (optional)
If not specified, folder must be selected.

FolderSelect

Purpose
Selects up to 50 folders. Each new selection deselects the previous selection.

Syntax
FolderSelect (FolderName00: *string*; ... FolderName49: *string*)

Parameters
FolderName00: *string*; ... FolderName49: *string*
Include full folder path. Separate multiple folders with a semicolon.

See Also
FolderAddSelections
FolderMoveTo

Route...
Select a folder, press Shift+Up Arrow or Shift+Down Arrow.

FolderSelectDlg

Purpose
Displays the Folders dialog box.

Syntax
FolderSelectDlg ()

See Also
FolderSelect

Route...
Open an In Box, Out Box, Trash, or Calendar view, choose File, Folders.

FontBold

Purpose
Turns Bold on or off.

Syntax
FontBold (State: *enum*)

Parameters
State: *enum* (optional)
If State not specified, acts as a toggle.
On!
Off!

See Also
FontItalic
FontNormal
FontUnderline

Route...
Place the insertion point in a message edit box, choose Edit, Font, Bold.

FontDlg

Purpose
Displays the Font dialog box. Not recordable.

Syntax
FontDlg ()

See Also
FontSet

FontItalic

Purpose
Turns Italic on or off.

Syntax
FontItalic (State: *enum*)

Parameters
State: *enum* (optional)
If State not specified, acts as a toggle.
Off!
On!

See Also
FontBold
FontNormal
FontUnderline

Route...
Place the insertion point in a message edit box, choose Edit, Font, Italic.

FontNormal

Purpose
Turns off all current font attributes (bold, italic, underline).

Syntax
FontNormal ()

See Also
FontBold
FontItalic
FontUnderline

Route...
Place the insertion point in a message edit box, choose Edit, Font, Normal.

FontSet

Purpose

Specifies a font and font attributes.

To see the height and weight values of a specified font, record your actions as you specify a font and font attributes.

Syntax

FontSet (FaceName: *string*; Height: *numeric*; Weight: *numeric*; Underline: *enum*; Italic: *enum*; CharSet: *numeric*)

Parameters

FaceName: *string*
Font name.

Height: *numeric* (optional)
Font height in pixels (65535 - N where N is the height in pixels). Cell height: 0-32767. Character height, including ascent and descent: 32768-65535.

Weight: *numeric* (optional)
Font weight. Some fonts have only three weights: Normal, Regular, and Bold.

100 (THIN)
200 (EXTRATHIN)
300 (ULTRALIGHT)
400 (NORMAL)
400 (REGULAR)
500 (MEDIUM)
600 (SEMIBOLD)
600 (DEMIBOLD)
700 (BOLD)
800 (EXTRABOLD)
800 (ULTRABOLD)
900 (BLACK)
900 (HEAVY)

Underline: *enum* (optional)
Turn on attribute.

No!
Yes!

Italics: *enum* (optional)
Turn on attribute.

No!
Yes!

CharSet: *numeric*
Asian versions only.

See Also

FontBold
FontItalic

FontNormal
FontUnderline

Route...
Place the insertion point in a message edit box, choose Edit, Font, Font.

FontUnderline

Purpose
Turns Underline on or off.

Syntax
FontUnderline (State: *enum*)

Parameters
State: *enum* (optional)
If State not specified, acts as a toggle.
Off!
On!

See Also
FontBold
FontItalic
FontNormal

Route...
Place the insertion point in a message edit box, choose Edit, Font, Underline.

GetAddressBookData

Purpose

Returns Address Book information (backward compatibility only). Use Address Book commands to return Address Book information. Not recordable.

Syntax

any GetAddressBookData (MacroVariable: *variable*; Section: *enum*; Query: *enum*; FieldNum: *numeric*; UserIDOrGroupName: *string*; Disambiguate: *enum*)

Return Value

any Address Book information.

Parameters

MacroVariable: *variable*
Output variable assigned the return value of the Query parameter.

Section: *enum*
Address Book section to query.

- ExternalAddress!
- PersonalGroups!
- PublicGroups!
- Resources!
- Users!

ExternalAddress! is for other mail systems.

Query: *enum*
Query type.

- Entry!
- Field!
- FieldCount!

Entry! returns field contents. Field! returns the field name. FieldCount! returns the number of fields in a specified section.

FieldNumber: *numeric*
Field number to query. The following list represents possible field names and numbers for each section. The field names on your system may be different. Contact your system administrator for more information. Section names are followed by a list of field numbers and names.

Users! (Field number field name)

0	First Name
1	Last Name
2	Mail
3	Dept
4	Phone
5	User ID
6	Post Office
7	Domain
8	Network ID

9 File ID
10 Fax Number

Resources!
0 Resource ID
1 Post Office
2 Domain
3 Owner
4 File ID
5 Type
6 Description

PublicGroups!
0 Group ID
1 Post Office
2 Domain
3 Description

PersonalGroups!
0 Group ID

UserIDOrGroupName: *string*
User ID or group name to query.

Disambiguate: *enum*
Choose the correct user when two or more users have the same ID.
No!
Yes!

No! redirects macro execution to an ONERROR label. The macro fails if an ONERROR label is not specified. Yes! displays the list of users and prompts you to choose one.

See Also
GetOfficeData

GetOfficeData

Purpose

Returns GroupWise, In Box, Out Box, or Trash information (backward compatibility only). Use environment (Env) commands to return system information. Not recordable.

Syntax

any GetOfficeData (MacroVariable: *variable*; SystemVariable: *enum*; IndexNumber: *numeric*)

Return Value

any GroupWise information.

Parameters

MacroVariable: *variable*
Requested information is returned in this variable (see SystemVariable parameter).

SystemVariable: *enum*
Returns a value for various types of Groupwise data in a variable (see MacroVariable parameter).

- ClipboardText!
- InterimVersion!
- IsItemAccepted!
- IsItemCompleted!
- IsItemPrivate!
- IsItemRouted!
- IsListItemAccepted!
- IsListItemCompleted!
- IsListItemDelegated!
- IsListItemPrivate!
- IsListItemRead!
- IsListItemRouted!
- IsViewPersonal!
- ItemAttachmentCount!
- ItemAttachmentName!
- ItemCreateDate!
- ItemPriority!
- ListCount!
- ListItemCreateDate!
- ListItemFrom!
- ListItemHasAttachments!
- ListItemPriority!
- ListItemSubject!
- ListItemType!
- MacroPath!
- MajorVersion!
- MinorVersion!
- PostOfficePath!
- UserId!
- UserName!
- ViewType!

IndexNumber: *numeric* (optional)
Index number (zero-based) of In Box, Out Box, or Trash items.

See Also

GetAddressBookData

HelpCoaches

Purpose
Displays the Coach dialog box.

Syntax
HelpCoaches ()

Route...
Choose Help, Coach.

HelpContents

Purpose

Displays the GroupWise Help Contents screen.

Syntax

HelpContents ()

Route...
Choose Help, Contents.

HelpContextSensitive

Purpose

Displays help information for a selected item or control.

Syntax

HelpContextSensitive ()

Route...
Choose Help, What Is, select an item or control.

HelpHowDoI

Purpose
Displays the How Do I help screen.

Syntax
HelpHowDoI ()

Route...
Choose Help, How Do I.

HelpMacros

Purpose

Displays the Online Macros Manual.

Syntax

HelpMacros ()

Route...
Choose Help, Macros.

HelpSearchFor

Purpose

Displays the Search dialog box to look up GroupWise Help topics.

Syntax

HelpSearchFor ()

Route...
Choose Help, Search for Help on.

HelpWhatIs

Purpose
Displays Help information on an item when the item is clicked.

Syntax
HelpWhatIs()

See Also
HelpContextSensitive

Route...
Choose Help, What Is.

IMEOff

Purpose

Turns off the Input Method Editor in Asian versions of GroupWise. Not recordable.

Syntax

IMEOff ()

IMEOn

Purpose

Turns on the Input Method Editor in Asian versions of GroupWise. Not recordable.

Syntax

IMEOn ()

IMEWordRegister

Purpose

Adds a selected word and its reading (phonetic representation) into an Input Method Editor dictionary. Not recordable.

Syntax

IMEWordRegister ()

ImportCalendar

Purpose

Imports memos, personal appointments, and tasks from Office 3.1 Calendar into GroupWise 4.1 Calendar views.

Syntax

ImportCalendar (Filename: *string*)

Parameters

Filename: *string*

Calendar filenames are usually named xxxcal.fil (xxx represent user inititals).

Route...
From the Main Window, choose File, Import Calendar, type the full path of a calendar file.

ImportCalendarDlg

F

Purpose
Displays the Import Calendar dialog box.

Syntax
ImportCalendarDlg ()

See Also
ImportCalendar

Route...
From the Main Window, choose File, Import Calendar.

InfoUpdate

Purpose

Updates an item's Information view. Not recordable.

Syntax

InfoUpdate ()

See Also

Refresh

InhibitInput

Purpose

Turns keyboard input on or off during macro execution. Not recordable.

Syntax

InhibitInput (State: *enum*)

Parameters

State: *enum*

Default: Off!

Off!

On!

Insert

Purpose

Turns Insert text on or off.

Syntax

Insert (Insert: *enum*)

Parameters

Insert: *enum* (optional)

If no parameter is specified, acts as a toggle.

Off!
On!

Off! types over text.

Route...
Press Insert.

InsertTab

Purpose
Inserts a tab.

Syntax
InsertTab ()

Route...
Place the insertion point in an edit box, Press Ctrl+Tab.

ItemAccept

Purpose
Accepts an Appointment, Task, or Note.

Syntax
ItemAccept (Comment: *string*)

Parameters
Comment: *string* (optional)
Response to an accepted item.

Route...
Open an Appointment, Task, or Note item, choose Actions, Accept with Comment, type comment.

ItemAcceptWithCommentDlg

Purpose

Displays the Accept With Comment dialog box.

Syntax

ItemAcceptWithCommentDlg ()

See Also

ItemAccept

Route...
Open an Appointment, Task, or Note item, choose Actions, Accept with Comment.

ItemAnnotationGetCount

Purpose
Returns the number of sound annotations on an item view. Not recordable.

Parameters
numeric ItemAnnotationGetCount (MessageID: *string*)

Return Value
numeric Number of sound annotations.

Parameters
MessageID: *string*
Unique item identifier, returned by ItemMessageIDFromView.

ItemAnnotationSaveAs

Purpose
Saves a sound annotation to a file. Not recordable.

Syntax
ItemAnnotationSaveAs (MessageID: *string*; Index: *numeric*; Filename: *string*)

Parameters
MessageID: *string*
Unique item identifier, returned by ItemMessageIDFromView.

Index: *numeric*
Index number (zero-based) of the sound annotation.

Filename: *string*

ItemArchive

Purpose
Saves an item to the archive database. Not recordable.

Syntax
ItemArchive (MessageID: *string*)

Parameters
MessageID: *string* (optional)
Unique item identifier, returned by ItemMessageIDFromView. If not specified, archives the selected item.

ItemArchiveOpenItem

Purpose

Saves one or more selected items to the archive database.

Syntax

ItemArchiveOpenItem ()

Route...
Select one or more items, choose Actions, Archive.

ItemAttachmentAdd

Purpose

Adds an attachment to an item. Not recordable.

Syntax

ItemAttachmentAdd (MessageID: *string*; AttachmentClass: *enum*; AttachmentName: *string*; AttachmentDisplayName: *string*)

Parameters

MessageID: *string*
Unique item identifier, returned by ItemMessageIDFromView.

AttachmentClass: *enum*
Type of attachment.
AttachClassFile!
AttachClassMessage!

AttachmentName: *string*
Name of an attachment.

AttachmentDisplayName: *string* (optional)
Name displayed in the Attach File box.

ItemAttachmentDelete

Purpose

Deletes an attachment. Not recordable.

Syntax

ItemAttachmentDelete (MessageID: *string*; Index: *numeric*)

Parameters

MessageID: *string*
Unique item identifier, returned by ItemMessageIDFromView.

Index: *numeric*
Attachment index number (zero-based).

ItemAttachmentGetClass

Purpose
Returns an attachment class. Not recordable.

Syntax
numeric ItemAttachmentGetClass (MessageID: *string*; Index: *numeric*)

Return Value
numeric Attachment class. The values are:
1 File
2 Encapsulated item
3 Embedded or linked object (OLE)

Parameters
MessageID: *string*
Unique item identifier, returned by ItemMessageIDFromView.

Index: *numeric*
Attachment index number (zero-based).

ItemAttachmentGetCount

Purpose
Returns the number of attachments. Not recordable.

Syntax
numeric ItemAttachmentGetCount (MessageID: *numeric*)

Return Value
numeric Number of attachments.

Parameters
MessageID: *numeric*
Unique item identifier, returned by ItemMessageIDFromView.

ItemAttachmentGetCurrentIndex

Purpose
Returns the current attachment index. Not recordable.

Syntax
numeric ItemAttachmentGetCurrentIndex ()

Return Value
numeric Current attachment index (zero-based).

ItemAttachmentGetDisplayName

Purpose
Returns the display name of an attachment. Not recordable.

Syntax
string ItemAttachmentGetDisplayName (MessageID: *string*; Index: *numeric*)

Return Value
string Display name, which may be different from the attachment filename.

Parameters
MessageID: *string*
Unique item identifier, returned by ItemMessageIDFromView.

Index: *numeric*
Attachment index number (zero-based).

ItemAttachmentGetName

Purpose
Returns the name of an attachment. Not recordable.

Syntax
string ItemAttachmentGetName (MessageID: *string*; Index: *numeric*)

Return Value
string Attachment name.

Parameters
MessageID: *string*
Unique item identifier, returned by ItemMessageIDFromView.

Index: *numeric*
Attachment index number (zero-based).

ItemAttachmentSaveAs

Purpose

Saves an attachment to a specified directory, or default macros directory. Not recordable.

Syntax

ItemAttachmentSaveAs (MessageID: *string*; Index: *numeric*; Filename: *string*)

Parameters

MessageID: *string*
Unique item identifier, returned by ItemMessageIDFromView.

Index: *numeric*
Index number (zero-based) of attachment to save.

Filename: *string*
Attachment name. Path is optional.

ItemAttachmentUpdate

Purpose
Replaces one attachment with another. Not recordable.

Syntax
ItemAttachmentUpdate (MessageID: *string*; Index: *numeric*; Filename: *string*)

Parameters
MessageID: *string*
Unique item identifier, returned by ItemMessageIDFromView.

Index: *numeric*
Index number (zero-based) of the attachment to replace.

Filename: *string*
Replacement attachment.

ItemComplete

Purpose
Marks a task or routed item done or not done.

Syntax
ItemComplete (Accept: *enum*)

Parameters
Accept: *enum* (optional)
If no parameter is specified, acts as a toggle.
Done!
NotDone!

NotDone! unmarks a task or routed item.

Route...
Open a task, select or deselect the Completed check box on the status bar.

ItemCustomReplyDlg

F

Purpose
Displays the Reply dialog box to set custom reply options.

Syntax
ItemCustomReplyDlg ()

See Also
ItemReply

Route...
Open and In Box item, choose Send, Reply.

ItemDecline

Purpose

Declines an appointment, task, or note, and attaches a comment. Not recordable.

Syntax

ItemDecline (Comment: *string*; MessageID: *string*; AllInstances: *enum*)

Parameters

Comment: *string* (optional)
Comment is displayed in an item's Information view.

MessageID: *string* (optional)
Unique item identifier, returned by ItemMessageIDFromView. If not specified, declines the selected item.

AllInstances: *enum* (optional)

- No!
- Yes!

See Also

ItemReply

ItemDeclineOpenItem

Purpose

Declines a selected or opened appointment, task, or note, and attaches a comment.

Syntax

ItemDeclineOpenItem (Comment: *string*)

Parameters

Comment: *string* (optional)
Comment is displayed in an item's Information view.

Route...
Open an appointment, task, or note, choose Actions, Decline with Comment, type comment.

ItemDeclineWithCommentDlg

Purpose

Displays the Decline with Comment dialog box.

Syntax

ItemDeclineWithCommentDlg ()

Route...
Open an appointment, task, or note, choose Actions, Decline with Comment.

ItemDelegate

Purpose

Delegates an appointment, task, or note, and attaches a comment. Not recordable.

Syntax

ItemDelegate (ToUserID: *string*; RecipComments: *string*; SenderComments: *string*; MessageID: *string*; AllInstances: *enum*)

Parameters

ToUserID: *string* (optional)
User ID.

RecipComments: *string* (optional)
Comments to the recipient (as attachment).

SenderComments: *string* (optional)
Comments to the sender (Information view).

MessageID: *string* (optional)
Unique item identifier, returned by ItemMessageIDFromView.

AllInstances: *enum* (optional)

No!
Yes!

ItemDelegateDlg

Purpose
Displays the Delegate dialog box.

Syntax
ItemDelegateDlg ()

Route...
Select or open an appointment, task, or note, choose Actions, Delegate.

ItemDelegateOpenItem

F

Purpose
Delegates an appointment, task, or note, and attaches comments.

Syntax
ItemDelegateOpenItem (ToUserID: *string*; RecipComments: *string*; SenderComments: *string*)

Parameters
ToUserID: *string*
User ID.

RecipComments: *string* (optional)
Comments to the recipient (as attachment).

SenderComments: *string* (optional)
Comments to the sender (Information view).

Route...
Select or open an appointment, task, or note, choose Actions, Delegate, type information in To and comment edit boxes.

ItemDelete

Purpose

Deletes an item from the In Box or Out Box. Not recordable.

Syntax

ItemDelete (MessageID: *string*; EmptyItem: *enum*; FolderListHandle: *numeric*; AllInstances: *enum*; Retract: *enum*)

Parameters

MessageID: *string* (optional)
Unique item identifier, returned by ItemMessageIDFromView. If not specified, deletes the selected item. If specified, you must also specify EmptyItem.

EmptyItem: *enum* (optional)
Do not place deleted item in the Trash. Default: No!.

- No!
- Yes!

FolderListHandle: *numeric* (optional)
Handle of a folder list. If not specified, deletes the item from all folders. Handle is returned in a variable by FolderListCreate or FolderListCreateFromView.

AllInstances: *enum* (optional)
Delete all instances of autodate item. Default: No!.

- No!
- Yes!

Retract: *enum* (optional)
Retract Out Box item. Default: ThisOutbox!.

- AllBoxes!
- AllInBoxes!
- ThisOutbox!

ItemDeleteOpenItem

Purpose
Deletes a selected or opened In Box or Out Box item.

Syntax
ItemDeleteOpenItem ()

Route...
Select or open an item, Choose Edit or Actions, Delete.

ItemFolderAltMove

Purpose

Moves an item to the folders in a folder list. Not recordable.

Syntax

ItemFolderAltMove (MessageID: *string*; FolderListHandle: *numeric*)

Parameters

MessageID: *string*
Unique item identifier, returned by ItemMessageIDFromView.

FolderListHandle: *numeric*
Handle of the destination folder list.

ItemFolderLink

Purpose
Links an item to the folders in a folder list. Not recordable.

Syntax
ItemFolderLink (MessageID: *string*; FolderListHandle: *numeric*)

Parameters
MessageID: *string*
Unique item identifier

FolderListHandle: *numeric*
Handle of the destination folder list.

ItemFolderMove

Purpose

Moves an item from one folder list to another. Not recordable.

Syntax

ItemFolderMove (MessageID: *string*; FolderListHandleDest: *numeric*; FolderListHandleSrc: *numeric*)

Parameters

MessageID: *string*
Unique item identifier, returned by ItemMessageIDFromView.

FolderListHandleDest: *numeric*
Handle of the destination folder list, returned by FolderListCreate.

FolderListHandleSrc: *numeric*
Handle of the source folder list, returned by FolderListCreate.

ItemForward

Purpose
Forwards a selected or opened In Box item to one or more users.

Syntax
ItemForward ()

See Also
ItemDelegate

Route...
Select or open an item, choose Send, Forward.

ItemGetAttribute

Purpose
Returns True or False, depending on the state of a specified attribute. Not recordable.

Syntax
boolean ItemGetAttribute (MessageID: *string*; Attribute: *enum*)

Return Value
boolean True/False.

Parameters
MessageID: *string*
Unique item identifier, returned by ItemMessageIDFromView.

Attribute: *enum*
- Accepted!
- Archived!
- Autodate!
- Completed!
- CustomView!
- Delegated!
- Deleted!
- Forwarding!
- Opened!
- Personal!
- PhoneCalled!
- PhoneCameToSeeYou!
- PhonePleaseCall!
- PhoneReturnedYourCall!
- PhoneUrgent!
- PhoneWantsToSeeYou!
- PhoneWillCallAgain!
- Private!
- Read!
- ReplyRequested!
- Replying!
- Resending!
- ReturnReceiptRequested!
- Routed!
- RoutingEndOfLine!

ItemGetDate

Purpose
Returns the date specified by the DateType parameter. Not recordable.

Syntax
string ItemGetDate (MessageID: *string*; DateType: *enum*; DateFormatType: *enum*)

Return Value
string Date.

Parameters
MessageID: *string*
Unique item identifier, returned by ItemMessageIDFromView.

DateType: *enum*
Type of date information.
- AssignedDate!
- BeginDateTime!
- CreateDate!
- DueDate!
- Duration!
- EndDateTime!

DateFormatType: *enum*
- MAPIDate!
- MailboxDate!

ItemGetMailboxID

Purpose
Returns the mailbox ID of an item's primary recipient. Not recordable.

Syntax
string ItemGetMailboxID (MessageID: *string*)

Return Value
string Mailbox ID in the form: Domain.PostOffice.ID.

Parameters
MessageID: *string*
Unique item identifier, returned by ItemMessageIDFromView.

ItemGetOutboxMessageID

Purpose
Returns the message ID of an Out Box item. Not recordable.

Syntax
string ItemGetOutboxMessageID (MessageID: *string*)

Return Value
string Message ID. ItemGetOutboxMessageID returns an "undefined function return value" if the MessageID parameter identifies a personal item, or an In Box item without a corresponding Out Box item.

Parameters
MessageID: *string*
Unique item identifier, returned by ItemMessageIDFromView.

ItemGetPriority

Purpose
Returns the priority setting of an item. Not recordable.

Syntax
numeric ItemGetPriority (MessageID: *string*)

Return Value
numeric 1 equals low; 2 equals medium; 3 equals high.

Parameters
MessageID: *string*
Unique item identifier, returned by ItemMessageIDFromView.

ItemGetReplyToMessageID

Purpose

Returns the message ID of the item that is being replied to. Not recordable.

Open a message and choose Reply. This is the only view to call this command from.

Syntax

string ItemGetReplyToMessageID (MessageID: *string*)

Return Value

string Message ID.

Parameters

MessageID: *string*
Message ID of the reply view ("X00").

ItemGetSenderID

Purpose
Returns the user ID of an item's sender. Not recordable.

Syntax
string ItemGetSenderID (MessageID: *string*)

Return Value
string A user ID in the form: Domain.PostOffice(UserID).

Parameters
MessageID: *string*
Unique item identifier, returned by ItemMessageIDFromView.

ItemGetSource

Purpose
Returns the location of a specified item. Not recordable.

Syntax
numeric ItemGetSource (MessageID: *string*)

Return Value
numeric 1 equals In Box; 2 equals Out Box; 3 equals Personal. Command fails on encapsulated items.

Parameters
MessageID: *string*
Unique item identifier, returned by ItemMessageIDFromView.

ItemGetText

Purpose
Returns the contents of a text field (edit box). Not recordable.

Syntax
string ItemGetText (MessageID: *string*; Field: *enum*)

Return Value
string Contents of a text field, or empty string if the text field is empty.

Parameters
MessageID: *string*
Unique item identifier, returned by ItemMessageIDFromView.

Field: *enum*
Text fields.

- Authority!
- BC!
- CC!
- Caller!
- Company!
- From!
- Message!
- Phone!
- Place!
- Subject!
- TaskCategory!
- TaskPriority!
- To!
- ViewName!

BC! only retrieves blind copy information from an Out Box when you are the sender or have proxy rights.

ItemGetType

Purpose
Returns an item's type. Not recordable.

Syntax
numeric ItemGetType (MessageID: *string*)

Return Value
numeric 1 equals Appointment; 2 equals Mail; 3 equals Note; 4 equals Phone Message; and 5 equals Task.

Parameters
MessageID: *string*
Unique item identifier, returned by ItemMessageIDFromView.

ItemInfo

Purpose
Displays an item Information view. Not recordable.

Syntax
ItemInfo (MessageID: *string*)

Parameters
MessageID: *string*
Unique item identifier, returned by ItemMessageIDFromView.

See Also
ItemInfoOpenItem

ItemInfoOpenItem

Purpose
Displays the item Information view of a selected item.

Syntax
ItemInfoOpenItem ()

See Also
ItemInfo

Route...
Select an item, choose Actions, Info.

ItemIsValid

Purpose

Returns True if an item is valid, False if not. Not recordable.

An item is invalid after it is deleted, or when it is associated with an unproxied user.

Syntax

boolean ItemIsValid (MessageID: *string*)

Return Value

boolean True/False.

Parameters

MessageID: *string*
Unique item identifier, returned by ItemMessageIDFromView.

ItemListCreate

Purpose
Creates an item list from a current, filter, or folder list. Not recordable.

Syntax
numeric ItemListCreate (UserID: *string*; FilterHandle: *numeric*; FolderListHandle: *numeric*)

Return Value
numeric Item list handle.

Delete an item list handle when no longer needed, or system resources will be lost. See ItemListDelete.

Parameters
UserID: *string* (optional)
User network ID.

FilterHandle: *numeric* (optional)
Filter list handle.

FolderListHandle: *numeric* (optional)
Folder list handle.

ItemListCreateFromControl

Purpose
Creates an item list from a control, such as an Appointment list. Not recordable.

Syntax
numeric ItemListCreateFromControl (ControlListType: *enum*; SelectedItemsOnly: *enum*; ControlName: *string*)

Return Value
numeric Item list handle.


Delete an item list handle when no longer needed, or system resources will be lost. See ItemListDelete.

Parameters
ControlListType: *enum*
- AppointmentList!
- ItemList!
- NoteList!
- SelectedControl!
- TaskList!
- WeekControl!

Apply AppointmentList!, NoteList!, and TaskList! to Calendar views. Apply ItemList! to In Box, Out Box, or Trash views. Apply WeekControl! to a Week Calendar view. Apply SelectedControl! to all item views.

SelectedItemsOnly: *enum* (optional)
Default: No!.
- No!
- Yes!

If Yes! is specified, the Index parameter of the ItemListGetItem command must be 0.

ControlName: *string* (optional)
Custom control name.

ItemListDelete

Purpose
Deletes an item list handle from memory. Not recordable.

Syntax
ItemListDelete (Handle: *numeric*)

Parameters
Handle: *numeric*
Item list handle.

See Also
ItemListCreate
ItemListCreateFromControl

ItemListGetCount

Purpose
Returns the number of items in a list. Not recordable.

Syntax
numeric ItemListGetCount (Handle: *numeric*)

Return Value
numeric Number of list items.

Parameters
Handle: *numeric*
Item list handle.

ItemListGetItem

Purpose

Returns the message ID of a list item. Not recordable.

ItemListCreate or ItemListCreateFromControl creates a list and returns an Item list handle.

Syntax

string ItemListGetItem (Handle: *numeric*; Index: *numeric*)

Return Value

string Message ID.

Parameters

Handle: *numeric*
Item list handle.

Index: *numeric*
Index number (zero-based) of a list item.

ItemListSort

Purpose

Sorts an item list. Not recordable.

Syntax

ItemListSort (Handle: *numeric*; SortKey: *enum*; SortOrder: *enum*)

Parameters

Handle: *numeric*
Item list handle.

SortKey: *enum*
Item field to sort.
- AssignedDate!
- BeginDateTime!
- CreateDate!
- DueDate!
- EndDateTime!
- From!
- ItemType!
- Subject!
- TaskCategory!
- TaskPriority!
- To!

SortOrder: *enum*
- Ascending!
- Descending!

ItemMarkPrivate

Purpose
Marks private the opened item or all selected items.

Syntax
ItemMarkPrivate (Private: *enum*)

Parameters
Private: *enum*
If no parameter is specified, acts as a toggle.
Yes!
No!

No! unmarks an item.

Route...
Open an item, choose Actions, Mark Private.

ItemMessageIDFromView

Purpose
Returns the message ID of the active view. Not recordable.

Syntax
string ItemMessageIDFromView ()

Return Value
string Message ID. If the item is being created, the message ID is always "X00".

ItemOpen

Purpose

Opens the item with a corresponding MessageID (see MessageID parameter). Not recordable.

Syntax

ItemOpen (MessageID: *string*)

Parameters

MessageID: *string*
Unique item identifier, returned by ItemMessageIDFromView.

ItemRead

Purpose
Opens a selected item to read.

Syntax
ItemRead ()

See Also
ItemReadNext
ItemReadPrevious

Route...
Select an item, choose Actions, Read.

ItemReadNext

Purpose
Opens the item after an already opened item.

Syntax
ItemReadNext ()

See Also
ItemRead
ItemReadPrevious

Route...
Open an item, choose Actions, Read Next.

ItemReadPrevious

Purpose
Opens the item before an already opened item.

Syntax
ItemReadPrevious ()

See Also
ItemRead
ItemReadNext

Route...
Open an item, choose Actions, Read Previous.

ItemReply

F

Purpose
Opens a reply view for a selected or opened item.

Syntax
ItemReply (ReplyTo: *enum*; IncludeTxt: *enum*)

Parameters
ReplyTo: *enum* (optional)
Reply to the sender or to the sender and all recipients. Default: Sender only.
All!
Sender!

IncludeTxt: *enum* (optional)
Include the original message. Default: No!.
No!
Yes!

See Also
ItemCustomReplyDlg

Route...
Select an item, choose Send, Reply.

ItemResend

Purpose
Resends a selected or opened Out Box item.

Syntax
ItemResend (RetractOriginal: *enum*)

Parameters
RetractOriginal: *enum* (optional)
Prompt if not specified.
No!
Yes!

Route...
Select an Out Box item, choose Send, Resend.

ItemRoute

F

Purpose
Routes a message or task to different users in turn.

Syntax
ItemRoute (DoRouting: *enum*)

Parameters
DoRouting: *enum* (optional)
Turn on routing. If no parameter is not specified, acts as a toggle.
No!
Yes!

See Also
ItemResend

Route...
Open a mail or task item, choose Send, Routing Slip.

ItemSaveInfo

Purpose

Saves item information to a specified file. Item information includes recipient names, routes, file sizes, creation date, and security options. Not recordable.

Syntax

ItemSaveInfo (MessageID: *string*; Filename: *string*; FileFormat: *enum*)

Parameters

MessageID: *string*
Unique item identifier, returned by ItemMessageIDFromView.

Filename: *string*
Path and name of a file.

FileFormat: *enum*
AnsiText!
WordPerfect60!

ItemSaveMessage

Purpose
Saves the information contained in an item view to a specified file. Not recordable.

Syntax
ItemSaveMessage (MessageID: *string*; Filename: *string*; FileFormat: *enum*)

Parameters
MessageID: *string*
Unique item identifier, returned by ItemMessageIDFromView.

Filename: *string*
Include full path.

FileFormat: *enum*
AnsiText!
WordPerfect60!

ItemSaveMessageDlg

Purpose

Displays the Save - Message dialog box of an item view, or the Save Message Entries dialog box of an In Box, Out Box, or Calendar view (selected or opened).

Syntax

ItemSaveMessageDlg ()

Route...
Select an item, choose File, Save.

ItemSaveView

Purpose
Saves a custom view to a file. Not recordable.

Parameters
ItemSaveView (MessageID: *string*; Filename: *string*)

Parameters
MessageID: *string*
Unique item identifier, returned by ItemMessageIDFromView.

Filename: *string*
Include full path.

ItemSaveViewDlg

Purpose
Displays the Save - View dialog box.

Syntax
ItemSaveViewDlg ()

Route...
Open an item, choose File, Save View.

ItemSend

Purpose

Sends a new item. The item must be open.

Syntax

ItemSend ()

Route...
Open a new item, choose Send, Send.

ItemSetAlarm

Purpose

Sets an alarm for the appointment specified by MessageID. Not recordable.

This command fails if the item identified by MessageID is not an appointment or if the appointment is past.

Syntax

ItemSetAlarm (MessageID: *string*; HoursBefore: *numeric*; MinutesBefore: *numeric*; ProgramToLaunch: *string*)

Parameters

MessageID: *string*
Unique item identifier, returned by ItemMessageIDFromView.

HoursBefore: *numeric*

MinutesBefore: *numeric*

ProgramToLaunch: *string* (optional)
Path and name of a program to launch when the alarm goes off.

ItemSetAttribute

Purpose

Sets an attribute for the item specified by MessageID. Not recordable.

You can only set attributes under certain conditions. For example, you can accept or decline scheduled items (appointments, tasks, and notes), but not mail or phone items. When using this command, correctly match attributes and conditions. See ModifyAttribute parameter for conditions.

Syntax

ItemSetAttribute (MessageID: *string*; ModifyAttribute: *enum*; AttributeValue: *enum*)

Parameters

MessageID: *string*
Unique item identifier, returned by ItemMessageIDFromView.

ModifyAttribute: *enum*
Apply attributes to specified item conditions. The conditions are: creating personal item, creating group item, existing personal item, existing group item.

- Accepted! (No, No, No, Yes)
- Completed! (No, No, Yes, Yes)
- Private! (Yes, Yes, Yes, Yes)
- Read! (No, No, Yes, Yes)

AttributeValue: *enum*
Set attribute.

- No!
- Yes!

ItemSetDate

Purpose

Sets the start and/or end date, or duration of Tasks, Appointments, and Notes. Not recordable.

Syntax

ItemSetDate (MessageID: *string*; StartDay: *numeric*; StartMonth: *numeric*; StartYear: *numeric*; StartMinute: *numeric*; StartHour: *numeric*; EndDay: *numeric*; EndMonth: *numeric*; EndYear: *numeric*; EndMinute: *numeric*; EndHour: *numeric*; DurationMinutes: *numeric*; DurationHours: *numeric*)

Parameters

MessageID: *string*
Unique item identifier, returned by ItemMessageIDFromView.

StartDay: *numeric* (optional)

StartMonth: *numeric* (optional)

StartYear: *numeric* (optional)

StartMinute: *numeric* (optional)

StartHour: *numeric* (optional)

EndDay: *numeric* (optional)

EndMonth: *numeric* (optional)

EndYear: *numeric* (optional)

EndMinute: *numeric* (optional)

EndHour: *numeric* (optional)

DurationMinutes: *numeric* (optional)

DurationHours: *numeric* (optional)

ItemSetItemType

Purpose
Specifies an item type for a new item. Not recordable.

Syntax
ItemSetItemType (MessageID: *string*; ItemType: *enum*; IsPersonal: *enum*)

Parameters
MessageID: *string*
Unique item identifier, returned by ItemMessageIDFromView.

ItemType: *enum*
New item type.
- Appointment!
- Mail!
- Note!
- Phone!
- Task!

IsPersonal: *enum* (optional)
- No!
- Yes!

ItemSetPriority

Purpose
Assign priority to a new item. Not recordable.

Syntax
ItemSetPriority (MessageID: *string*; Priority: *enum*)

Parameters
MessageID: *string*
Unique item identifier, returned by ItemMessageIDFromView.

Priority: *enum*
- High!
- Low!
- Normal!

ItemSetText

Purpose

Appends to, or replaces the text of, a specified field. Not recordable.

Syntax

ItemSetText (MessageID: *string*; Field: *enum*; FieldText: *string*; AppendText: *enum*)

Parameters

MessageID: *string*
Unique item identifier, returned by ItemMessageIDFromView.

Field: *enum*
- Authority!
- BC!
- CC!
- Caller!
- Company!
- From!
- Message!
- Phone!
- Place!
- Subject!
- TaskCategory!
- TaskPriority!
- To!
- ViewName!

FieldText: *string*
Replacement text.

AppendText: *enum*
Default: No!.
- No!
- Yes!

No! replaces text.

ItemUndelete

Purpose
Restores an item from Trash. Not recordable.

Syntax
ItemUndelete (MessageID: *string*)

Parameters
MessageID: *string*
Unique item identifier, returned by ItemMessageIDFromView.

See Also
ItemUndeleteOpenItem

ItemUndeleteOpenItem

Purpose
Restores a selected item from Trash.

Syntax
ItemUndeleteOpenItem ()

See Also
ItemUndelete

Route...
Select a Trash item, choose Edit, Undelete.

MacroFilePlay

Purpose

Plays a macro.

If this command is called from a DDE client, macro execution is asynchronous (control returns to the DDE client before the macro plays). Use EnvIsMacroPlaying to determine if a macro is playing.

Syntax

MacroFilePlay (Filename: *string*)

Parameters

Filename: *string*
Include full path.

See Also

EnvIsMacroPlaying
MacroPlayDlg
MacroStop

Route...
Choose Tools, Macro, Play, type a macro filename.

MacroPause

Purpose

Pauses a macro (playing or recording). Not recordable.

This command acts as a toggle. If called a second time, the macro resumes playing or recording.

Syntax

MacroPause ()

MacroPlayDlg

Purpose
Displays the Play Macro dialog box.

Syntax
MacroPlayDlg ()

Route...
Choose Tools, Macro, Play.

MacroRecordDlg

Purpose
Displays the Record Macro dialog box.

Syntax
MacroRecordDlg ()

Route...
Choose Tools, Macro, Record.

MacroStop

Purpose

Ends a macro (playing or recording). Not recordable.

Syntax

MacroStop ()

MainWindowHide

Purpose

Hides the Main Window.

Cannot be the only open view. If there is only one other active view, and you close it while the Main Window is hidden, GroupWise closes.

Syntax

MainWindowHide ()

See Also

CloseWindow

Route...
From the Main Window, and with at least one view open, choose File, Hide Window.

MainWindowShow

Purpose

Shows the Main Window when it is hidden, and activates the Main Window if it is behind an item view (not displayed), or maximizes the Main Window if it is minimized.

Syntax

MainWindowShow ()

See Also

MainWindowHide

Route...
Open a view, choose Window, Main Window.

MoviePause

Purpose
Pauses a multi-media graphic running in the graphics box of a customized view.

Syntax
MoviePause ()

See Also
MovieResume

Route...
Run a multi-media graphic, click the right mouse button, choose Pause.

MovieResume

F

Purpose

Resumes play of a multi-media graphic, paused in the graphics box of a customized view.

Syntax

MovieResume ()

See Also

MoviePause

Route...
Pause a multi-media graphic, click the right mouse button, choose Resume.

NewAppointment

Purpose

Creates a new appointment item. Not recordable.

Syntax

NewAppointment ()

NewMail

Purpose

Creates a new Mail item. Not recordable.

Syntax

NewMail ()

NewNote

Purpose

Creates a new Note item. Not recordable.

Syntax

NewNote ()

NewPhone

Purpose

Creates a new Phone message item. Not recordable.

Syntax

NewPhone ()

NewTask

Purpose

Creates a new Task item. Not recordable.

Syntax

NewTask ()

NextPane

Purpose

Advances the input focus to the next pane in the Main Window.

Syntax

NextPane ()

Route...
In the Main Window, press Tab.

OfficeCloseViews

Purpose
Closes all views except the Main Window.

Syntax
OfficeCloseViews ()

Route...
With the Main Window and at least one view open, choose Window, Close All.

OfficeMinimize

Purpose

Minimizes all GroupWise windows and views to an icon.

Syntax

OfficeMinimize ()

Route...
Choose Windows, Minimize GroupWise.

OleAttachDlg

Purpose

Displays the Attach Object dialog box.

Syntax

OleAttachDlg ()

Route...
Open a new item view, choose Edit, Attach Object.

OleAttachInsertObject

F

Purpose

Opens an application associated with an OLE object.

To attach the object, choose File, Update or Update GroupWise depending on the application. The recipient can open, view, edit, and/or save an attached OLE object.

Syntax

OleAttachInsertObject (ObjectClass: *string*)

Parameters

ObjectClass: *string*
Object to attach, such as a WordPerfect 6.0 object.

See Also

OleAttachPaste

Route...
Open a new item view, choose Edit, Attach Object, select an object, choose Attach New.

OleAttachPaste

Purpose
Attaches the contents of the clipboard to a new item, as an OLE object.

Syntax
OleAttachPaste ()

See Also
OleAttachInsertObject

Route...
Open a new item view, choose Edit, Attach Object, select Attach From clipboard, choose Paste.

OleConvertToStatic

Purpose

Converts an OLE embedded object so that it cannot be edited by the recipient.

Inserted objects are embedded in OLE boxes in custom views. Create custom views with View Designer, a program (vewin.exe) that comes with GroupWise.

Syntax

OleConvertToStatic ()

Route...
Open a custom view that has an OLE box, select the OLE box, choose Edit, Convert to Static.

OleDoVerb

F

Purpose

Performs an action on an attached or embedded OLE object. The action depends on the object. See Verb parameter.

Syntax

OleDoVerb (Verb: *string*; AttachmentIndex: *numeric*)

Parameters

Verb: *string*

Action to perform on an OLE object. If you open an item with two attached objects, a Paintbrush Picture and WordPerfect 6.0 document, the following example,

```
Application (A1; "WPOffice"; Default; "US")
OleDoVerb (Verb: "Edit"; AttachmentIndex: 1)
```

opens the second attached object (WP object) in WordPerfect for editing.

AttachmentIndex: *numeric*

Attachment index number (zero-based).

See Also

OleInsertObjectDlg

Route...
Open an item with one or more embedded or attached objects, select an object, choose Edit. Edit menu items depend on the object. For example, "Edit Paintbrush Picture Object" appears on the Edit menu if it is a Paintbrush object. "Edit WordPerfect Document (6.0) Object" appears if it is a WordPerfect 6.0 object. "Package Object" appears as a cascading menu if it is a Package object.

OleInsertObject

Purpose

Opens an application associated with an OLE object in a custom view.

To embed the object in an OLE box, choose File, Update or Update GroupWise depending on the application. The recipient can open, view, edit, and/or save an embedded OLE object.

Create custom views with View Designer, a program (vewin.exe) that ships with GroupWise.

Syntax

OleInsertObject (ObjectClass: *string*)

Parameters

ObjectClass: *string*
Object to embed, such as a Paintbrush object.

See Also

OleInsertObjectDlg

Route...
Open a custom view that has an OLE box, select the OLE box, choose Edit, Insert Object, select an object name, choose OK.

OleInsertObjectDlg

Purpose

Displays the Insert Object dialog box.

Syntax

OleInsertObjectDlg ()

Route...
Open a custom view that has an OLE box, select the OLE box, choose Edit, Insert Object.

OleLinksDlg

Purpose

Displays the Links dialog box to update, cancel, or change a Link.

Syntax

OleLinksDlg ()

Route...
Open a view that has one or more linked objects, choose Edit, Links.

OlePasteLink

Purpose

Pastes the contents of the clipboard into an OLE box, and creates a link to a server application.

The object must have been saved in the server application and copied to the clipboard. You cannot cut the object to the clipboard or make any changes after it is saved and use this command.

Syntax

OlePasteLink ()

Route...
Open a custom view that has an OLE box, select the OLE box, choose Edit, Paste Link.

OpenCalendar

Purpose

Opens or switches to the default Calendar view. Not recordable.

Syntax

OpenCalendar ()

OpenInBox

Purpose

Opens or switches to the In Box. Not recordable.

Syntax

OpenInBox ()

OpenOutBox

Purpose

Opens or switches to the Out Box. Not recordable.

Syntax

OpenOutBox ()

OpenTrashWindow

Purpose

Opens or switches to the Trash window. Not recordable.

Syntax

OpenTrashWindow ()

PosCharNext

Purpose
Moves the insertion point to the next character or space.

Syntax
PosCharNext ()

See Also
PosCharPrevious

Route...
Press Right Arrow.

PosCharPrevious

Purpose
Moves the insertion point to the previous character or space.

Syntax
PosCharPrevious ()

See Also
PosCharNext

Route...
Press Left Arrow.

PosLineBegin

Purpose
Moves the insertion point to the beginning of the current line.

Syntax
PosLineBegin ()

Route...
Press Home.

PosLineDown

Purpose
Moves the insertion point down one line.

Syntax
PosLineDown ()

Route...
Press Down Arrow.

PosLineUp

Purpose
Moves the insertion point up one line.

Syntax
PosLineUp ()

Route...
Press Up Arrow.

PosScreenDown

Purpose

Displays the next screen. The insertion point stays on the same line, and position if possible.

Syntax

PosScreenDown ()

Route...
Press Page Down.

PosScreenLeft

Purpose
Moves the insertion point to the left side of a message box.

Syntax
PosScreenLeft ()

Route...
Press Ctrl+Page Up.

PosScreenRight

Purpose
Moves the insertion point to the end of the current line.

Syntax
PosScreenRight ()

Route...
Press Ctrl+Page Down.

PosScreenUp

Purpose

Displays the previous screen. The insertion point stays on the same line, and position if possible.

Syntax

PosScreenUp ()

Route...
Press Page Up.

PosTextTop

Purpose
Moves the insertion point to the beginning of the first line.

Syntax
PosTextTop ()

Route...
Press Ctrl+Home.

PosToEndOfLine

Purpose
Moves the insertion point to the end of the current line.

Syntax
PosToEndOfLine ()

Route...
Press End.

PosToEndOfText

Purpose

Moves the insertion point to the end of the last line.

Syntax

PosToEndOfText ()

Route...
Press Ctrl+End.

PosWordLeft

Purpose

Moves the insertion point to the beginning of the current word, or to the beginning of the previous word depending on its current position.

Syntax

PosWordLeft ()

Route...
Press Ctrl+Left Arrow.

PosWordRight

Purpose
Moves the insertion point to the beginning of the next word.

Syntax
PosWordRight ()

Route...
Press Ctrl+Right Arrow.

PrefAdvanced

F

Purpose

Sets the default Advanced Send options.

Syntax

PrefAdvanced (Security: *enum*; ConcealSubject: *enum*; DeliverDelay: *numeric*; InsertOutBox: *enum*; ConvertAttachments: *enum*; RequireRoutedPassword: *enum*)

Parameters

Security: *enum* (optional)
Security label appears on the first line of a message box.

1 Normal! (no security label)
Proprietary!
Confidential!
Secret!
TopSecret!
ForYourEyesOnly!

ConcealSubject: *enum* (optional)
Conceal In Box and Out Box subject lines.

No!
Yes!

DeliverDelay: *numeric* (optional)
Number of days to delay delivery.

InsertOutBox: *enum* (optional)
Insert items in sender's Out Box.

No!
1 Yes!

ConvertAttachments: *enum* (optional)
Convert attached files through a gateway.

No!
Yes!

RequireRoutedPassword: *enum* (optional)
Require password to complete routed items.

No!
Yes!

See Also

PrefDlg
SendOptions
SendOptionsAdvanced

Route...
From the Main Window, choose File, Preferences, double-click Advanced, select options.

PrefAppointment

F

Purpose

Sets the default Appointment Send options.

Syntax

PrefAppointment (Priority: *enum*; Notify: *enum*; ReturnOnOpen: *enum*; ReturnOnAccept: *enum*; ReturnOnDelete: *enum*; StatusInfo: *enum*)

Parameters

Priority: *enum* (optional)

High!
Low!
Normal!

Notify: *enum* (optional)

Notify recipients when an Appointment item is delivered to their In Box. Notify must be active.

No!
Yes!

ReturnOnOpen: *enum* (optional)

Return notification options. Notify must be active.

Mail!
MailAndNotify!
None!
Notify!

ReturnOnAccept: *enum* (optional)

Return notification options for accepted or completed items. Notify must be active.

Mail!
MailAndNotify!
None!
Notify!

ReturnOnDelete: *enum* (optional)

Return notification options. Notify must be active.

Mail!
MailAndNotify!
None!
Notify!

StatusInfo: *enum* (optional)

Information to track when an Appointment item is sent.

AllInfo!
Delivered!
None!
Opened!

See Also

PrefDlg

Route...
From the Main Window, choose File, Preferences, double-click Appointment, select options.

PrefAppointmentTime

F

Purpose

Sets the default Appointment Time options.

Syntax

PrefAppointmentTime (ShowTimeIntervals: *enum*; StartTimeMinute: *numeric*; StartTimeHour: *numeric*; Interval: *numeric*; Alarm: *enum*; AlarmMinutes: *numeric*; AlarmHours: *numeric*; DisplayEventType: *enum*; DefaultLengthMinutes: *numeric*; DefaultLengthHours: *numeric*)

Parameters

ShowTimeIntervals: *enum* (optional)
Display time intervals (see Interval parameter).
- No!
- Yes!

StartTimeMinute: *numeric* (optional)
Minute to begin day on.

StartTimeHour: *numeric* (optional)
Hour to begin day on.

Interval: *numeric* (optional)
Length of display time interval (see ShowTimeIntervals parameter).

Alarm: *enum* (optional)
Automatically turn on alarm when an appointment is accepted.
- No!
- Yes!

AlarmMinutes: *numeric* (optional)
Minutes before appointment to sound alarm.

AlarmHours: *numeric* (optional)
Hours before appointment to sound alarm.

DisplayEventType: *enum* (optional)
Specifies how to display the appointment length.
- Duration!
- EndDateAndTime!

DefaultLengthMinutes: *numeric* (optional)
Default length of appointment in minutes.

DefaultLengthHours: *numeric* (optional)
Default length of appointment in hours.

See Also

PrefDlg

Route...
From the Main Window, choose File, Preferences, double-click Appointment Time, select options.

PrefBusySearch

F

Purpose
Sets the default Busy Search options.

Syntax
PrefBusySearch (StartTimeMinute: *numeric*; StartTimeHour: *numeric*; EndTimeMinute: *numeric*; EndTimeHour: *numeric*; AppointmentLengthMinutes: *numeric*; AppointmentLengthHours: *numeric*; BusySunday: *enum*; BusyMonday: *enum*; BusyTuesday: *enum*; BusyWednesday: *enum*; BusyThursday: *enum*; BusyFriday: *enum*; BusySaturday: *enum*; ExtendedInformation: *enum*; SearchRange: *numeric*)

Parameters
StartTimeMinute: *numeric* (optional)

StartTimeHour: *numeric* (optional)

EndTimeMinute: *numeric* (optional)

EndTimeHour: *numeric* (optional)

AppointmentLengthMinutes: *numeric* (optional)

AppointmentLengthHours: *numeric* (optional)

BusySunday: *enum* (optional)
Include Sunday in the search.
No!
Yes!

BusyMonday: *enum* (optional)
Include Monday in the search.
No!
Yes!

BusyTuesday: *enum* (optional)
Include Tuesday in the search.
No!
Yes!

BusyWednesday: *enum* (optional)
Include Wednesday in the search.
No!
Yes!

BusyThursday: *enum* (optional)
Include Thursday in the search.
No!
Yes!

BusyFriday: *enum* (optional)
Include Friday in the search.

No!
Yes!

BusySaturday: *enum* (optional)
Include Saturday in the search.
No!
Yes!

ExtendedInformation: *enum* (optional)
Display additional appointment information, depending on your access rights.
No!
Yes!

SearchRange: *numeric* (optional)
Number of days to search.

See Also
PrefDlg

Route...
From the Main Window, choose File, Preferences, double-click Busy Search, select options.

PrefCleanup

F

Purpose

Sets the default Cleanup options.

Syntax

PrefCleanUp (MailAndPhoneDelete: *enum*; MailAndPhoneDelayDays: *numeric*; AppointmentTaskAndNoteDelete: *enum*; AppointmentTaskAndNoteDelayDays: *numeric*; EmptyTrash: *enum*; EmptyTrashDelayDays: *numeric*)

Parameters

MailAndPhoneDelete: *enum* (optional)

AutoArchive!
AutoDelete!
Manual!

MailAndPhoneDelayDays: *numeric* (optional)

Number of days to AutoArchive! or AutoDelete! cleanup.

AppointmentTaskAndNoteDelete: *enum* (optional)

AutoArchive!
AutoDelete!
Manual!

AppointmentTaskAndNoteDelayDays: *numeric* (optional)

Number of days to AutoArchive! or AutoDelete! cleanup.

EmptyTrash: *enum* (optional)

Automatic!
Manual!

EmptyTrashDelayDays: *numeric* (optional)

Number of days to Automatic! cleanup.

See Also

PrefDlg

Route...
From the Main Window, choose File, Preferences, double-click CleanUp, select options.

PrefDateTimeFormat

F

Purpose

Sets the default Date and Time format.

Syntax

PrefDateTimeFormat (DateDefault: *string*; TimeDefault: *string*; MailboxDate: *string*; InfoDate: *string*; FileDate: *string*)

Parameters

DateDefault: *string* (optional)
Date format for all GroupWise views.

TimeDefault: *string* (optional)
Time format for all GroupWise views.

MailboxDate: *string* (optional)
Mailbox date and time formats.

InfoDate: *string* (optional)
Info date and time formats. To see date, open In Box, choose Actions, Info.

FileDate: *string* (optional)
Info date and time formats for attached items. To see date, open Out Box, choose Actions, Info.

See Also

PrefDlg

Route...
From the Main Window, choose File, Preferences, double-click Date Time Format, select options.

PrefDlg

Purpose

Displays the Preferences dialog box to set GroupWise defaults.

Syntax

PrefDlg ()

Route...
From the Main Window, choose File, Preferences.

PrefEnvironment

F

Purpose

Sets the default Environment options.

Syntax

PrefEnvironment (Language: *string*; AdvanceOnDelete: *enum*; NewScreenOnSend: *enum*; UpdateRateSeconds: *numeric*; UpdateRateMinutes: *numeric*; FirstDayofWeek: *enum*; UseSystemColors: *enum*; DisplaySculpting: *enum*; MacroSecurity: *enum*)

Parameters

Language: *string* (optional)
The display language. Only affects the interface. The desired language version must be installed. After setting, exit and re-enter GroupWise.

AdvanceOnDelete: *enum* (optional)
Display the next In Box or Out Box item after you accept, decline, or delete the current item.

- No!
- Yes!

NewScreenOnSend: *enum* (optional)
Open a new item view after item is sent.

- No!
- Yes!

UpdateRateSeconds: *numeric* (optional)
Specify in seconds how frequently GroupWise checks your mailbox for incoming items.

UpdateRateMinutes: *numeric* (optional)
Specify in minutes how frequently GroupWise checks your mailbox for incoming items.

FirstDayofWeek: *enum* (optional)
First day on mini-month calendars.

- Sunday!
- Monday!
- Tuesday!
- Wednesday!
- Thursday!
- Friday!
- Saturday!

UseSystemColors: *enum* (optional)
Use Windows System Colors.

- No!
- Yes!

DisplaySculpting: *enum* (optional)
Display controls in 3-D.

- No!
- Yes!

MacroSecurity: *enum* (optional)
Specify when to play a macro when you open a view with a macro button or a start-up macro.

AlwaysPrompt!
AutoPlay!
NeverPlay!

See Also

PrefDlg

Route...
From the Main Window, choose File, Preferences, double-click Environment, select options.

PrefFolderList

Purpose
Sets the default Folder List options.

Syntax
PrefFolderList (Inbox: *enum*; Outbox: *enum*; CalendarView: *enum*)

Parameters
Inbox: *enum* (optional)
Display options.
ExpandAll!
ExpandOne!
UseCurrent!

UseCurrent! displays the folder list as it appeared the last time it was opened.

Outbox: *enum* (optional)
Display options.
ExpandAll!
ExpandOne!
UseCurrent!

UseCurrent! displays the folder list as it appeared the last time it was opened.

CalendarView: *enum* (optional)
Display options.
ExpandAll!
ExpandOne!
UseCurrent!

UseCurrent! displays the folder list as it appeared the last time it was opened.

See Also
FolderExpand
FolderExpandAll

Route...
From the Main Window, choose File, Preferences, double-click Folder List, select options.

PrefLocationOfFiles

Purpose
Sets the default Location of Files options.

Syntax
PrefLocationOfFiles (ArchiveDir: *string*; CustomViews: *string*; ButtonbarMacroFilter: *string*; DefaultSaveDir: *string*)

Parameters
ArchiveDir: *string* (optional)

CustomViews: *string* (optional)

ButtonbarMacroFilter: *string* (optional)
Button Bar, Macro, and Filter files.

DefaultSaveDir: *string* (optional)
Item files and attachments.

See Also
PrefDlg

Route...
From the Main Window, choose File, Preferences, double-click Location of Files, select options.

PrefMailAndPhone

F

Purpose

Sets the default Mail and Phone Send options.

Syntax

PrefMailAndPhone (Priority: *enum*; Notify: *enum*; ReturnOnOpen: *enum*; ReturnOnCompleted: *enum*; ReturnOnDelete: *enum*; StatusInfo: *enum*; ReplyRequested: *enum*; ReplyRequestedDays: *numeric*; ExpireDays: *numeric*; AutoDelete: *enum*)

Parameters

Priority: *enum* (optional)

High!
Low!
Normal!

Notify: *enum* (optional)

Notify recipients when a Mail or Phone item is delivered to their In Box. Notify must be active.

No!
Yes!

ReturnOnOpen: *enum* (optional)

Return notification options. Notify must be active.

Mail!
MailAndNotify!
None!
Notify!

ReturnOnCompleted: *enum* (optional)

Return notification options for routed items. Notify must be active.

Mail!
MailAndNotify!
None!
Notify!

ReturnOnDelete: *enum* (optional)

Return notification options. Notify must be active.

Mail!
MailAndNotify!
None!
Notify!

StatusInfo: *enum* (optional)

Mail or Phone item information to track.

AllInfo!
Delivered!
None!
Opened!

ReplyRequested: *enum* (optional)

Request reply on the first line of the message box.

None!
WhenConvenient!
WithinDays!

ReplyRequestedDays: *numeric* (optional)
Number of days. The recipient sees: **Reply Requested:** By xx/xx/xx.

ExpireDays: *numeric* (optional)
Number of days a mail or phone message remains in a recipient's In Box.

AutoDelete: *enum* (optional)
Delete the item from the sender's Out Box when deleted from the recipient's In Box.
No!
Yes!

See Also
PrefDlg

Route...
From the Main Window, choose File, Preferences, double-click Mail/Phone, select options.

PrefNote

F

Purpose
Sets the default Note Send options.

Syntax
PrefNote (Priority: *enum*; Notify: *enum*; ReturnOnOpen: *enum*; ReturnOnDelete: *enum*; StatusInfo: *enum*)

Parameters
Priority: *enum* (optional)
- High!
- Low!
- Normal!

Notify: *enum* (optional)
Notify recipients when a Note is delivered to their In Box. Notify must be active.
- No!
- Yes!

ReturnOnOpen: *enum* (optional)
Return notification options. Notify must be active.
- Mail!
- MailAndNotify!
- None!
- Notify!

ReturnOnDelete: *enum* (optional)
Return notification options. Notify must be active.
- Mail!
- MailAndNotify!
- None!
- Notify!

StatusInfo: *enum* (optional)
Note information to track.
- AllInfo!
- Delivered!
- None!
- Opened!

See Also
PrefDlg

Route...
From the Main Window, choose File, Preferences, double-click Note, select options.

PrefTask

F

Purpose

Sets the default Task Send options.

Syntax

PrefTask (Priority: *enum*; Notify: *enum*; ReturnOnOpen: *enum*; ReturnOnAccept: *enum*; ReturnOnDelete: *enum*; ReturnOnCompleted: *enum*; StatusInfo: *enum*; DisplayEventType: *enum*)

Parameters

Priority: *enum* (optional)

High!
Low!
Normal!

Notify: *enum* (optional)

Notify recipients when a Task item is delivered to their In Box. Notify must be active.

No!
Yes!

ReturnOnOpen: *enum* (optional)

Return notification options. Notify must be active.

Mail!
MailAndNotify!
None!
Notify!

ReturnOnAccept: *enum* (optional)

Return notification options. Notify must be active.

Mail!
MailAndNotify!
None!
Notify!

ReturnOnDelete: *enum* (optional)

Return notification options. Notify must be active.

Mail!
MailAndNotify!
None!
Notify!

ReturnOnCompleted: *enum* (optional)

Return notification options. Notify must be active.

Mail!
MailAndNotify!
None!
Notify!

StatusInfo: *enum* (optional)

Task information to track.

AllInfo!

Delivered!
None!
Opened!

DisplayEventType: *enum* (optional)
Specify how to display a Task due date.
Duration!
EndDateAndTime!

See Also
PrefDlg

Route...
From the Main Window, choose File, Preferences, double-click Tasks, select options.

PrefViewDefaults

Purpose

Sets the default Views for personal and group views.

Syntax

PrefViewDefaults (GroupAppointmentView: *string*; PersonalAppointmentView: *string*; FolderView: *string*; MailView: *string*; GroupNoteView: *string*; PersonalNoteView: *string*; PhoneView: *string*; GroupTaskView: *string*; PersonalTaskView: *string*)

Parameters

GroupAppointmentView: *string*
Example: "Meeting w/attach"

PersonalAppointmentView: *string*
Example: "Personal Appointment w/attach"

FolderView: *string*
Calendar view. Example: "Day Planner"

MailView: *string*
Example: "Expanded Mail"

GroupNoteView: *string*
Example: "Notice"

PersonalNoteView: *string*
Example: "Personal Note"

PhoneView: *string*
Example: "Phone Message w/attach"

GroupTaskView: *string*
Example: "Task"

PersonalTaskView: *string*
Example: "Personal Task"

See Also

PrefDlg

Route...
From the Main Window, choose File, Preferences, and double-click Default Views, select options.

PrevPane

Purpose

Advances the input focus to the previous pane in the Main Window.

Syntax

PrevPane ()

Route...
In the Main Window, press Shift+Tab.

Print

Purpose
Prints an opened attachment.

Syntax
Print ()

Route...
Open an attachment, choose File, Print.

PrintCalendar

F

Purpose

Prints seven calendar types with options for each.

Syntax

PrintCalendar (Launch: *enum*; StartDay: *numeric*; StartMonth: *numeric*; StartYear: *numeric*; Gateway: *string*; PrintForm: *enum*; PrintLength: *numeric*; PageFormat: *enum*; SmallMonthHeaders: *enum*; OmitWeekends: *enum*; GreyBusyHours: *enum*; PrintEmptyDays: *enum*; OneDayPerPage: *enum*; PrintToFile: *enum*; FileName: *string*; UseTitle: *enum*; Title: *string*; UseFooter: *enum*; Footer: *string*; IncludeAppointments: *enum*; AppointmentEndTime: *enum*; AppointmentPlace: *enum*; AppointmentLines: *enum*; AppointmentDescription: *enum*; IncludeTasks: *enum*; TaskDeadline: *enum*; OmitCompletedTasks: *enum*; TaskDescription: *enum*; IncludeMemos: *enum*; MemoDescription: *enum*)

Parameters

Launch: *enum* (optional)
Print calendar using the associated application.

- No!
- Yes!

StartDay: *numeric* (optional)

StartMonth: *numeric* (optional)

StartYear: *numeric* (optional)

Gateway: *string* (optional)
Address string. See Fax/Print Gateway guide.

PrintForm: *enum* (optional)
Calendar type.

- DayPlanner!
- DayTrifold!
- Week!
- Month!
- Year!
- Multiuser!
- ASCIIList!

Multiuser! requires a proxy user. ASCIIList! prints an ASCII list of calendar events.

PrintLength: *numeric* (optional)
Number of days to print.

PageFormat: *enum* (optional)
Every page format is not available for every calendar type.

- Portrait3x5!
- Portrait5x8!
- Portrait8x11!
- Landscape8x11!

A6!
A5!
PortraitA4!
LandscapeA4!

SmallMonthHeaders: *enum* (optional)
Print small month calendar headers. Calendar type: Month.
No!
Yes!

OmitWeekends: *enum* (optional)
Calendar types: Day Organizer, Day Tri-Fold, Week Schedule, and Multi-User.
No!
Yes!

GreyBusyHours: *enum* (optional)
Shade scheduled hours. Calendar types: Week Schedule and Multi-User.
No!
Yes!

PrintEmptyDays: *enum* (optional)
Print days that have no scheduled appointments. Calendar type: ASCII List.
No!
Yes!

OneDayPerPage: *enum* (optional)
Print a new page for every day. Calendar type: ASCII List.
No!
Yes!

PrintToFile: *enum* (optional)
No!
Yes!

FileName: *string* (optional)
To print to file. Set PrintToFile to Yes!.

UseTitle: *enum* (optional)
Print a title. Calendar types: Week Schedule, Month, Year, and Multi-User.
No!
Yes!

Title: *string* (optional)
Title to print. Set UseTitle to Yes!.

UseFooter: *enum* (optional)
Print a footer. Calendar types: Week Schedule, Month, Year, and Multi-User.
No!
Yes!

Footer: *string* (optional)
Footer to print. Set UseFooter to Yes!.

IncludeAppointments: *enum*
Print appointments. Calendar types: Week Schedule, Month, Year, Multi-User, and ASCII

List.
    No!
    Yes!

AppointmentEndtime: *enum* (optional)
Print appointment end times. Calendar types: all.
    No!
    Yes!

AppointmentPlace: *enum* (optional)
Print appointment places. Calendar types: all.
    No!
    Yes!

AppointmentLines: *enum* (optional)
Print separator lines between appointments. Calendar types: Day Organizer and Day Tri-Fold.
    No!
    Yes!

AppointmentDescription: *enum* (optional)
Print appointment descriptions (explanations). Calendar type: ASCII List.
    No!
    Yes!

IncludeTasks: *enum* (optional)
Print tasks. Calendar types: Week Schedule, Month, Year, Multi-User, and ASCII List.
    No!
    Yes!

TaskDeadline: *enum* (optional)
Print task deadlines. Calendar types: all.
    No!
    Yes!

OmitCompletedTasks: *enum* (optional)
Print completed tasks. Calendar types: all.
    No!
    Yes!

TaskDescription: *enum* (optional)
Print task descriptions (explanations). Calendar type: ASCII List.
    No!
    Yes!

IncludeMemos: *enum* (optional)
Print memos (notes). Calendar types: Week Schedule, Month, Year, Multi-User, and ASCII List.
    No!
    Yes!

MemoDescription: *enum* (optional)
Print memo descriptions (explanations). Calendar type: ASCII List.
    No!
    Yes!

See Also

PrintCalendarDlg

Route...
Open a Calendar view, choose File, Print Calendar, select options.

PrintCalendarDlg

Purpose
Displays the Print Calendar dialog box.

Syntax
PrintCalendarDlg ()

See Also
PrintCalendar

Route...
Open a Calendar view, choose File, Print Calendar.

PrintDlg

Purpose
Displays the Print dialog box.

Syntax
PrintDlg ()

Route...
Select an item or open a view, choose File, Print.

PrintSetup

Purpose
Displays the Print Setup dialog box.

Syntax
PrintSetup ()

Route...
Choose File, Print Setup.

PromptForPassword

Purpose

Prompts the user for a password and verifies that it is correct. User may enter a password up to three times. Not recordable.

Syntax

PromptForPassword ()

PromptSetMode

Purpose

Specifies whether to suppress macro prompt messages. Not recordable.

Prompt messages that cannot be anticipated or responded to can interfer with DDE macro playback. When *PromptingMode* is Suppressed!, macros perform default actions and do not display a prompt message.

This command, which does not affect error messages, is automatically set to Normal! when the macro or DDE conversation ends.

Syntax

PromptSetMode (PromptingMode: *enum*)

Parameters

PromptingMode: *enum*

Normal!
Suppressed!

Proxy

Purpose
Specifies a person to proxy for (the person must grant proxy rights).

Syntax
Proxy (UserID: *string*; Window: *enum*)

Parameters
UserID: *string*
Person to proxy for.

Window: *enum*
CurrentWindow!
MainWindow!
None!

See Also
ProxyDlg

Route...
Choose File, Proxy, type a user's ID.

ProxyDlg

Purpose
Displays the Proxy dialog box.

Syntax
ProxyDlg ()

See Also
Proxy

Route...
Choose File, Proxy.

Refresh

Purpose

Updates In Box, Out Box, Trash, or Calendar views to display items received after the view was opened.

Syntax

Refresh ()

See Also

InfoUpdate

Route...
From the Main Window, In Box, Out Box, Trash, or a Calendar view, choose View, Refresh.

RemoteConnect

F

Purpose

Connects GroupWise Remote to the master system and updates the remote mailbox.

Syntax

RemoteConnect (RequestItems: *enum*; RequestFolders: *enum*; RequestRules: *enum*; RequestPersonalGroups: *enum*; RequestAddressBook: *enum*; RequestPublicGroups: *enum*)

Parameters

RequestItems: *enum* (optional)
Default: request items if no item request is pending.
- No!
- Yes!

RequestFolders: *enum* (optional)
Default: No!.
- No!
- Yes!

RequestRules: *enum* (optional)
Default: No!.
- No!
- Yes!

RequestPersonalGroups: *enum* (optional)
Default: No!.
- No!
- Yes!

RequestAddressBook: *enum* (optional)
Default: No!.
- No!
- Yes!

RequestPublicGroups: *enum* (optional)
Default: No!.
- No!
- Yes!

Route...
Choose Remote, Send/Retrieve, Connect.

RemoteCreateNetworkConnection

Purpose

Creates a network connection.

Syntax

RemoteCreateNetworkConnection (ConnectionName: *string*; PostOfficePath: *string*; DisconnectMethod: *enum*)

Parameters

ConnectionName: *string*
User-defined.

PostOfficePath: *string*
Include full path.

DisconnectMethod: *enum*
AfterAllUpdates!
AfterRequestsSent!
Manual!

AfterAllUpdates! disconnects after all items are sent, and all requests are processed and retrieved to the remote mailbox. AfterRequestsSent! disconnects after all items are sent to the master mailbox and all waiting responses are retrieved. The connection does not wait for the master system to process requests. Manual! disconnects when you manually terminate the connection.

Route...
Run GroupWise Remote 4.1, choose Remote, Send/Retrieve, Connections, Create, select Network, OK, select options.

RemoteCreateModemConnection

F

Purpose

Creates a modem connection.

Syntax

RemoteCreatemodemConnection (ConnectionName: *string*; PhoneNumber: *string*; GatwayLoginID: *string*; GatewayPassword: *string*; ModemScriptFilePath: *string*; DisconnectMethod: *enum*; RedialAttempts: *numeric*; RedialInterval: *numeric*)

Parameters

ConnectionName: *string*
User-defined name.

PhoneNumber: *string*

GatwayLoginID: *string*

GatewayPassword: *string*

ModemScriptFilePath: *string* (optional)

DisconnectMethod: *enum* (optional)
AfterAllUpdates!
AfterRequestsSent!
Manual!

Default: AfterAllUpdates!.

RedialAttempts: *numeric* (optional
Default: 5.

RedialInterval: *numeric* (optional)
Number of minutes. Default: 1.

See Also

RemoteSetPortInfo

Route...
Run GroupWise Remote 4.1, choose Remote, Send/Retrieve, Connections, Create, select Modem, OK, select options.

RemoteDeleteConnection

Purpose

Deletes a remote connection from the Connection dialog box. Not recordable.

Syntax

RemoteDeleteConnection (ConnectionName: *string*)

Parameters

ConnectionName: *string*

See RemoteCreateNetworkConnection.

RemoteDisconnect

Purpose

Disconnects GroupWise Remote from the master system, and closes the Connection Status window.

Token fails if there is no connection.

Syntax

RemoteDisconnect ()

Route...
Run GroupWise Remote 4.1, choose Remote, Send/Retrieve, Connect, Disconnect.

RemoteGetDefaultConnectionName

Purpose
Returns the name of the current connection. Not recordable.

Syntax
string RemoteGetDefaultConnectionName ()

Return Value
string Connection name.

RemoteGetPhoneNumber

Purpose
Returns the phone number of the default modem connection. Not recordable.

Syntax
string RemoteGetPhoneNumber ()

Return Value
string Phone number.

RemoteModemCommand

Purpose
Sends commands to a modem while a Remote connection is being initiated. Not recordable.

Syntax
string RemoteModemCommand (Command: *enum*; Parameter: *string*)

Return Value
string Information about the modem connection.

Parameters
Command: *enum*
- Break!
- CharAvailable!
- Connected!
- TerminalMode!
- Read!
- AllowLogging!
- Settings!
- Write!

Break!, CharAvailable!, Connected! TerminalMode, and Read! ignore optional string parameter (see Parameter). AllowLogging!, Settings!, and Write! use the string parameter.

Break! sends a one second break command to the modem. CharAvailable! returns True if characters are available, False if not. Connected! returns True if connected, False if not. TerminalMode! displays the Modem Terminal dialog box. Read! returns the characters waiting at the modem (more than one Read! may be needed to return a desired string). AllowLogging! turns logging to the Log Window "On" or "Off" (see RemoteViewConnectionLog). Settings! modifies a modem's baud rate, parity, data bits, and stop bits using a comma delimited string. An empty field uses the current setting. Write! sends the string parameter to a modem (see Parameter).

Parameter: *string* (optional)
String data passed to a modem.

RemotePendingRequestsDlg

Purpose

Displays the Pending Requests to Master Mailbox dialog box.

Syntax

RemotePendingRequestsDlg ()

Route...
Run GroupWise Remote 4.1, choose Remote, Pending Requests.

RemoteSelectedRetrieveDlg

Purpose

Displays the Retrieve Selected Items dialog box. See GroupWise Help, Retrieve Selected Items (Remote) for an example.

Syntax

RemoteSelectedRetrieveDlg ()

Route...
Run GroupWise Remote 4.1, select one or more In Box or Out Box items, choose Remote, Retrieve Selected Items.

RemoteSendRetrieveDlg

Purpose

Displays the Send/Retrieve dialog box.

Syntax

RemoteSendRetrieveDlg ()

Route...
Run GroupWise Remote 4.1, choose Remote, Send/Retrieve.

RemoteSetAddrBookDnloadFilter

F

Purpose

Requests user IDs, resources, and public group names from the master system.

Syntax

RemoteSetAddrBookDnloadFilter (DomainName: *string*; PostOfficeName: *string*)

Parameters

DomainName: *string*
All post offices in this domain. If blank, all domains.

PostOfficeName: *string*
One post office in a specified domain (see DomainName parameter). Each post office represents a collection of mailboxes. If blank, all post offices.

Route...
Run GroupWise Remote 4.1, choose Remote, Send/Retrieve, select Address Book (check box), Address Book (button), type Domain name or Domain and Post Office name.

RemoteSetDaylightTimeDate

F

Purpose

Sets mailbox beginnng and end of standard and daylight saving time according to absolute dates.

Syntax

RemoteSetDaylightTimeDate (UseDaylightSavingTime: *enum*; ClockChange: *numeric*; DSTStartDay: *numeric*; DSTStartMonth: *numeric*; StandardTimeStartDay: *numeric*; StandardTimeStartMonth: *numeric*)

Parameters

UseDaylightSavingTime: *enum*

No!
Yes!

ClockChange: *numeric*
Minutes to move clock forward or back.

DSTStartDay: *numeric*
Day of the month.

DSTStartMonth: *numeric*
Zero-based. 0 = January, 1 = February, and so forth.

StandardTimeStartDay: *numeric*
Day of the month.

StandardTimeStartMonth: *numeric*
Zero-based. 0 = January, 1 = February, and so forth.

See Also

RemoteSetDaylightTimeFormula

Route...
Run GroupWise Remote 4.1, choose File, Preferences, Remote, Time Zone, Edit Daylight Saving Time, select Define by Date, select options.

RemoteSetDaylightTimeFormula

F

Purpose

Sets mailbox beginnng and end of standard and daylight saving time according to a formula.

Example: First Sunday in April.

Syntax

RemoteSetDaylightTimeFormula (UseDaylightSavingTime: *enum*; ClockChange: *numeric*; DSTRelativeStartDay: *enum*; DSTStartDayOfWeek: *enum*; DSTStartMonth: *numeric*; StandardTimeStartDay: *numeric*; StandardTimeStartDayOfWeek: *enum*; StandardTimeStartMonth: *numeric*)

Parameters

UseDaylightSavingTime: *enum*

- No!
- Yes!

ClockChange: *numeric*

Minutes to move clock forward or back.

DSTRelativeStartDay: *enum*

- First!
- Fourth!
- Last!
- Second!
- Third!

DSTStartDayOfWeek: *enum*

- Friday!
- Monday!
- Saturday!
- Sunday!
- Thursday!
- Tuesday!
- Wednesday!

DSTStartMonth: *numeric*

Zero-based. 0 = January, 1 = February, and so forth.

StandardTimeStartDay: *enum*

- First!
- Fourth!
- Last!
- Second!
- Third!

StandardTimeStartDayOfWeek: *enum*

- Friday!
- Monday!
- Saturday!

Sunday!
Thursday!
Tuesday!
Wednesday!

StandardTimeStartMonth: *numeric*
Zero-based. 0 = January, 1 = February, and so forth.

See Also

RemoteSetDaylightTimeDate

Route...
Run GroupWise Remote 4.1, choose File, Preferences, Remote, Time Zone, Edit Daylight Saving Time, select Define by Day, select options.

RemoteSetDefaultConnection

Purpose
Sets the current modem or network connection.

Syntax
RemoteSetDefaultConnection (ConnectionName: *string*)

Parameters
ConnectionName: *string*

Route...
Run GroupWise Remote 4.1, choose Remote, Send/Retrieve, Connections, select a connection, Select.

RemoteSetItemsFolders

Purpose

Download one or all folders from the master system. Repeat this token to add more than one folder but fewer than all.

Syntax

RemoteSetItemsFolders (RetrieveContents: *enum*; FolderName: *string*)

Parameters

RetrieveContents: *enum*

Add!
All!
Reset!

Add! adds one folder to the folder list. Reset! clears a previous folder list and adds one folder (see FolderName parameter) to a new folder list.

FolderName: *string*

Ignored if RetrieveContents is set to All!.

Route...
Run GroupWise Remote 4.1, choose Remote, Send/Retrieve, Items (select options), Items to Update (select options), OK.

RemoteSetPortInfo

F

Purpose

Sets port information for using a modem.

Syntax

RemoteSetPortInfo (PortNum: *numeric*; BaudRate: *numeric*; Parity: *enum*; DataBits: *enum*; StopBits: *enum*; FlowControl: *enum*)

Parameters

PortNum: *numeric*
Serial communications port. Example: 1 (for Com1).

BaudRate: *numeric*
Rate for sending and receiving information from the gateway. Example: 2400.

Parity: *enum*
- Even!
- None!
- Odd!

DataBits: *enum*
- DataBits7!
- DataBits8!

StopBits: *enum*
- StopBit1!
- StopBits2!

FlowControl: *enum*
- CtsRts!
- NoFlowControl!
- XonXoff!

See Also

RemoteCreateModemConnection

Route...
Run GroupWise Remote 4.1, choose File, Preferences, Remote icon, Modem, Port.

RemoteSetPreferences

Purpose

Sets remote preferences, such as name, userID and so forth.

Syntax

RemoteSetPreferences (Fullname: *string*; UserID: *string*; MMPassword: *string*; Domain: *string*; PostOffice: *string*; Modem: *string*)

Parameters

FullName: *string*

UserID: *string*

MMPassword: *string*
Master mailbox password.

Domain: *string*
Post office location.

PostOffice: *string*
Mailbox location.

Modem: *string*

See Also

RemoteSetDefaultConnection

Route...
Run GroupWise Remote 4.1, choose File, Preferences, double-click Remote, select options.

RemoteSetPublicGroupDnloadFilter

Purpose
Sets a filter for retrieving public groups.

Syntax
RemoteSetPublicGroupDnloadFilter (GroupName: *string*)

Parameters
GroupName: *string*

Route...
Run GroupWise Remote 4.1, choose Remote, Send/Retrieve, select Public Group Members, Public Group, select a group.

RemoteSetReconcile

F

Purpose

Specifies how to reconcile master and remote mailbox when connection is made.

Syntax

RemoteSetReconcile (UploadPersonalItems: *enum*; UploadPersonalGroups: *enum*; UploadRules: *enum*; DelMissingItems: *enum*; UpdateModifiedItems: *enum*; ConfirmDeletions: *enum*)

Parameters

UploadPersonalItems: *enum*

UploadAlways!
UploadNever!
UploadPrompt!

UploadPrompt! prompts whether you want a newly created personal item uploaded to the master system the next time you connect.

UploadPersonalGroups: *enum*

UploadAlways!
UploadNever!
UploadPrompt!

UploadPrompt! prompts whether you want a newly created personal group uploaded to the master system the next time you connect.

UploadRules: *enum*

UploadAlways!
UploadNever!
UploadPrompt!

UploadPrompt! prompts whether you want a newly created rule uploaded to the master system the next time you connect.

DelMissingItems: *enum*

Delete remote items that do not exist as master items each time you connect. The update occurs as items are retrieved, and only for items that fall in a specified date range (see RemoteSetRequestItemsFilter).

No!
Yes!

UpdateModifiedItems: *enum*

Update remote mailbox with new items or changes to the master mailbox each time you connect. The update occurs as items are retrieved, and only for items that fall in a specified date range (see RemoteSetRequestItemsFilter).

No!
Yes!

ConfirmDeletions: *enum*

Prompt when you delete a remote item to have it deleted in the master mailbox the next time you connect.

No!
Yes!

Route...
Run GroupWise Remote 4.1, choose File, Preferences, double-click Remote, Reconcile, select options.

RemoteSetRequestItemsFilter

F

Purpose

Sets filters for retrieving items from the master system. Parameters not specified are not retrieved to the user's remote database.

Syntax

RemoteSetRequestItemsFilter (FromInBox: *enum*; FromOutBox: *enum*; FromPersonalBox: *enum*; RetrieveMailAndPhone: *enum*; RetrieveAppointments: *enum*; RetrieveTasks: *enum*; RetrieveNotes: *enum*; MessageSizeLimit: *numeric*; AttachmentSizeLimit: *numeric*; DistributionListSizeLimit: *numeric*; DaysPrior: *numeric*; DaysAfter: *numeric*)

Parameters

FromInBox: *enum*

No!
Yes!

FromOutBox: *enum*

No!
Yes!

FromPersonalBox: *enum*

No!
Yes!

RetrieveMailAndPhone: *enum*

All!
None!
Opened!
Unopened!

RetrieveAppointments: *enum*

Accepted!
All!
None!
Unaccepted!

RetrieveTasks: *enum*

All!
Completed!
None!
Uncompleted!

RetrieveNotes: *enum*

All!
None!
Opened!
Unopened!

MessageSizeLimit: *numeric*

Retrieve message text if less than a specified size in bytes. Example: 30720 (30x1024 or 30 kilobytes).

AttachmentSizeLimit: *numeric*
Retrieve attachments if less than a specified size in bytes. Example: 30720 (30x1024 or 30 kilobytes).

DistributionListSizeLimit: *numeric*
Retrieve To, CC, and BC text if less than a specified size in bytes. Example: 5120 (5x1024 or 5 kilobytes).

DaysPrior: *numeric*
Retrieve items no older than a specified number of days prior to the current date. Example: 5.

DaysAfter: *numeric*
Retrieve items that were received no later than a specified number of days after the current date. Example: 365.

Route...
Run GroupWise Remote 4.1, choose Remote, Send/Retrieve, select Items (check box), Items (button), select options.

RemoteSetTimeZone

F

Purpose

Sets the time zone. For a list of cities that correspond to Time Zone names and GMT Offset values, follow route to Set Time Zone dialog box.

Syntax

RemoteSetTimeZone (TimeZone: *enum*; GMTOffset: *numeric*)

Parameters

TimeZone: *enum*

- AtlanticStandard!
- AustraliaCentral!
- AustraliaEastern!
- AustraliaWestern!
- CentralEurope!
- CentralStandard!
- EasternStandard!
- Europe2!
- Greenwich!
- HawaiianStandard!
- MiddleEast!
- MountainStandard!
- NewZealand!
- Orient8!
- Orient9!
- PacificStandard!
- SouthAmerica3!
- SouthAmerica4!
- SouthAmerica5!
- UserDefined!

GMTOffset: *numeric*

Offset from Greenwich Mean Time.

Route...
Run GroupWise Remote 4.1, choose File, Preferences, Remote, Time Zone, Set Time Zone, select option.

RemoteViewConnectionLog

Purpose

Displays the Connection Log message box.

Syntax

RemoteViewConnectionLog ()

Route...
Run GroupWise Remote 4.1, choose Remote, Connection Log.

Retrieve

Purpose
Retrieves a WordPerfect or text file into a message box.

Syntax
Retrieve (FileName: *string*)

Parameters
FileName: *string*

See Also
RetrieveFileDlg

Route...
Open an item view, choose File, Retrieve, type a filename.

RetrieveFileDlg

Purpose
Displays the Retrieve dialog box.

Syntax
RetrieveFileDlg ()

See Also
Retrieve

Route...
Open an item view, choose File, Retrieve.

RuleAddActionAccept

Purpose
Accepts a task or appointment when rule conditions are met (see RuleCreate).

Syntax
RuleAddActionAccept (UserID: *string*; RuleName: *string*; Comment: *string*)

Parameters
UserID: *string* (optional)

RuleName: *string*

Comment: *string* (optional)
Comment to include with accepted task or appointment.

See Also
RuleCreate
RuleDelete

Route...
Choose Tools, Rules, select a rule, Edit, Add, select Accept, type comment, choose OK.

RuleAddActionArchive

F

Purpose
Archives an item when rule conditions are met (see RuleCreate).

Syntax
RuleAddActionArchive (UserID: *string*; RuleName: *string*)

Parameters
UserID: *string* (optional)

RuleName: *string*

See Also
RuleCreate
RuleDelete

Route...
Choose Tools, Rules, select a rule, Edit, Add, select Archive, choose OK.

RuleAddActionDecline

Purpose
Declines a task or appointment when rule conditions are met (see RuleCreate).

Syntax
RuleAddActionDecline (UserID: *string*; RuleName: *string*; Comment: *string*)

Parameters
UserID: *string* (optional)

RuleName: *string*

Comment: *string* (optional)
Comment to include with declined task or appointment.

See Also
RuleCreate
RuleDelete

Route...
Choose Tools, Rules, select a rule, Edit, Add, select Delete/Decline, type comment, OK.

RuleAddActionDelegate

F

Purpose

Delegates a task, appointment, or note item when rule conditions are met (see RuleCreate).

Syntax

RuleAddActionDelegate (UserID: *string*; RuleName: *string*; To: *string*; SenderComments: *string*; RecipientComments: *string*)

Parameters

UserID: *string* (optional)

RuleName: *string*

To: *string*
User ID of the person delegated a task, appointment, or note.

SenderComments: *string* (optional)
Comments to the person who sent the task, appointment, or note.

RecipientComments: *string* (optional)
Comments to the person delegated the task, appointment, or note.

See Also

RuleCreate
RuleDelete

Route...
Choose Tools, Rules, select a rule, Edit, Add, select Delegate, type comment, OK.

RuleAddActionEmptyItem

F

Purpose

Deletes an item and then empties it from Trash when rule conditions are met (see RuleCreate).

F You cannot recover emptied items.

Syntax

RuleAddActionEmptyItem (UserID: *string*; RuleName: *string*)

Parameters

UserID: *string* (optional)

RuleName: *string*

See Also

RuleCreate
RuleDelete

Route...
Choose Tools, Rules, select a rule, Edit, Add, select Empty Item.

RuleAddActionForward

F

Purpose

Forwards an item when rule conditions are met (see RuleCreate). Forwarded items are sent as an attachment to a mail message.

Syntax

RuleAddActionForward (UserID: *string*; RuleName: *string*; To: *string*; CC: *string*; BC: *string*; Subject: *string*; Files: *string*; Message: *string*; From: *string*)

Parameters

UserID: *string* (optional)

RuleName: *string*

To: *string* (optional)
User ID of the person to receive the forwarded item.

CC: *string* (optional)
User ID of the person to receive a copy of the forwarded item.

BC: *string* (optional)
User ID of the person to receive a blind copy of the forwarded item.

Subject: *string* (optional)

Files: *string* (optional)
Separate multiple attachments with a comma.

Message: *string* (optional)

From: *string* (optional)
Name of person forwarding the item.

See Also

RuleCreate
RuleDelete

Route...
Choose Tools, Rules, select a rule, Edit, Add, select Forward, specify options, choose OK.

RuleAddActionLinkToFolder

F

Purpose
Links an item to a folder when rule conditions are met (see RuleCreate).

Syntax
RuleAddActionLinkToFolder (UserID: *string*; RuleName: *string*; FolderName: *string*)

Parameters
UserID: *string* (optional)

RuleName: *string*

FolderName: *string*
Include full folder path.

See Also
RuleCreate
RuleDelete

Route...
Choose Tools, Rules, select a rule, Edit, Add, select Link To Folder, select a folder, Link.

RuleAddActionMarkPrivate

F

Purpose
Marks an item private when rule conditions are met (see RuleCreate).

Syntax
RuleAddActionMarkPrivate (UserID: *string*; RuleName: *string*)

Parameters
UserID: *string* (optional)

RuleName: *string*

See Also
RuleCreate
RuleDelete

Route...
Choose Tools, Rules, select a rule, Edit, Add, select Mark As Private.

RuleAddActionMoveToFolder

F

Purpose
Moves an item to a folder when rule conditions are met (see RuleCreate).

Syntax
RuleAddActionMoveToFolder (UserID: *string*; RuleName: *string*; FolderName: *string*)

Parameters
UserID: *string* (optional)

RuleName: *string*

FolderName: *string*
Include full folder path.

See Also
RuleCreate
RuleDelete

Route...
Choose Tools, Rules, select a rule, Edit, Add, select Move to Folder, select a folder, Move.

RuleAddActionReply

F

Purpose

Replies to the sender of a mail, appointment, task, or note item when rule conditions are met (see RuleCreate).

Syntax

RuleAddActionReply (UserID: *string*; RuleName: *string*; ReplyType: *enum*; CC: *string*; BC: *string*; Subject: *string*; Files: *string*; Message: *string*; From: *string*)

Parameters

UserID: *string* (optional)

RuleName: *string*

ReplyType: *enum*
Reply to sender only or to sender and all recipients of the same mail, appointment, task, or note item.

ToAll!
ToSender!

CC: *string* (optional)
User ID of the person to receive a copy.

BC: *string* (optional)
User ID of the person to receive a blind copy.

Subject: *string* (optional)
Reply subject.

Files: *string* (optional)
Reply attachments. Separate multiple attachments with a comma.

Message: *string* (optional)
Reply message.

From: *string* (optional)
Name of the person replying.

See Also

RuleCreate
RuleDelete

Route...
Choose Tools, Rules, select a rule, Edit, Add, select Reply, select option, choose OK, fill in Message, CC, BC, and Files text boxes.

RuleAddActionSendMail

F

Purpose

Sends a mail item when rule conditions are met (see RuleCreate).

Syntax

RuleAddActionSendMail (UserID: *string*; RuleName: *string*; To: *string*; CC: *string*; BC: *string*; Subject: *string*; Files: *string*; Message: *string*; From: *string*; ViewName: *string*)

Parameters

UserID: *string* (optional)

RuleName: *string*

To: *string*
Recipient user ID.

CC: *string* (optional)
User ID of the person to receive a copy.

BC: *string* (optional)
User ID of the person to receive a blind copy.

Subject: *string* (optional)
Mail item subject.

Files: *string* (optional)
Mail item attachments. Separate multiple attachments with a comma.

Message: *string* (optional)
Mail item message.

ViewName: *string* (optional)
Name of a mail view, such as "Mail", or the name of a custom mail view.

See Also

RuleCreate

Route...
Choose Tools, Rules, select a rule, Edit, Add, select Send Mail, fill in To, Subject, Message, CC, BC, and Files text boxes.

RuleCreate

F

Purpose

Creates a rule that sets up conditions for performing an action on one or more items.

This command must be followed with RuleAddAction commands.

Syntax

RuleCreate (UserID: *string*; RuleName: *string*; Event: *enum*; Source: *enum*; ItemTypeAppointment: *enum*; ItemTypeMail: *enum*; ItemTypeNote: *enum*; ItemTypePhone: *enum*; ItemTypeTask: *enum*; FolderName: *string*; To: *string*; CC: *string*; Subject: *string*; Message: *string*; From: *string*; PlaceName: *string*; TaskPriority: *string*; CallerName: *string*; CompanyText: *string*; PhoneNumber: *string*; DateRangeType: *enum*; StartDay: *numeric*; StartMonth: *numeric*; StartYear: *numeric*; StartHour: *numeric*; StartMinute: *numeric*; EndDay: *numeric*; EndMonth: *numeric*; EndYear: *numeric*; EndHour: *numeric*; EndMinute: *numeric*; IsAccepted: *enum*; IsCompleted: *enum*; HasConflictingAppt: *enum*; IsOpened: *enum*; IsPrivate: *enum*; HasReplyRequested: *enum*; IsRouted: *enum*; HasAttachedFile: *enum*; HasEmbeddedMovie: *enum*; HasSoundAnnotation: *enum*; IsLowPriority: *enum*; IsNormalPriority: *enum*; IsHighPriority: *enum*)

Parameters

UserID: *string* (optional)

RuleName: *string*

Event: *enum*
Rule criteria: Activate rule on one of the following events.
- CloseFolder!
- Exit!
- FiledItem!
- NewItem!
- OpenFolder!
- Startup!
- UserActivated!

Source: *enum* (optional)
NewItem! event (see Event parameter).
- Inbox!
- Outbox!
- Personal!

ItemTypeAppointment: *enum* (optional)
Rule criteria: Appointment item.
- No!
- Yes!

ItemTypeMail: *enum* (optional)
Rule criteria: Mail item.
- No!
- Yes!

ItemTypeNote: *enum* (optional)

Rule criteria: Note item.
    No!
    Yes!

ItemTypePhone: *enum* (optional)
Rule criteria: Phone item.
    No!
    Yes!

ItemTypeTask: *enum* (optional)
Rule criteria: Task item.
    No!
    Yes!

FolderName: *string* (optional)
The folder name when an event is CloseFolder!, FiledItem!, or OpenFolder! (see Event parameter).

To: *string* (optional)
Rule criteria: Recipient user ID.

CC: *string* (optional)
Rule criteria: Carbon copy recipient user ID.

Subject: *string* (optional)
Rule criteria: Item subject.

Message: *string* (optional)
Rule criteria: Item message.

From: *string* (optional)
Rule criteria: Sender's first and last name.

PlaceName: *string* (optional)
Rule criteria: Place when Appointment is the only item (see ItemTypeAppointment parameter).

TaskPriority: *string* (optional)
Rule criteria: Priority when Task is the only item (see ItemTypeTask parameter).

CallerName: *string* (optional)
Rule criteria: Caller's name when Phone is the only item (see ItemTypePhone parameter).

CompanyText: *string* (optional)
Rule criteria: Company's name when Phone is the only item (see ItemTypePhone parameter).

PhoneNumber: *string* (optional)
Rule criteria: Phone number when Phone is the only item (see ItemTypePhone parameter).

DateRangeType: *enum* (optional)
Rule criteria: Date and time ranges when Appointment, Task, and/or Note items are the only items (see Start and End parameters).
    After!
    Anytime!

Before!
Between!
On!

StartDay: *numeric* (optional)
Rule criteria: Beginning day when After!, Before!, Between!, or On! are set (see DateRangeType parameter).

StartMonth: *numeric* (optional)
Rule criteria: Beginning month when After!, Before!, Between!, or On! are set (see DateRangeType parameter).

StartYear: *numeric* (optional)
Rule criteria: Beginning year when After!, Before!, Between!, or On! are set (see DateRangeType parameter).

StartHour: *numeric* (optional)
Rule criteria: Beginning hour when After!, Before!, or Between! are set (see DateRangeType parameter).

StartMinute: *numeric* (optional)
Rule criteria: Beginning minute when After!, Before!, or Between! are set (see DateRangeType parameter).

EndDay: *numeric* (optional)
Rule criteria: Ending day when Between! is set (see DateRangeType parameter).

EndMonth: *numeric* (optional)
Rule criteria: Ending month when Between! is set (see DateRangeType parameter).

EndYear: *numeric* (optional)
Rule criteria: Ending year when Between! is set (see DateRangeType parameter).

EndHour: *numeric* (optional)
Rule criteria: Ending hour when Between! is set (see DateRangeType parameter).

EndMinute: *numeric* (optional)
Rule criteria: Ending minute when Between! is set (see DateRangeType parameter).

IsAccepted: *enum* (optional)
Rule criteria: Appointment or task accepted.
DontCare!
No!
Yes!

IsCompleted: *enum* (optional)
Rule criteria: Task completed.
DontCare!
No!
Yes!

HasConflictingAppt: *enum* (optional)
Rule criteria: Conflicting appointments.
DontCare!
No!

Yes!

IsOpened: *enum* (optional)
Rule criteria: Item opened.
DontCare!
No!
Yes!

IsPrivate: *enum* (optional)
Rule criteria: Item marked private.
DontCare!
No!
Yes!

HasReplyRequested: *enum* (optional)
Rule criteria: Reply requested.
DontCare!
No!
Yes!

IsRouted: *enum* (optional)
Rule criteria: Routed item.
DontCare!
No!
Yes!

HasAttachedFile: *enum* (optional)
Rule criteria: Attached text or object file.
DontCare!
Yes!

HasEmbeddedMovie: *enum* (optional)
Rule criteria: Movie file attached.
DontCare!
Yes!

HasSoundAnnotation: *enum* (optional)
Rule criteria: Sound file attached.
DontCare!
Yes!

IsLowPriority: *enum* (optional)
Rule criteria: Sender priority low.
DontCare!
Yes!

IsNormalPriority: *enum* (optional)
Rule criteria: Sender priority normal.
DontCare!
Yes!

IsHighPriority: *enum* (optional)
Rule criteria: Sender priority high.
DontCare!
Yes!

See Also

RuleDelete

Route...
Choose Tools, Rules, Create, select options.

RuleDelete

Purpose
Deletes a rule.

Syntax
RuleDelete (UserID: *string*; RuleName: *string*)

Parameters
UserID: *string* (optional)

RuleName: *string*

See Also
RuleCreate

Route...
Choose Tools, Rules, select a rule, choose Delete.

RuleExecute

Purpose
Runs a rule on selected In Box, Out Box, Trash, or Calendar items.

Syntax
RuleExecute (RuleName: *string*)

Parameters
RuleName: *string*

Route...
Select an In Box, Out Box, Trash, or Calendar item, choose Tools, Rules, select a rule, Run.

RuleListDlg

Purpose

Displays the Rules dialog box to list, create, edit, copy, delete, and run rules.

Syntax

RuleListDlg ()

Route...
Choose Tools, Rules.

ScrollLeft

Purpose

Display the previous day in the Week calendar view.

Syntax

ScrollLeft ()

See Also

ScrollNext
ScrollPrior
ScrollRight

Route...
Select the left-arrow button in the upper left corner of the Week Calendar view.

ScrollNext

Purpose

Displays the first day after the last day in the Week Calendar view as the first day. Not recordable.

For example, if the last day of the week is June 17, the first day of the week is June 18 after this command.

Syntax

ScrollNext ()

See Also

ScrollLeft
ScrollPrior
ScrollRight

ScrollPrior

Purpose

Displays the day before the first day in the Week Calendar view as the last day. Not recordable.

For example, if the first day of the week is June 17, the last day of the week is June 16 after this command.

Syntax

ScrollPrior ()

See Also

ScrollLeft
ScrollNext
ScrollRight

ScrollRight

Purpose

Displays the next day in the Week calendar view.

Syntax

ScrollRight ()

See Also

ScrollLeft
ScrollNext
ScrollPrior

Route...
Select the right-arrow button in the upper right corner of the Week Calendar view.

SelectDown

Purpose

Moves the insertion point down one line and selects the text in between.

Syntax

SelectDown ()

Route...
Press Shift+Down Arrow.

SelectLeft

Purpose

Moves the insertion point one character to the left and selects the character.

Syntax

SelectLeft ()

Route...
Press Shift+Left Arrow.

SelectLeftWord

Purpose

Moves the insertion point to the beginning of the previous word and selects the text in between.

Syntax

SelectLeftWord ()

Route...
Press Shift+Ctrl+Left Arrow.

SelectPageDown

Purpose

Moves the insertion point from its current position to the first line of the next screen, and selects the text in between.

Syntax

SelectPageDown ()

See Also

SelectPageUp

Route...
Press Shift+Page Down.

SelectPageUp

Purpose

Moves the insertion point from its current position to the first line of the previous screen, and selects the text in between.

Syntax

SelectPageUp ()

See Also

SelectPageDown

Route...
Press Shift+Page Up.

SelectRight

Purpose

Moves the insertion point one character to the right and selects the character.

Syntax

SelectRight ()

Route...
Press Shift+Right Arrow.

SelectRightWord

Purpose

Moves the insertion point to the beginning of the next word and selects the text in between.

Syntax

SelectRightWord ()

Route...
Press Shift+Ctrl+Right Arrow.

SelectToBegLine

Purpose

Moves the insertion from its current position to the beginning of the line and selects the text in between.

Syntax

SelectToBegLine ()

Route...
Press Shift+Home.

SelectToBegText

Purpose

Moves the insertion point from its current position to the beginning of a message box, and selects the text in between.

Syntax

SelectToBegText ()

Route...
Press Ctrl+Shift+Home.

SelectToEndLine

Purpose

Moves the insertion point from its current position to the end of a line, and selects the text in between.

Syntax

SelectToEndLine ()

Route...
Press Shift+End.

SelectToEndText

Purpose

Moves the insertion point from its current position to the end of a message box, and selects the text in between.

Syntax

SelectToEndText ()

Route...
Press Ctrl+Shift+End.

SelectUp

Purpose
Moves the insertion point up one line and selects the text in between.

Syntax
SelectUp ()

Route...
Press Shift+Up Arrow.

SendAppointment

Purpose
Schedules an appointment. Not recordable.

Syntax
string SendAppointment (To: *string*; Subject: *string*; StartDay: *numeric*; StartMonth: *numeric*; StartYear: *numeric*; StartMinute: *numeric*; StartHour: *numeric*; EndDay: *numeric*; EndMonth: *numeric*; EndYear: *numeric*; EndMinute: *numeric*; EndHour: *numeric*; Message: *string*; Attach: *string*; Place: *string*; CC: *string*; BC: *string*; From: *string*; ViewName: *string*; AttachListHasDisplayNames: *enum*; UserID: *string*; MessageIDs: *string*; MessageIsFile: *enum*; SubjectIsFile: *enum*; ViewNameIsFile: *enum*)

Return Value
string Message ID for tracking an Appointment item.

Parameters
To: *string*
User ID of one or more appointment recipients. Separate multiple recipients with commas.

Subject: *string* (optional)
Parameter also accepts a filename. See SubjectIsFile parameter.

StartDay: *numeric* (optional)

StartMonth: *numeric* (optional)

StartYear: *numeric* (optional)

StartMinute: *numeric* (optional)

StartHour: *numeric* (optional)

EndDay: *numeric* (optional)

EndMonth: *numeric* (optional)

EndYear: *numeric* (optional)

EndMinute: *numeric* (optional)

EndHour: *numeric* (optional)

Message: *string* (optional)
Parameter also accepts a filename. See MessageIsFile parameter.

Attach: *string* (optional)
Separate multiple attachments with commas (see AttachListHasDisplayNames parameter).

Place: *string* (optional)

CC: *string* (optional)
User ID of one or more carbon copy recipients. Separate multiple recipients with commas.

BC: *string* (optional)
User ID of one or more blind copy recipients. Separate multiple recipients with commas.

From: *string* (optional)
Name of person scheduling the appointment.

ViewName: *string* (optional)
Name of an Appointment view, such as "Meeting w/attach", or the name of a custom Aappointment view. Parameter also accepts a filename. See ViewNameIsFile parameter.

AttachListHasDisplayNames: *enum* (optional)
If Yes!, the Attach parameter must include the filename and display name for each attachment (in that order).

No!
Yes!

UserID: *string* (optional)
Use this parameter to schedule an appointment from a proxy mailbox.

MessageIDs: *string* (optional)
Attach encapsulated items. Separate multiple message IDs with a comma (see ItemListGetItem).

MessageIsFile: *enum* (optional)
Message parameter contains a filename.

No!
Yes!

SubjectIsFile: *enum* (optional)
Subject parameter contains a filename.

No!
Yes!

ViewNameIsFile: *enum* (optional)
ViewName parameter contains a filename.

No!
Yes!

SendMail

Purpose
Sends a Mail item. Not recordable.

Syntax
string SendMail (To: *string*; Subject: *string*; Message: *string*; Attach: *string*; CC: *string*; BC: *string*; From: *string*; IsRouted: *enum*; ViewName: *string*; AttachListHasDisplayNames: *enum*; UserID: *string*; MessageIDs: *string*; MessageIsFile: *enum*; SubjectIsFile: *enum*; ViewNameIsFile: *enum*)

Return Value
string Message ID for tracking a Mail item.

Parameters
To: *string*
User ID of one or more Mail item recipients. Separate multiple users with commas.

Subject: *string* (optional)
Parameter also accepts a filename. See SubjectIsFile parameter.

Message: *string* (optional)
Parameter also accepts a filename. See MessageIsFile parameter.

Attach: *string* (optional)
Separate multiple attachments with commas (see AttachListHasDisplayNames parameter).

CC: *string* (optional)
User ID of one or more carbon copy recipients. Separate multiple users with commas.

BC: *string* (optional)
User ID of one or more blind copy recipients. Separate multiple users with commas.

From: *string* (optional)
Name of person sending the Mail item.

IsRouted: *enum* (optional)
Route Mail item to two or more users in turn.
No!
Yes!

ViewName: *string* (optional)
Name of a Mail view. For example, "Expanded Mail", or the name of a custom Mail view. Parameter also accepts a filename. See ViewNameIsFile parameter.

AttachListHasDisplayNames: *enum* (optional)
If Yes!, the Attach parameter must include the filename and display name for each attachment (in that order).
No!
Yes!

UserID: *string* (optional)
Use this parameter to send a Mail item from a proxy mailbox.

MessageIDs: *string* (optional)
Attach encapsulated items. Separate multiple message IDs with commas (see ItemListGetItem).

MessageIsFile: *enum* (optional)
Message parameter contains a filename.
- No!
- Yes!

SubjectIsFile: *enum* (optional)
Subject parameter contains a filename.
- No!
- Yes!

ViewNameIsFile: *enum* (optional)
ViewName parameter contains a filename.
- No!
- Yes!

SendNote

Purpose
Sends a Note item. Not recordable.

Syntax
string SendNote (To: *string*; Subject: *string*; StartDay: *numeric*; StartMonth: *numeric*; StarYear: *numeric*; Note: *string*; Attach: *string*; CC: *string*; BC: *string*; From: *string*; ViewName: *string*; AttachListHasDisplayNames: *enum*; UserID: *string*; MessageIDs: *string*; MessageIsFile: *enum*; SubjectIsFile: *enum*; ViewNameIsFile: *enum*)

Return Value
string Message ID for tracking a Note item.

Parameters
To: *string*
User ID of one or more Note recipients. Separate multiple users with commas.

Subject: *string* (optional)
Parameter also accepts a filename. See SubjectIsFile parameter.

StartDay: *numeric* (optional)

StartMonth: *numeric* (optional)

StartYear: *numeric* (optional)

Note: *string* (optional)

Attach: *string* (optional)
Separate multiple attachments with commas (see AttachListHasDisplayNames parameter).

CC: *string* (optional)
User ID of one or more carbon copy recipients. Separate multiple users with commas.

BC: *string* (optional)
User ID of one or more blind copy recipients. Separate multiple users with commas.

From: *string*
Name of person sending the notice.

ViewName: *string* (optional)
Name of a Note view. For example, "Notice w/attach", or the name of a custom Note view. Parameter also accepts a filename. See ViewNameIsFile parameter.

AttachListHasDisplayNames: *enum* (optional)
If Yes!, the Attach parameter must include the filename and display name for each attachment (in that order).
No!
Yes!

UserID: *string* (optional)
Use this parameter to send a Note item from a proxy mailbox.

MessageIDs: *string* (optional)
Attach encapsulated items. Separate multiple message IDs with commas (see ItemListGetItem).

MessageIsFile: *enum*
Message parameter contains a filename.
- No!
- Yes!

SubjectIsFile: *enum*
Subject parameter contains a filename.
- No!
- Yes!

ViewNameIsFile: *enum*
ViewName parameter contains a filename.
- No!
- Yes!

SendOptions

F

Purpose

Sets Send Options for the current item (see SendMail for example). Not all options are available for every item.

Syntax

SendOptions (Priority: *enum*; Notify: *enum*; ReturnOnOpen: *enum*; ReturnOnAccept: *enum*; ReturnOnCompleted: *enum*; ReturnOnDelete: *enum*; StatusInfo: *enum*; ReplyRequested: *enum*; ReplyRequestedDays: *numeric*; ExpireDays: *numeric*; AutoDelete: *enum*)

Parameters

Priority: *enum* (optional)

High!
Low!
Normal!

Notify: *enum* (optional)

Notify recipients when an item arrives. Notify must be active.

No!
Yes!

ReturnOnOpen: *enum* (optional)

Return notification options. Notify must be active.

Mail!
MailAndNotify!
None!
Notify!

ReturnOnAccept: *enum* (optional)

Return notification options. Notify must be active.

Mail!
MailAndNotify!
None!
Notify!

ReturnOnCompleted: *enum* (optional)

Return notification options. Notify must be active.

Mail!
MailAndNotify!
None!
Notify!

ReturnOnDelete: *enum* (optional)

Return notification options. Notify must be active.

Mail!
MailAndNotify!
None!
Notify!

StatusInfo: *enum* (optional)

Information to track when a Mail or Phone item is sent.
- AllInfo!
- Delivered!
- None!
- Opened!

ReplyRequested: *enum* (optional)
Request reply on the first line of the message box.
- None!
- WhenConvenient!
- WithinDays!

ReplyRequestedDays: *numeric* (optional)
Number of days. The recipient sees: **Reply Requested:** By xx/xx/xx.

ExpireDays: *numeric* (optional)
Number of days a mail or phone message remains in a recipient's In Box.

AutoDelete: *enum* (optional)
Delete the item from the sender's Out Box when it is deleted from the recipient's In Box.
- No!
- Yes!

See Also
- PrefAdvanced
- SendOptions

Route...
Open an item, choose Send, Send Options, select options.

SendOptionsAdvanced

F

Purpose

Sets Advanced Send Options for the current item.

Syntax

SendOptionsAdvanced (Security: *enum*; ConcealSubject: *enum*; DeliverDelay: *numeric*; InsertOutBox: *enum*; ConvertAttachments: *enum*)

Parameters

Security: *enum* (optional)
Security label on the first line of a message box.

- Confidential!
- ForYourEyesOnly!
- Normal! (no security label)
- Proprietary!
- Secret!
- TopSecret!

ConcealSubject: *enum* (optional)
Conceal In Box and Out Box subject lines.

- No!
- Yes!

DeliverDelay: *numeric* (optional)
Number of days to delay delivery.

InsertOutBox: *enum* (optional)
Insert items in sender's Out Box.

- No!
- Yes!

ConvertAttachments: *enum* (optional)
Convert attached files through a gateway.

- No!
- Yes!

See Also

PrefAdvanced
SendOptions

Route...
Open an item, choose Send, Send Options, Advanced, select options.

SendOptionsDlg

Purpose

Displays the Send Options dialog box of the current item. For example, for Task items, this command displays the Task Send Options dialog box.

Syntax

SendOptionsDlg ()

See Also

SendOptions
SendOptionsAdvanced

Route...
Open an item, choose Send, Send Options.

SendPhone

Purpose
Sends a Phone item. Not recordable.

Syntax
string SendPhone (To: *string*; CallerName: *string*; CallerCompanyName: *string*; CallerPhone: *string*; Telephone: *enum*; PleaseCall: *enum*; WillCallAgain: *enum*; ReturnedYourCall: *enum*; WantsToSeeYou: *enum*; CameToSeeYou: *enum*; Urgent: *enum*; Subject: *string*; Message: *string*; Attach: *string*; CC: *string*; BC: *string*; From: *string*; ViewName: *string*; AttachListHasDisplayNames: *enum*; UserID: *string*; MessageIDs: *string*; MessageIsFile: *enum*; SubjectIsFile: *enum*; ViewNameIsFile: *enum*)

Return Value
string Message ID for tracking an Appointment item.

Parameters
To: *string*
User ID of Phone item recipients. Separate multiple users with commas.

CallerName: *string* (optional)

CallerCompanyName: *string* (optional)

CallerPhone: *string* (optional)

Telephoned: *enum* (optional)
No!
Yes!

PleaseCall: *enum* (optional)
Return phone call.
No!
Yes!

WillCallAgain: *enum* (optional)
No!
Yes!

ReturnedYourCall: *enum* (optional)
No!
Yes!

WantsToSeeYou: *enum* (optional)
No!
Yes!

CameToSeeYou: *enum* (optional)
No!
Yes!

Urgent: *enum* (optional)

No!
Yes!

Subject: *string* (optional)
Parameter also accepts a filename. See SubjectIsFile parameter.

Message: *string* (optional)
Parameter also accepts a filename. See MessageIsFile parameter.

Attach: *string* (optional)
Separate multiple attachments with commas (see AttachListHasDisplayNames parameter).

CC: *string* (optional)
User ID of carbon copy recipients. Separate multiple users with commas.

BC: *string* (optional)
User ID of blind copy recipients. Separate multiple users with commas.

From: *string* (optional)
Name of person sending the Phone item.

ViewName: *string* (optional)
Name of a Phone view, such as "Phone Message w/attach", or the name of a custom Phone view. Parameter also accepts a filename. See ViewNameIsFile parameter.

AttachListHasDisplayNames: *enum* (optional)
If Yes!, the Attach parameter must include the filename and display name for each attachment (in that order).
No!
Yes!

UserID: *string* (optional)
Use this parameter to send a Phone item from a proxy mailbox.

MessageIDs: *string* (optional)
Attach encapsulated items. Separate multiple message IDs with commas (see ItemListGetItem).

MessageIsFile: *enum* (optional)
Message parameter contains a filename.
No!
Yes!

SubjectIsFile: *enum* (optional)
Subject parameter contains a filename.
No!
Yes!

ViewNameIsFile: *enum* (optional)
ViewName parameter contains a filename.
No!
Yes!

See Also

CheckboxSelect

SendTask

Purpose
Sends a Task item. Not recordable.

Syntax
string SendTask (To: *string*; Subject: *string*; CategoryAndPriority: *string*; StartDay: *numeric*; StartMonth: *numeric*; StartYear: *numeric*; DueDay: *numeric*; DueMonth: *numeric*; DueYear: *numeric*; Message: *string*; Attach: *string*; CC: *string*; BC: *string*; From: *string*; IsRouted: *enum*; ViewName: *string*; AttachListHasDisplayNames: *enum*; UserID: *string*; MessageIDs: *string*; MessageIsFile: *enum*; SubjectIsFile: *enum*; ViewNameIsFile: *enum*)

Return Value
string Message ID for tracking an Appointment item.

Parameters
To: *string*
User ID of Task recipients. Separate multiple recipients with commas.

Subject: *string* (optional)
Parameter also accepts a filename. See SubjectIsFile parameter.

CategoryAndPriority: *string* (optional)

StartDay: *numeric* (optional)

StartMonth: *numeric* (optional)

StartYear: *numeric* (optional)

DueDay: *numeric* (optional)
Day to complete a task.

DueMonth: *numeric* (optional)
Month to complete a task.

DueYear: *numeric* (optional)
Year to complete a task.

Message: *string* (optional)
Parameter also accepts a filename. See MessageIsFile parameter.

Attach: *string* (optional)
Separate multiple attachments with commas (see AttachListHasDisplayNames parameter).

CC: *string* (optional)
User ID of carbon copy recipients. Separate multiple recipients commas.

BC: *string* (optional)
User ID of blind copy recipients. Separate multiple recipients commas.

From: *string* (optional)
Name of person sending the Task item.

IsRouted: *enum* (optional)
Route Task item to two or more users in turn.
No!
Yes!

ViewName: *string* (optional)
Name of a Mail view, such as "Expanded Mail", or the name of a custom Mail view. Parameter also accepts a filename. See ViewNameIsFile parameter.

AttachListHasDisplayNames: *enum* (optional)
If Yes!, the Attach parameter must include the filename and display name for each attachment (in that order).
No!
Yes!

UserID: *string* (optional)
Use this parameter to send a Task item from a proxy mailbox.

MessageIDs: *string* (optional)
Attach encapsulated items. Separate multiple message IDs with commas (see ItemListGetItem).

MessageIsFile: *enum* (optional)
Message parameter contains a filename.
No!
Yes!

SubjectIsFile: *enum* (optional)
Subject parameter contains a filename.
No!
Yes!

ViewNameIsFile: *enum* (optional)
ViewName parameter contains a filename.
No!
Yes!

ShelfIconArrange

Purpose

Arranges Shelf icons. Call from the Main Window or any view.

Syntax

ShelfIconArrange ()

See Also

ShelfIconDelete

Route...
From the Main Window, choose Window, Arrange Icons.

ShelfIconDelete

Purpose

Deletes a Shelf icon.

Call from the Main Window or any view. If you do not pass an icon name (see IconName parameter), the selected icon is deleted. A selected icon remains selected if you click another Window or press Alt+Tab.

A run-time error occurs if no icon is selected, or if you pass an icon name that does not exist (see ONERROR or ONERROR CALL).

Syntax

ShelfIconDelete (IconName: *string*)

Parameters

IconName: *string* (optional)

See Also

ShelfIconArrange
ShelfNewIconDlg
ShelfIconPropertiesDlg
ShelfIconOpen

Route...
Select a Shelf icon, choose Edit, Delete.

ShelfIconOpen

Purpose

Opens a Shelf icon.

Call from the Main Window or from a view. If you do not pass an icon name (see IconName parameter), the selected icon opens. A selected icon remains selected if you click another Window or press Alt+Tab.

A run-time error occurs if no icon is selected, or if you pass an icon name that does not exist (see ONERROR or ONERROR CALL).

Syntax

ShelfIconOpen (IconName: *string*)

Parameters

IconName: *string*

See Also

ShelfIconArrange
ShelfIconDelete
ShelfIconPropertiesDlg
ShelfNewIconDlg

Route...
Select a Shelf icon, Choose File, Open.

ShelfIconPropertiesDlg

F

Purpose

Displays the Shelf Icon Properties dialog box.

Call from the Main Window or from a view. If you do not pass an icon name (see IconName parameter), the Shelf Icon Properties dialog box opens for the selected icon. A selected icon remains selected if you click another Window or press Alt+Tab.

A run-time error occurs if no icon is selected, or if you pass an icon name that does not exist (see ONERROR or ONERROR CALL).

Syntax

ShelfIconPropertiesDlg (IconNme: *string*)

Parameters

IconName: *string*
Name of a Shelf icon.

See Also

ShelfIconArrange
ShelfIconDelete
ShelfIconOpen
ShelfNewIconDlg

Route...
Select a Shelf icon, choose File, Properties.

ShelfNewIconDlg

Purpose

Displays the New Shelf Icon dialog box.

Syntax

ShelfNewIconDlg ()

See Also

ShelfIconArrange
ShelfIconDelete
ShelfIconOpen
ShelfIconPropertiesDlg

Route...
From the Main Window, choose File, New.

SortDlg

Purpose
Displays the Sort dialog box to sort In Box or Out Box items.

Syntax
SortDlg ()

Route...
Open In Box or Out Box, choose View, Sort.

SoundAnnotationList

Purpose

Displays the Sound Annotations dialog box to play or save a sound, or get the sound annotation properties of a sound.

Syntax

SoundAnnotationList ()

Route...
Open an In Box or Out Box item, choose File, Sounds.

SoundAnnotationPlay

F

Purpose
Plays a sound annotation.

Syntax
SoundAnnotationPlay (Index: *numeric*)

Parameters
Index: *numeric*
Index number (zero-based) of a sound annotation.

Route...
Open an In Box or Out Box item, choose File, Sounds, select a sound annotation, Play.

Speller

Purpose
Displays the Spell Checker dialog box.

Syntax
Speller ()

See Also
Thesaurus

Route...
Open an item, place the insertion point in a text box, choose Tools, Speller.

SplitBar

Purpose

Selects the bar separating folders and items to expand the width of the folder or item list box.

Syntax

SplitBar ()

Route...
Open In Box or Out Box, choose View, Split.

TabNext

Purpose
Moves the insertion point forward a specified number of tab stops. Not recordable.

Syntax
TabNext (Count: *numeric*)

Parameters
Count: *numeric* (optional)
Number of tab stops. Default: 1.

See Also
FocusSet
TabPrevious

TabPrevious

Purpose
Moves the insertion point backward a specified number of tab stops. Not recordable.

Syntax
TabPrevious (Count: *numeric*)

Parameters
Count: *numeric* (optional)
Number of tab stops. Default: 1.

See Also
FocusSet
TabNext

TextSetAuthority

Purpose

Inserts the name of a person or entity in an Authority text box (available only in a customized Appointment view). Not recordable.

Syntax

TextSetAuthority (AuthorityText: *string*; Append: *boolean*)

Parameters

AuthorityText: *string*
Person or entity responsible for a meeting.

Append: *boolean* (optional)
Default: FALSE.

FALSE (replace existing text)
TRUE (append to existing text)

TextSetBC

Purpose

Inserts a user ID in a BC text box. Not recordable.

Syntax

TextSetBC (BCText: *string*; Append: *boolean*)

Parameters

BCText: *string*

User ID of blind copy recipients. Separate multiple users with commas.

Append: *boolean* (optional)

Default: FALSE.

FALSE (replace existing text)

TRUE (append to existing text)

TextSetCallerName

Purpose

Inserts a caller's name in the Caller text box of a Phone Message view. Not recordable.

Syntax

TextSetCallerName (CallerName: *string*; Append: *boolean*)

Parameters

CallerName: *string*

Append: *boolean* (optional)
Default: FALSE.

FALSE (replace existing text)
TRUE (append to existing text)

TextSetCategoryAndPriority

Purpose

Inserts a character in the Priority text box of a Task view. Not recordable.

Syntax

TextSetCategoryAndPriority (CatAndPrior: *string*; Append: *boolean*)

Parameters

CatAndPrior: *string*
For example: "A5" (A is the category and 5 the priority).

Append: *boolean* (optional)
Default: FALSE.
FALSE (replace existing text)
TRUE (append to existing text)

TextSetCC

Purpose

Inserts a user ID in a CC text box. Not recordable.

Syntax

TextSetCC (CCText: *string*; Append: *boolean*)

Parameters

BCText: *string*

User ID of carbon copy recipients. Separate multiple users with commas.

Append: *boolean* (optional)

Default: FALSE.

FALSE (replace existing text)

TRUE (append to existing text)

TextSetCompany

Purpose

Inserts a company's name in the Company text box of a Phone view. Not recordable.

Syntax

TextSetCompany (CompanyText: *string*; Append: *boolean*)

Parameters

CompanyText: *string*

Append: *boolean* (optional)
Default: FALSE.

FALSE (replace existing text)
TRUE (append to existing text)

TextSetDateTime

Purpose

Inserts a date and time in the Appointment date and time text boxes of an Appointment view. Not recordable.

If Display Event Length is set to Duration, do not use End parameters. If Display Event Length is set to End Date and End Time, do not use Duration parmeters. See PrefAppointmentTime.

Syntax

TextSetDateTime (StartDay: *numeric*; StartMonth: *numeric*; StartYear: *numeric*; StartMinute: *numeric*; StartHour: *numeric*; EndDay: *numeric*; EndMonth: *numeric*; EndYear: *numeric*; EndMinute: *numeric*; EndHour: *numeric*; DurationMinutes: *numeric*; DurationHours: *numeric*)

Parameters

StartDay: *numeric* (optional)

StartMonth: *numeric* (optional)

StartYear: *numeric* (optional)

StartMinute: *numeric* (optional)

StartHour: *numeric* (optional)
Military time (24-hour clock).

EndDay: *numeric* (optional)

EndMonth: *numeric* (optional)

EndYear: *numeric* (optional)

EndMinute: *numeric* (optional)

EndHour: *numeric* (optional)
Military time (24-hour clock).

DurationMinutes: *numeric* (optional)

DurationHours: *numeric* (optional)

See Also

PrefAppointmentTime

TextSetFrom

Purpose

Inserts a user's name in the From text box. The logged in user name is included in parenthesis. Not recordable.

Syntax

TextSetFrom (FromText: *string)*; Append: *boolean*)

Parameters

FromText: *string*

Append: *boolean* (optional)
Default: FALSE.

FALSE (replace existing text)
TRUE (append to existing text)

TextSetMessage

Purpose
Inserts text in a Message text box. Not recordable.

Syntax
TextSetMessage (MessageText: *string*; Append: *boolean*)

Parameters
MessageText: *string*

Append: *boolean* (optional)
Default: FALSE.
FALSE (replace existing text)
TRUE (append to existing text)

TextSetPhoneNumber

Purpose

Inserts text in the Phone text box of a Phone Message view. Not recordable.

Syntax

TextSetPhoneNumber (PhoneNumber: *string*; Append: *boolean*)

Parameters

PhoneNumber: *string*

Append: *boolean* (optional)
Default: FALSE.

FALSE (replace existing text)
TRUE (append to existing text)

TextSetPlaceName

Purpose
Inserts a location in the Place text box of an Appointment view. Not recordable.

Syntax
TextSetPlaceName (PlaceName: *string*; Append: *boolean*)

Parameters
PlaceName: *string*

Append: *boolean* (optional)
Default: FALSE.
FALSE (replace existing text)
TRUE (append to existing text)

TextSetSubject

Purpose

Inserts a subject in a Subject text box. Not recordable.

Syntax

TextSetSubject (SubjectText: *string*; Append: *boolean*)

Parameters

SubjectText: *string*

Subject of a mail or note message, an appointment, or a task.

Append: *boolean* (optional)

Default: FALSE.

FALSE (replace existing text)

TRUE (append to existing text)

TextSetTo

Purpose

Inserts a user ID in a To text box. Not recordable.

Syntax

TextSetTo (ToText: *string*; Append: *boolean*)

Parameters

ToText: *string*

Append: *boolean* (optional)
Default: FALSE.

FALSE (replace existing text)
TRUE (append to existing text)

Thesaurus

Purpose
Displays the Thesaurus dialog box.

Syntax
Thesaurus ()

See Also
Speller

Route...
Place the insertion point on a word, choose Tools, Thesaurus.

Type

F

Purpose
Inserts text into a text box at the insertion point.

Syntax
Type (Text: *string*)

Parameters
Text: *string*

See Also
TypeChar
WPCharBox

Route...
Place the insertion point in a text box, type text.

TypeChar

Purpose
Inserts a WordPerfect character at the insertion point.

Syntax
TypeChar (CharSet: *numeric*; CharNum: *numeric*)

Parameters
CharSet: *numeric*
Number of a WordPerfect character set.

CharNum: *numeric*
Number of a character in a WordPerfect character set.

See Also
Type
WPCharBox

Route...
Place the insertion point in a text box, choose Edit, Font, WP Characters, select a character set and character.

UserButtonAction

Purpose
Activates a button control on a customized view.

Syntax
UserButtonAction (ButtonName: *string*)

Parameters
ButtonName: *string*

Route...
Open a customized view, choose a custom button control.

UserFunction

Purpose

Executes a command from another application. Primarily used by third-party developers. Not recordable.

Syntax

UserFunction (Command: *string*)

Parameters

Command: *string*

Command to execute in a third-party DLL. The command name should take the form *xxxx: name* (xxxx represents a four-character identifier).

UtilWPFileRead

Purpose

Returns the contents of a WordPerfect file in a variable. Use Type to write the contents of the variable into a GroupWise text box. Not recordable.

Syntax

UtilWPFileRead (FileName: *string*; FileContents: *variable*)

Parameters

FileName: *string*
Name of a file to read.

FileContents: *variable*

See Also

Type
UtilWPFileWrite

UtilWPFileWrite

Purpose

Writes text to a WordPerfect file, which can be read by UtilWPFileRead and inserted into a GroupWise text box with the Type command. Not recordable.

Syntax

UtilWPFileWrite (FileName: *string*; FileContents: *string*)

Parameters

FileName: *string*

FileContents: *string*

See Also

Type
UtilWPFileRead

ViewHideMenu

Purpose
Hides or shows the Caption and Menu bar on a Calendar view.

Syntax
ViewHideMenu (State: *enum*)

Parameters
State: *enum*
If no parameter is specified, acts as a toggle.
Hide!
Show!

Route...
Open a Calendar view, choose View, Hide Menu.

ViewMaximize

Purpose

Enlarges a window or view to full size. Items reduced to an icon can only be restored (see ViewRestore), they cannot be maximized.

Syntax

ViewMaximize ()

See Also

ViewMinimize
ViewRestore

Route...
Click a control-menu box, choose Maximize.

ViewMinimize

Purpose

Reduces a window or view to an icon.

Syntax

ViewMinimize ()

See Also

ViewMaximize
ViewRestore

Route...
Click a control-menu box, choose Minimize.

ViewOnTop

Purpose

Displays In Box, Out Box, Trash, or Calendar views and icons on top of other views.

Syntax

ViewOnTop (OnTop: *enum*)

Parameters

OnTop: *enum*

If no parameter is specified, acts as a toggle.

No!

Yes!

See Also

ViewMinimize

Route...
Open In Box, Out Box, Trash, or a Calendar view, click the System Menu button, choose Always on Top.

ViewOpen

F

Purpose

Opens a default view such as a Mail or Calendar view.

Syntax

ViewOpen (ViewType: *enum*; IsGroupItem: *numeric*)

Parameters

ViewType: *enum*

- Appointment!
- Calendar!
- InBox!
- Mail!
- Note!
- OutBox!
- PhoneMessage!
- Task!
- Trash!

IsGroupItem: *enum* (optional)

- No! (personal item)
- Yes! (default)

See Also

NewAppointment
NewMail
NewNote
NewPhone
NewTask
OpenCalendar
OpenInBox
OpenOutBox
OpenTrashWindow
PrefViewDefaults
ViewOpenFile
ViewOpenNamed

Route...
Choose Send, New Mail, New Appointment, New Task, New Note, or New Phone Message.

or

Choose Window, In Box, Out Box, Calendar, or Trash.

ViewOpenDlg

Purpose
Displays the Open View dialog box.

Syntax
ViewOpenDlg ()

See Also
ViewOpen

Route...
From the Main Window, choose File, Open View.

ViewOpenFile

Purpose
Opens a view by its file name.

Syntax
ViewOpenFile (FileName: *string*)

Parameters
FileName: *string*
Include full path.

See Also
ViewOpen

Route...
From the Main Window, choose File, Open View, select a view.

ViewOpenNamed

F

Purpose
Opens a view by menu name and type.

Syntax
ViewOpenNamed (ViewName: *string*; ViewType: *enum*)

Parameters
ViewName: *string*
Example: "Mail" or "Appointment w/attach"

ViewType: *enum*
- Appointment!
- Calendar!
- InBox!
- Mail!
- Note!
- OutBox!
- PhoneMessage!
- Task!
- Trash!

See Also
ViewOpen
ViewOpenFile

Route...
From the Main Window, double-click an icon or select an item from a popup list below an icon.

ViewRestore

Purpose
Restores a window to its previous size and position.

Syntax
ViewRestore ()

See Also
ViewMaximize
ViewMinimize

Route...
Click a control-menu box, choose Restore.

ViewSculpturing

Purpose

Turns sculpturing on or off (see Sculpturing in GroupWise Help).

Syntax

ViewSculpturing (State: *enum*)

Parameters

State: *enum* (optional)

If no parameter is specified, acts as a toggle.

Off!

On!

Route...
Open an item, choose View, Sculpturing.

ViewSwitch

F

Purpose

Opens a view from a view, copies the information from the old view to the new, and then closes the old view.

You cannot switch from a Calendar view to and Item view or vice versa.

Syntax

ViewSwitch (Filename: *string*; ViewType: *enum*; ViewName: *string*)

Parameters

Filename: *string* (optional)
Open a view by filename. With this parameter, do not use ViewType and ViewName.

ViewType: *enum* (optional)
Open a view by menu name and type. With this parameter, do not use Filename.

- Appointment!
- Calendar!
- InBox!
- Mail!
- Note!
- OutBox!
- PhoneMessage!
- Task!
- Trash!

ViewName: *string* (optional)
Menu name such as "Mail" or "Appointment w/attach". See ViewType parameter.

Route...
Open an Item or Calendar view, choose View, Switch View, select a view.

ViewSwitchDlg

Purpose
Displays the Switch View dialog box.

Syntax
ViewSwitchDlg ()

See Also
ViewSwitch

Route...
Open an Item or Calendar view, choose View, Switch View.

ViewUseSystemColors

Purpose
Applies the Windows color scheme (System Colors) to the current view.

Syntax
ViewUseSystemColors (State: *enum*)

Parameters
State: *enum* (optional)
If no parameter is specified, acts as a toggle.
- No!
- Yes!

Route...
Open a view, choose View, System Colors.

WPCharBox

Purpose
Displays the WordPerfect Characters dialog box.

Syntax
WpCharBox ()

See Also
TypeChar

Route...
Open a new item, place the insertion point in a text box, choose Edit, Font, WP Characters.

// (Comment)

Purpose

The compiler ignores all text between // and the next hard return [HRt].

Comment text does not affect macro execution. The comment line, including spaces, has a maximum length of 512 characters.

Syntax

// *...comments...* [HRt]

APPACTIVATE

F

Purpose

Activates and displays a window.

Use APPLOCATE to return the window handle in a variable.

```
ASSIGN(hWnd; APPLOCATE("Program Manager"))
IF(hWnd)   // if hWnd not equal to 0
    APPACTIVATE(hWnd)
ELSE
    QUIT
ENDIF
```

Syntax

APPACTIVATE (*WindowHandle*)

Parameters

WindowHandle: *numeric*
The window handle of the window to activate (see APPLOCATE).

See Also

APPEXECUTE
APPLOCATE

APPEXECUTE

Purpose

Starts an application and displays it.

The macro terminates if the application is not successfully started. APPEXECUTE is the equivalent of the Windows Program Manager Run command.

Syntax

APPEXECUTE (*CommandLine*)

Parameters

CommandLine: *string*
The path and name of a program file.

See Also

APPACTIVATE
APPLOCATE
APPEXECUTEEXT

APPEXECUTEEXT

F

Purpose

Starts an application and specifies how its window is displayed.

Syntax

ReturnValue := APPEXECUTEEXT (*CommandLine; ShowWindow*)

Returns

The application handle (unique number) if the application is successfully started, or an error value less than 32 if not. For the list of error values, see Windows 3.1 WinExec documentation (Programmer's Reference, Volume 2: Functions). A partial list follows:

0 System out of memory or program file is corrupt.
2 File not found.
3 Path not found.
8 Insufficient memory to start application.
10 Incorrect Windows version.
11 Non-windows application.

Check error values with code similar to the following:

```
vError := APPEXECUTEEXT(CommandLine; ShowWindow)
SWITCH(vError)
    CASEOF 0: ...statement block...
    CASEOF 2: ...statement block...
ENDSWITCH
```

Parameters

CommandLine: *string*
The path and name of a program file.

ShowWindow: *numeric*
Specifies how an application's window is displayed. The values are:

0 Hide window. APPLOCATE and DDEINITIATE return the window handle of a hidden window. DDEEXECUTE sends a command string to a hidden window. APPACTIVATE activates a hidden window. Accelerator keys choose a hidden window's menu options.
1 Show window at normal size.
2 Minimize window to an icon.
3 Maximize window to full screen size.

See Also

APPACTIVATE
APPLOCATE
APPEXECUTE
DDEEXECUTE

APPLICATION

F

Purpose

Identifies, for the compiler, an application to use in a macro.

APPLICATION is a non-executable statement that can occur anywhere in a macro, but must precede product commands to the application it identifies.

Applications used in the macro automatically start unless already active. The user cannot terminate an application used in a macro can until the macro ends.

Syntax

APPLICATION (*ProductPrefix; ApplicationName;* Default!; *Language*)

Parameters

ProductPrefix: *string*
A two-character identifier that begins with a letter. The second character can be a letter or number. *ProductPrefix* directs product commands to non-default applications. It is also used in NEWDEFAULT and ENDAPP statements.

Use a period to attach *ProductPrefix* to a product command. If WordPerfect is the application and WP the product prefix, WP.AboutDlg is a valid command statement. A recorded macro returns a product prefix of A1, A2, A3, and so forth, depending on the number of applications involved.

ApplicationName: *string*
The name of an application. For example, "WPOffice" for GroupWise, or "WordPerfect" for WordPerfect 6.1 for Windows.

Default!
An enumeration that identifies a default application. Product commands to the default application do not require a product prefix. There can be only one default application.

Language: *string*
The application's language code. For example, "US" is the language code for United States English.

See Also

ENDAPP
NEWDEFAULT

APPLOCATE

Purpose

Returns a window handle.

APPLOCATE compares a name to the title bar text of all open windows. If a match is found, the window handle of the matching window is returned. If the search criteria matches more than one window, the window handle of the most recently opened window is returned. If no match is found, zero is returned.

You can use an asterisk (*) as a wildcard character at any position in the APPLOCATE parameter. If the parameter contains only an asterisk, APPLOCATE returns the window handle of the active window.

Syntax

ReturnValue := APPLOCATE (*WindowTitle*)

Returns

The window handle of the window found in the search, or 0 if no window is found.

Parameters

WindowTitle: *string*

The window title to find. The match between this parameter and the title must be exact unless you use an asterisk as a wildcard character. Titles are case sensitive.

See Also

APPACTIVATE
APPEXECUTE

ASSERT

F

Purpose

Creates a Cancel, Error, or Not Found condition.

A Cancel, Error, or Not Found condition stops a macro unless preceded by ONCANCEL, ONERROR, or ONNOTFOUND, which direct macro execution to a specified LABEL. ASSERT has no effect when preceded by CANCEL(Off!), ERROR(Off!), or NOTFOUND(Off!).

An error value is assigned to variable ErrorNumber when one of the three conditions occurs. The values are:

1 Cancel condition.
2 Error condition.
7 Not found condition.

You cannot assign a value to ErrorNumber.

Use code similar to the following to check all three conditions with one subroutine (see Subroutines).

```
ONERROR(Condition)
ONCANCEL(Condition)
ONNOTFOUND(Condition)

...other statements...

LABEL(Condition)
    SWITCH(ErrorNumber)
        CASEOF 1: ...statement block...
        CASEOF 2: ...statement block...
        CASEOF 7: ...statement block...
    ENDSWITCH
```

Syntax

ASSERT (*Condition*)

Parameters

Condition: *enum*
A condition to assert.
CancelCondition!
ErrorCondition!
NotFoundCondition!

CancelCondition! stops a macro unless preceded by ONCANCEL. ErrorCondition! stops a macro unless preceded by ONERROR. NotFoundCondition! stops a macro unless preceded by ONNOTFOUND.

See Also

CANCEL
ERROR
LABEL
NOTFOUND

ONCANCEL
ONERROR
ONNOTFOUND
RETURN
SWITCH

ASSIGN

F

Purpose

Assigns a value to a variable.

ASSIGN is equivalent to the assignment operator (:=). Consider the following four statements and their equivalents:

```
ASSIGN(A; B) // assign the contents of B to A
ASSIGN(A; 487) // assign 487 to A
ASSIGN(A; B + 487) // assign the contents of B + 487 to A
ASSIGN(A; "John Doe") // assign the name John Doe to A

A := B // assign the contents of B to A
A := 487 // assign 487 to A
A := B + 487 // assign the contents of B + 487 to A
A := "John Doe" // assign the name John Doe to A
```

Syntax

ASSIGN (*VariableName; Value*)

Parameters

VariableName: *variable*
Contains the contents of *Value*.

Value: *any*
A variable, constant, character string, or expression. The source line has a maximum length of 512 characters.

See Also

DISCARD
GLOBAL
INDIRECT
LOCAL
PERSIST
PERSISTALL

BEEP

Purpose

Causes a computer speaker or installed sound board to beep.

BEEP is commonly used with warning messages, or to signal when a macro pauses or finishes a task.

Syntax

BEEP

BIFFILEPATH

Purpose

Retrieves the path and filename of a Binary Initialization File (BIF).

See *GroupWise Software Developer's Kit* for more information about the BIF approach to initialization files.

Syntax

BIFFILEPATH (*BIFFilePath; BIFFileType*)

Parameters

BIFFilePath: *string*
The BIF path and filename is returned in this variable.

BIFFileType: *enum*
The BIF type that is returned in *BIFFilePath*. A user can have only two active BIF files.

Private!	A BIF file containing settings that apply to a single user.
Public!	A BIF file containing settings that are shared by many, possibly concurrent, users.

See Also

BIFINFO
BIFREAD
BIFWRITE

BIFINFO

F

Purpose

Retrieves information from a Binary Initialization File (BIF).

See *GroupWise Software Developer's Kit* for more information about the BIF approach to initialization files.

Syntax

BIFINFO (*Status; Group; Section; Item; DateTime; ItemFlags; ItemLength; ItemType; UseCount; ItemInfoType; BIFFile*).

Parameters

Status: *numeric*
The status of a BIF call is returned in this variable. The values are:

0	Call was successful.
-2	Length of Group, Section, or Item exceeds buffer size.
-3	Requested Group, Section, or Item not found.
-4	Unknown error.

Group: *string*
The name of a Group. If this parameter is not used, the number of Group names is returned in the UseCount parameter.

Section: *string*
The name of a Section. If this parameter is not used, the number of Section names in the specified Group is returned in the UseCount parameter.

Item: *string*
The name of an Item. If this parameter is not used, the number of Item names in the specified Group and Section is returned in the UseCount parameter.

DateTime: *variable*
The date and time the item was created is returned in this variable. If Group, Section, and Item are not used, the current date and time is returned.

ItemFlags: *variable*
The flags associated with an item are returned in this variable. If Group, Section, and Item are not used, 0 is returned.

ItemLength: *variable*
The length of an item entry is returned in this variable. If Group, Section, and Item are not used, 0 is returned.

ItemType: *variable*
The item type is returned in this variable. If Group, Section, and Item are not used, 0 is returned.

UseCount: *variable*
If Group, Section, or Item is not used, the number of Groups, Sections, or Items in a BIF is returned in this variable (see Group, Section, and Item parameters). Otherwise, 0 is returned.

ItemInfoType: *enum*
The standard BIF types to examine.

Both!	Look in both standard BIFs.	Only valid when counting Groups, Sections, or Items.
Private!	Look in Private BIF only.	
Public!	Look in Public BIF only.	

BIFFile: *string*
The name of a non-standard BIF.

See Also

BIFFILEPATH
BIFREAD
BIFWRITE

BIFREAD

F

Purpose

Reads a Binary Initialization File (BIF), and returns Group, Section, or Item names, or item value.

Group, the highest BIF level. For example, in the standard BIF, Group is associated with a WordPerfect product such as WP Macros, WP Shared Code, and WordPerfect.

Section, the next BIF level after Group, is associated with Group subdivisions. For example, WordPerfect Sections include features such as Ruler, Power Bar, Table of Authorities, Repeat Value, and Files.

Item, the lowest BIF level, is associated with a Section entry. For example, in the WordPerfect Section named Files, Items include Hyphenation Directory, Timed Document Backup, Macros Directory, and Template File.

Items contain values. For example, the value of Macros Directory may be C:\WPWIN60\MACROS.

Group, Section, and Item names are subject to change. See an application's documentation for more information.

Syntax

BIFREAD (*Status, Group; Section; Item; Value; ItemType; NameSeparator; NameListType; BIFFile*)

Parameters

Status: *variable*

The status of a BIF call is returned in this variable. The values are:

- 0 (or positive integer) Number of bytes read from the BIF.
- -1 *ItemType* does not match returned item type (see ItemType parameter).
- -2 Length of Group, Section, or Item exceeds buffer size.
- -3 Requested Group, Section, or Item was not found.
- -4 Unknown error.

Group: *string*

Contains a Group name. If *Group* is not used, the BIF Group names are returned in a variable (see Value parameter).

Section: *string*

Contains a Section name. If *Group* is used and *Section* is not, the BIF Section names for the specified Group are returned in a variable (see Value parameter).

Item: *string*

Contains an Item name. If *Group* and *Section* are used and *Item* is not, the BIF Item names for the specified Group and Section are returned in a variable (see Value parameter).

Value: *variable*

The *Group, Section,* or *Item* names are returned in this variable, depending on which one is not used. If *Group, Section,* and *Item* are used, the *Item* value is returned. For

example, given that "c:\wpwin60\macros\" is the default macros directory in a standard BIF,

```
BIFREAD(vStatus; "WordPerfect"; "Files"; "Macro Directory"; vEntry; AnsiString!; ; ;)
```

returns 18 to vStatus (the number of bytes returned in *Value*) and "C:\WPWIN60\MACROS\" to the Value parameter, vEntry.

ItemType: *variable*
The expected item type to return in a variable (see Value parameter). *ItemType* is not used if Group, Section, or Item is not used.

0	Any!
1	Boolean!
2	SignedByte!
3	UnsignedByte!
4	SignedWord!
5	UnsignedWord!
6	SignedDWord!
7	UnsignedDWord!
8	Float4!
9	Float8!
10	AnsiString!
11	WPWordString!
12	BagOfbits!

NameSeparator: *string* (optional)
A delimiter string that separates Group, Section, or Item names. The delimiter is not limited to a single character. If not used, the default is a semicolon. *NameSeparator* is not used when returning an item value (Group, Section, and Item are used).

NameListType: *enum* (optional)
A standard BIF to read. Standard BIFs are associated with WordPerfect products (see BIFFile parameter). The values are:

Both!	Look in both standard BIFs.
Private!	Look in Private BIF only.
Public!	Look in Public BIF only.

You cannot specify Public! to read an Item value. Public! is ignored unless you are reading a list (Group, Section, or Item names).

BIFFile: *string* (optional)
The name of a non-standard BIF (a user-defined or non-WordPerfect product BIF). If both parameters are used, *BifFile* overrrides *NameListType*. If neither parameter is used, the default is *NameListType*.

See Also

BIFFILEPATH
BIFINFO
BIFWRITE

BIFWRITE

F

Purpose

Writes data to a Binary Initialization File (BIF).

See BIFREAD Comments for a description of BIF Groups, Sections, Items, and Values.

F To write data to a BIF, you must use the Group, Section, Item, and Value parameters. If you do not specify a value for each parameter, you will delete a level or item from the BIF. See parameter descriptions below.

Syntax

BIFWRITE (*Status, Group; Section; Item; Value; ItemType; ItemFlags, BIFFile*)

Parameters

Status: *variable*
The status of a BIF call is returned in this variable. The values are:

0	(or positive integer) Number of bytes written to the BIF.
-1	Unknown Group, Section, or Item type.
-2	Requested Group, Section, or Item not found.
-3	Unknown error.

Group: *string* (optional)
A Group name. To delete a Group, specify the Group name and do not use the Section parameter. To delete all Groups, Sections, and Items, do not use the Group parameter.

Section: *string* (optional)
A Section name. The Group name must be used. To delete a Section, specify the Section name and do not use the Item parameter.

Item: *string* (optional)
An Item name. The Group and Section names must be used. To delete an Item, specify the Item name and do not use the Value parameter.

Value: *string* (optional)
The Item value to write to the BIF. For example,

```
BIFWRITE(vStatus; "WordPerfect"; "Files"; "Macro Directory"; "C:\WPWIN60\MACROS";
AnsiString!; 0; )
```

writes "C:\WPWIN60\MACROS" to a private BIF. It is the value of the "Macro Directory" item in the "Files" section of the "WordPerfect" group.

F Do not write to a standard BIF while the associated WordPerfect application is running. Standard BIFs remain open while Shared Code is running. Play the previous example from Macro Facility.

ItemType: *enum*
The *Value* data type to write to the BIF.

0 Any!
1 Boolean!
2 SignedByte!
3 UnsignedByte!
4 SignedWord!
5 UnsignedWord!
6 SignedDWord!
7 UnsignedDWord!
8 Float4! (32-bit float)
9 Float8! (64-bit float)
10 AnsiString!
11 WPWordString!
12 BagOfbits!

If you write a new value to a BIF item, you must specifiy the same type as the previous item. If not specified, *ItemType* is determined by the default value for each data type.

If *ItemType* does not match the current value type, it is converted if possible. If not converted properly, you may not be able to read the value back later (see BIFFILEREAD).

ItemFlags: *enum*
Flags (attributes) only affect the public BIF. Value 4 is processed when an item is read from a BIF. All other values are processed when Shared Code starts.

None!	No flags.
DeletePrivate!	Removes an item in a private BIF that has the same name as an item in the public BIF.
OverridePrivate!	The public BIF item has precedence over an item with the same name in a private BIF. Normally, the opposite is true.
DistributeToPrivate!	An item is copied to the private BIF when opened.

BIFFile: *string*
Contains the name of a non-standard BIF to write to. If the BIF does not exist, it is created with specified Group, Section, and Item names, and Item value. To write to a private BIF, do not use this parameter (see BIFREAD, NameListType parameter).

See Also
BIFFILEPATH
BIFINFO
BIFREAD

BREAK

Purpose

Ends a loop, and directs macro execution to the first statement after the ending loop statement (see Loop Statements). In a SWITCH statement, BREAK directs macro execution to the first statement after ENDSWITCH.

BREAK bypasses the normal test expression of a loop or conditional statement. Statements after BREAK are ignored.

Syntax

BREAK

See Also

FOR
FOREACH
FORNEXT
NEXT
REPEAT
WHILE
SWITCH

CALL

F

Purpose

Calls a subroutine that ends with RETURN (see Subroutines).

RETURN directs macro execution to the statement that follows the subroutine's caller.

Syntax

CALL (*label*)

Parameters

Label: *label*

A LABEL, PROCEDURE, or FUNCTION name that begins with a letter and consists of one or more letters or numbers. The format for calling LABEL is,

```
CALL(LabelSub)

...other statements...

LABEL(LabelSub)
    ...statement block...
    RETURN
```

The formats for callng PROCEDURE and FUNCTION are,

```
CALL ProcSub(Parameter; Parameter)

...other statements...

PROCEDURE ProcSub(Parameter; Parameter)
    ...statementblock...
ENDPROC

CALL FuncSub(Parameter; Parameter)

...other statements...

FUNCTION FuncSub(Parameter; Parameter)
    ...statementblock...
ENDFUNC
```

See Also

CASE
CASE CALL
GO
INDIRECT
LABEL
RETURN

CANCEL

Purpose

Determines how a macro responds to a Cancel condition.

Create a Cancel condition by pressing Esc, or with ASSERT(CancelCondition!).

Syntax

CANCEL (*State*)

Parameters

State: *enum*

Specifies the Cancel state. The default is CANCEL(On!).

Off!	Overrides a Cancel condition.
On!	Stops a macro unless preceded by ONCANCEL, which directs macro execution to a specified LABEL.

See Also

ASSERT
ERROR
LABEL
NOTFOUND
ONCANCEL

CASE

F

Purpose

A conditional statement that tests for matching expressions. If a match is found, a LABEL is executed. See Conditional Statements.

CASE compares *Test* to a set of *Cases* (values). If the first comparison is true (if *Test* and *Case* match), the *label* following *Case* is executed. If the comparison is false, the next *Case* is evaluated and so forth. If no comparison is true, *DefaultLabel* is executed.

Syntax

The general form of a CASE statement is,

CASE (*Test*; {*Case*; *label*; *Case*; *Label...*}; *DefaultLabel*)

Parameters

Test: *any*
The control expression. Variables are assigned values by commands such as GETSTRING, GETNUMBER, or MENU.

Case: *any*
An expression (variable, constant, character) with a value that is usually assigned before the macro is compiled. It is possible to assign the value at run-time.

Label: *label*
The LABEL to execute if *Case* matches *Test*.

DefaultLabel: *label* (optional)
The LABEL to execute if no *Case* matches *Test*. If not specified, statement immediately following CASE is executed.

See Also

CASE CALL
IF
INDIRECT
LABEL
SWITCH

CASE CALL

F

Purpose

A conditional statement that tests for matching expressions. If a match is found, a LABEL is called. See Conditional Statements.

CASE CALL compares *Test* to a set of *Cases* (values). If the first comparison is true (if *Test* and *Case* match), the *label* following *Case* is executed. If the comparison is false, the next *Case* is evaluated and so forth. If no comparison is true, *DefaultLabel* is executed.

CASE is different from CASE CALL, in that a RETURN statement after LABEL directs macro execution to the statement that follows CASE CALL.

Syntax

The general form of a CASE CALL statement is,

CASE CALL (*Test*; {*Case*; *label*; *Case*; *Label...*}; *DefaultLabel*)

Parameters

Test: *any*
The control expression. Variables are assigned values by commands such as GETSTRING, GETNUMBER, or MENU.

Case: *any*
An expression (variable, constant, character) with a value that is usually assigned before the macro is compiled. It is possible to assign the value at run-time.

Label: *label*
The LABEL to execute if *Case* matches *Test*.

DefaultLabel: *label* (optional)
The LABEL to execute if no *Case* matches *Test*. If not specified, statement immediately following CASE CALL is executed.

See Also

CASE
IF
INDIRECT
LABEL
RETURN
SWITCH

CHAIN

F

Purpose

Calls (starts) another macro when the parent macro ends.

A CHAIN macro does not return to its caller (see RUN). A macro must be compiled before it is called by CHAIN.

Although CHAIN can occur anywhere in a macro, it is the last statement to execute. You can cancel CHAIN (macro is not executed) with a QUIT statement. For example,

```
CHAIN(?Pathmacros + "Test4.wcm")

...other statements...

SWITCH(Test)
    CASEOF 1: CHAIN(?Pathmacros + "Test1.wcm")
    CASEOF 2: CHAIN(?Pathmacros + "Test2.wcm")
    CASEOF 3: CHAIN(?Pathmacros + "Test3.wcm")
    CASEOF 4: CALL(DoSomethingElse)
    DEFAULT: QUIT
ENDSWITCH
```

Explanation: If *Test* equals 1, 2, or 3, the corresponding macro plays when the parent macro ends. If *Test* equals 4, the DoSomethingElse subroutine is called and Test4.wcm plays when the parent macro ends. If *Test* does not equal 1, 2, 3, or 4, the macro quits and a CHAIN statement does not execute.

Syntax

CHAIN (*MacroFilename*; {*Parameter*})

Parameters

MacroFilename: *string*
The path and name of a compiled macro.

Parameter: *any* (optional)
Enclose multiple parameters in braces ({}), separated by a semicolon. For example: CHAIN ("macro"; {"a"; "b"; "c"}). Parameter values are passed to a special array variable named MacroArgs.

See Also

RUN

CHARLEN

Purpose

Returns the number of characters in a string, including codes.

The string can be a variable, constant, character string, or result of an expression. See STRLEN for examples.

Syntax

ReturnValue := CHARLEN (*String*)

Returns

The number of characters, including codes, in a string.

Parameters

String: *string*

A variable, constant, character string, or result of an expression.

See Also

CHARPOS
STRLEN
SUBCHAR

CHARPOS

F

Purpose
Determines whether a character string is also a substring.

Syntax
ReturnValue := CHARPOS (*String; SubString*)

Returns
The beginning position of a substring, or zero if a substring is not found.

```
vPos := CHARPOS("WordPerfect"; "Perfect") // vPos = 5
vPos := CHARPOS("WordPerfect"; "Scott") // vPos = 0
```

Parameters
String: *string*
A character string to evaluate.

Substring: *string*
A substring to locate in *String*.

See Also
CHARLEN
STRPOS
SUBCHAR

CLOSEFILE

Purpose
Closes one file or all files.


Close file(s) when no longer needed, or system resources will be lost. Files are closed automatically when macro ends. See OPENFILE.

Syntax
boolean CLOSEFILE (FileID: *numeric*)

Returns
boolean True if successful, False if not.

Parameters
FileID: *numeric* (optional)
If not specified, all open files are closed.

See Also
FILEISEOF
FILEERROR
FILEPOSITION
FILEREAD
FILESIZE
FILETRUNCATE
FILEWRITE

COACHANIMATE

Purpose

Specifies mouse actions or key strokes to execute without user intervention. This allows a Coach macro to demonstrate how to perform a function. You cannot use COACHANIMATE when a filter is enabled.

Syntax

COACHANIMATE (*Type; Specification*)

Parameters

Type

Specifies whether COACHANIMATE animates the mouse or the keyboard.

Mouse!	Mouse movements are animated.
Keyboard!	Keystrokes are animated.

Specification

Wpstring: specifies actions to animate. It contains a keystring or a mouse string, depending on the value of *Type*. See KEYSTRING and MOUSE STRING.

COACHFILTERADD

Purpose

Defines a filter. COACHFILTERENABLE enables defined filters.

Filters monitor: 1) mouse clicks in a specified area of the screen, 2) a single keystroke, 3) a combination of keystrokes (keystring), and 4) commands (tokens).

Compound Filters

To monitor more than one type of event, create a compound filter by using COACHFILTERADD several times, each time with the same filter name but with different filter specifications. A compound filter can monitor single keystrokes and mouse events, but cannot also monitor keystrings or commands. A compound filter can monitor several different commands, but cannot also monitor mouse events or keystrokes. A filter that monitors keystrings cannot be a compound filter.

For example, to create a compound filter that waits for either 1) a left mouse click in the OK button of a specified dialog, or 2) an Enter keystroke, use COACHFILTERADD twice with the same name each time. Specify a mouse string for the first command and a keystroke for the second.

You can enable one interface filter and one command filter at the same time (See COACHFILTERENABLE).

Filter Callback Procedure

Each filter can call at least one associated callback procedure which may give additional help when the user makes a mistake. A callback procedure begins with a label name and ends with RETURN.

After an enabled filter compares the user's actions with those defined in the filter, it determines whether to execute the filter callback procedure. *LabelCondition* in COACHFILTERADD determines when to call the callback: 1) the user's actions match the filter definition, 2) the user's actions do not match the filter definition, and 3) all cases.

When the macro executes a callback, return and array variables are created by the macro system and are accessible to the callback.

Filter Array Variable

Contains information about the user's actions while a filter is enabled. The callback can access the array values for information about the event. The name of the array is formed from the label name. For example, if the label is LABEL1@, the array elements are LABEL1[1], LABEL1[2] and so on.

The information in the array depends on the type of callback receiving the array. There are three callback types: 1) mouse, 2) key (key and keystring), and 3) command.

Mouse Filter Callbacks

The array element values are:

[1] CBTYPE_MOUSE (0). Mouse callback type.
[2] Result of comparison between user's action and defined action. The possible values are:

-1 Error. The macro system did not complete the array.
0 Match. The user's actions exactly match the defined actions.
1 No match. The user's actions do not match the defined actions.
2 Partial Match. The user's actions partially match the defined actions. Possible only for a keyboard callback.
3 Regression. The event undoes previous events (for example, the Backspace key is pressed). A regression is not a match. A callback receives this value only if LabelType is All! or NonMatch!.
4 Quit Filter. The callback is not called.

[3] Handle (hWnd) of the window in which the event occurs.
[4] Horizontal coordinate of the mouse position.
[5] Vertical coordinate of the mouse position.
[6] Windows flags for the message. Specific to the event message. (See the *Microsoft Windows Software Development Kit*.)
[7] Windows event message, such as WM_LBUTTONDOWN or WM_LBUTTONUP.

Key Filter Callbacks

The array element values are:

[1] CBTYPE_KEY (1). Key type.
[2] Filter Compare result (see Mouse Filter Callback above).
[3] Key code.
[4] Windows flags for the event message. Specific to the event message. (See the *Microsoft Windows Software Development Kit*.)
[5] Windows event message, such as WM_KEYDOWN or WM_KEYUP.

Command Callbacks

The array element values are:

[1] CBTYPE_TOKEN (2). Command type.
[2] Filter compare result (see Mouse Filter Callback above).
[3] Application that sent the command.
[4] Command MacroID. If a macro sent the command, this contains the macro ID.
[5] Unique command ID.
[6] Number of parameters for this command.
[7] Command Flags. MAC_REQUIRED, MAC_REPEATING, and/or MAC_NODATA (see Command Flags below).
[8] Value of first parameter.
[9] Value of second parameter (command flags):

MAC_REQUIRED	0x0100	Required parameter.
MAC_REPEATING	0x0400	Parameter is part of a repeating group.
MAC_NODATA	0x8000	No parameters.

Filter Return Variable

Specifies whether to ignore an event or send it to the application. Return variables have the same name as the callback label.

When a macro executes a callback's RETURN, the macro system checks the return variable, then passes the event to the application or throws it away. If the variable is

undefined the macro system performs the default action.

Callback Type	Default Action
Mouse	Throw away the event.
Key	Throw away the event.
Command	Pass the event to the application.

If the return variable is defined, the callback overrides the default action:

Value	Action
0	Pass the event to the application.
1	Throw away the event.

Compound Filters and Callbacks

A compound filter may have several associated callback procedures, such as a different callback procedure for each filter specification (for example, one callback to handle mouse clicks and a second callback to handle key presses). An enabled compound filter checks its specifications in the order defined in the macro. After the filter determines that a callback procedure should be called, the remaining filter specifications are not checked.

Performance

While the macro system processes a filter callback, all mouse movements, mouse clicks, and keystrokes are disabled until the callback returns.

Syntax

COACHFILTERADD (Name; Label; LabelCondition; Type; Filter; AppName; Command; RepeatingGroup)

Parameters

Name: *Wpstring*
The name of the filter to define. If a filter with this name already exists, the new filter definition is appended to the existing definition to create a compound filter (see Compound Filters above).

Mouse! and Key! filters can be combined. KeyString! filters cannot be added to other filters. Command! filters can be appended to other Command! filters.

Label: *Wpstring*
The macro label which is the name of a callback. Macro execution moves to this label as determined by LabelCondition (see Filter Callback Procedure above).

LabelCondition: *enum*
Specifies when to call the callback procedure marked by the *label* label.

All!	All user actions call the callback.
Match!	All user actions that match the filter call the callback.
Nonmatch!	Only user actions that do not match the filter call the callback.

Type: *enum*
Specifies the type of user action to filter (mouse movements, keystrokes, or generated product commands) and what values to assign to some of the remaining parameters.

Mouse!	Filter mouse actions.
Key!	Filter single keystrokes.

KeyString!	Filter a series of keystrokes (keystring).
Command!	Filter a product command.

Filter: *WPstring* (optional)
Specifies the actions to filter. For example, a mouse filter can specify that the user should click in the OK button of a particular dialog. A keystring filter can specify that the user should press Ctrl+B.

If *Type* is Mouse!, this parameter is a mouse string (see MOUSE STRING below). If *Type* is Key!, this parameter is a single keystroke specification (see KEYSTRING). If *Type* is KeyString!, this parameter is a keystring (see KEYSTRING). If *Type* is Command!, this parameter is ignored.

AppName: *Ansistring* (optional)
Non-translatable application name, such as "WordPerfect".

Command: *Unsigned word* (optional)
Name of a product command to filter. Required if *Type* is Command!.

RepeatingGroup (optional)
Filters parameter information for the product command specified in *Command*. Required only if *Type* is Command!.

If any parameter in a repeating group is assigned a value, all parameters of the repeating group must be assigned a value. You can specify more than one repeating group in the same COACHFILTERADD command.

Parameter: *Unsigned word* (First parameter of repeating group)
The parameter number to filter. The Value command evaluates a parameter to a number. For example, the Filename parameter of FileOpenDlg is "Value(FileOpenDlg.Filename)".

Compare: *enum* (Second parameter of repeating group)
Specifies how to compare *Parameter* and *Value*.

Equal!	*Parameter* is equal to *Value*.
NotEqual!	*Parameter* is not equal to *Value*.
Less!	*Parameter* is less than *Value*.
Greater!	*Parameter* is greater than *Value*.
NotGreater!	*Parameter* is less than or equal to *Value*.
Exist!	*Parameter* exists but can be any value.
NotExist!	*Parameter* does not exist.
DontCare!	*Parameter* may or may not exist. If it exists, it can be any value.

Value: *any* (Third parameter of repeating group)
Contains any value type to compare with *Parameter*.

See Also
COACHFILTERENABLE

COACHFILTERDESTROY

Purpose

Removes a filter definition.

Only one interface filter and one command filter can be enabled at a time.

When a filter is enabled, it compares a mouse movement, keystroke, or token to actions defined in the filter, then calls a callback procedure or passes the event unchanged.

When several filters are enabled, the filters are chained together. When an event occurs, the first filter filters the event. If that filter does not call a callback, the second filter filters the event, and so on. As soon as a filter calls a callback, no other filters can filter the event.

A filter must be defined by COACHFILTERADD before it can be enabled. An enabled filter remains enabled until it is disabled by COACHFILTERDISABLE or destroyed by COACHFILTERDESTROY.

Syntax

COACHFILTERDESTROY (*Name*)

Parameters

Name: *Wpstring*
The name of a filter to destroy.

See Also

COACHFILTERDISABLE

COACHFILTERDISABLE

Purpose

Disables an active filter. A filter definition remains in memory until COACHFILTERDESTROY destroys it, or until the macro ends.

Syntax

COACHFILTERDISABLE (*Command; Interface*)

Parameters

Command: *WPstring* (optional)
The name of a command filter to disable.

Interface: *WPstring* (optional)
The name of an interface filter to disable.

See Also

COACHFILTERDESTROY

COACHFILTERENABLE

Purpose
Enables (activates) specified filters.

Syntax
COACHFILTERENABLE (*Command; Interface*)

Parameters
Command: *WPstring* (optional)
The name of a command filter to enable.

Interface: *WPstring* (optional)
The name of an interface filter to enable. Only one interface filter can be enabled at a time.

See Also
COACHFILTERADD
COACHFILTERDESTROY
COACHFILTERDISABLE

COACHGETREGIONINFO

Purpose

Returns information about a named region, such as a window handle, rectangle, existence of the window, and control ID.

Syntax

COACHGETREGIONINFO ("Region"; Existvar:*exist*; Hwndvar:*hwnd*; Leftvar:*left*; Topvar:*top*; RightVar:*right*; Bottomvar:*bottom*)

Parameters

Region
Name of the region to test. For example:

"WordPerfect.Bookmark.GoToBttn"

Existvar
Returns 0 if the region does not exist, and 1 if it does exist.

Hwndvar
Windows handle value of a specified region.

Leftvar
Screen coordinates for the left edge of a specified region.

Topvar
Screen coordinates for the top edge of a specified region.

Rightvar
Screen coordinates for the right edge of a specified region.

Bottomvar
Screen coordinates for the bottom edge of a specified region.

COACHMESSAGEBOX

Purpose

Displays a message box.

Message boxes display available options, such as a list of Coaches. Message boxes can include title, text, two bitmap images, radio buttons, check boxes, and pushbuttons.

Users can display more than one message box at a time and drag them to different postions on the screen.

Message Box Callback Procedure

Message boxes can have associated callback procedures. The callback procedure responds to events such as radio button, check box, and push button. For example, if the user clicks a message box Close button, the callback procedure calls COACHMESSAGEBOXCLOSE to dismiss the message box.

When macro execution moves to a callback, the macro system creates an array variable that the callback can access. The array name is formed from the label name. For example, the array elements of LABEL1@ are LABEL1[1], LABEL1[2], and so on. The array variable contains information about the message box and identifies the selected control. The element values are:

[1] CBTYPE_MESSAGEBOX (4). Message Box callback.
[2] Name of the message box specified in *MessageBoxName* of COACHMESSAGEBOX.
[3] Name of the control specified in *ControlName* of COACHMESSAGEBOX.
[4] A value that specifies whether a control is checked/pressed (1) or unchecked (0).

Syntax

COACHMESSAGEBOX (MessageBoxName; Title; Text; Library; IdBitmap1; IdBitmap2; Bmp2RedMask; Bmp2GreenMask; Bmp2BlueMask; BorderType; Bitmap1Position; Bitmap2Position; Top; Left; TitleFont; TitleFontSize; TextFont; TextFontSize; Modality; Label; ResultVariable; RepeatingGroup)

Parameters

MessageBoxName: *Ansistring*
Unique name of a message box. Passed to COACHMESSAGEBOXCLOSE to close the message box. COACHMESSAGEBOXCLOSE is required only if the message box modality is Modal! or Modeless! (see Modality below).

The same name can be used for more than one message. All messages with the same name close at the same time.

Title: *WPstring* (optional)
Message box title. The title is displayed at the top of the message box, below the bitmap specified in *Bitmap1*. *TitleFont* and to specify font and font size.

Text: *Wpstring* (optional)
Formatted text:

Character	Result
\n	Insert hard return.
\b	Toggle bold.

\u Toggle underline.
\i Toggle italic.

Bitmap2Position specifies text position. *TextFont* and *TextFontSize* specify font and font size.

Library: *Dword* (optional)
The Windows handle of the DLL that contains the bitmaps specified in *IdBitmap1* and *IdBitmap2*. DLLOAD retrieves the handle. DLLFREE removes the DLL from memory.

IdBitmap1: *Word* (optional)
Specifies a bitmap image, such as the Coach graphic, to display at the top of the message box.

Each bitmap specified in *IdBitmap1* must be followed by a monochrome version of the bitmap (transparency bitmap) in the DLL library. White areas in a transparency bitmap are transparent, black areas are not.

IdBitmap2: *Word* (optional)
Specifies a bitmap image to display in the message box. *Bitmap2Position* specifies bitmap location.

Bitmaps specified by *IdBitmap2* do not require a transparency bitmap. *Bmp2RedMask*, *Bmp2GreenMask*, and *Bmp2BlueMask* specify a transparency color for *IdBitmap2* bitmaps to conserve disk space. Areas filled with the transparency color are transparent.

Bmp2RedMask: *Byte* (optional)
Bmp2RedMask, *Bmp2GreenMask* and *Bmp2BlueMask* set red, green, and blue values (RGB) for the Bitmap2 transparency color.

The transparency color is the fill color that is transparent when the bitmap is displayed.

Bmp2GreenMask: *Byte* (optional)
See Bmp2RedMask above.

Bmp2BlueMask: *Byte* (optional)
See Bmp2RedMask above.

Bitmap1Position: *enum* (optional)
Bitmap1 position in the message box.

Left!	Display bitmap in the top left corner. The top of the message box is flat.
Top!	Display bitmap in a semicircle at the top.

Bitmap2Position: *enum* (optional)
Bitmap2 position in the message box.

Left!	Align the left edge of the bitmap with the left edge of the message box. If the message box is longer than the bitmap, position the bitmap at the top of the message box. Text and controls will wrap to the right.
Top!	Align the top edge of the bitmap directly under the message box title. If the message box is wider than the bitmap, display the bitmap at the left of the message box. Text and controls appear below the bitmap.
Right!	Align the right edge of the bitmap with the right edge of the message box. If the message box is longer than the bitmap, display the bitmap at the top of the message box. Text and controls wrap to the left.

Bottom!	Position the bottom edge of the bitmap above button controls, and below text and all other controls. Pushbuttons always appear at the bottom of the box. If the message box is wider than the bitmap, display the bitmap at the left of the message box.
LeftCenter!	Center bitmap vertically.
TopCenter!	Center bitmap horizontally.
RightCenter!	Center bitmap vertically.
BottomCenter!	Center bitmap horizontally.

Top: *Word* (optional)
Specifies the percentage of the document window's vertical space above the center of the message box.

Left: *Word* (optional)
Specifies the percentage of the document window's horizontal space left of the center of the message box.

TitleFont: *Ansistring* (optional)
The name of the Windows font to use to display the title. If no titlefont is specified, the default dialog font is used.

TitleFontSize: *Word* (optional)
The point size of the message box title.

TextFont: *Ansistring* (optional)
The name of the Windows font to display text. If no font is specified, the default font (MS Sans Serif 8pt) is used.

TextFontSize: *Word* (optional)
The point size of the message box text.

Modality: *enum*
The modality of the message box: 1) whether to dismiss the message box when a control is selected, 2) whether to call a callback procedure while a message box is displayed, and 3) whether the user can perform actions in the document window while the message box is displayed.

Simple!	Dismiss the message box when the user clicks a control. No callback procedure. User cannot perform actions in the document window while the message box is displayed.
Modal!	Do not dismiss the message box until COACHMESSAGEBOXCLOSE is called. The callback procedure specified in *label* is called when the user selects a control. The user cannot perform actions in the document window while the message box is displayed.
Modeless!	Do not dismiss the message box until COACHMESSAGEBOXCLOSE is called. The callback procedure specified in *label* is called when the user selects a control. The user can perform actions in the document window while the message box is displayed.

Label: *Wpstring* (optional)
The label name of a message box callback procedure. Set this parameter only if the message box modality is Modal! or Modeless!. If *Modality* is Simple!, this parameter is ignored.

The message box calls this label when the user selects a control (radio button, check box, or pushbutton) on the message box. Also see Message Box Callback Procedure above.

ResultVariable: *variable* (optional)
Used only if *Modality* is set to Simple!. When a message box's modalityis simple, the message box is automatically dismissed when the user selects a control. The result variable specified in this parameter holds the name of the control (ControlName) that was selected.

RepeatingGroup (optional)
Specifies the controls (such as radio buttons, check boxes, and pushbuttons) to display in the message box. Each repeating group contains three parameters which hold the information for one control. You can specify multiple controls.

Controls always appear beneath the message box text (see Text). Regardless of the order in which the controls are specified, they appear in the message box in this order: radio buttons, check boxes, and pushbuttons. If Bitmap2Position is Bottom!, bitmap2 displays below radio buttons and check boxes, and above pushbuttons (pushbuttons always display at the bottom of the message box).

If one parameter of a repeating group is specified, all three must be specified.

ControlName: *Ansistring* (First Parameter of Repeating Group)
A control name. The name does not appear on the message box. If thiscontrol is selected in the message box, the control name is displayed in Array[3]. If the message box modality is Simple!, the control name is copied to *ResultVariable*.

ControlType: *enum* (Second Parameter of Repeating Group)
Indicates a control type.
RadioButton!
CheckBox!
Button!

ControlText: *Ansistring* (Third Parameter of Repeating Group)
Control text. Text is displayed to the right of radio buttons and checkboxes, and inside push buttons.

See Also
COACHMESSAGEBOXCLOSE

COACHMESSAGEBOXCLOSE

Purpose

Closes a message box. Required only for Modeless! or Modal! message boxes (see Modality in COACHMESSAGEBOX).

Syntax

COACHMESSAGEBOXCLOSE (*MessageBoxName*)

Parameters

MessageBoxName: *Ansistring*
Name of a message box to close. This should be the same name as in *MessageBoxName* of COACHMESSAGEBOX.

See Also

COACHMESSAGEBOX

COACHPROMPT

Purpose

Displays a prompt.

The prompt can display a message and a bitmap image that points to an item, such as a prompt that points to the File menu with the text "Click the File menu item."

The prompt location is set relative to a region (see Location and Region), or set to a fixed position in the document window.

When you set a location relative to a region, you can have a prompt display to the right, left, upper right, upper left, lower right, or lower left of the region. The system adjusts the size of the prompt box to fit in the display area. If the prompt is too large, the system checks for a suitable location by rotating the prompt clockwise. For example, if the prompt is to display to the right of a region, but there isn't room, the system checks to the lower right. If it will not fit there, the system checks the lower left, and so on.

COACHPROMPT points to items with bitmap images. The bitmap image is displayed between the prompt box and the item.

IdBitmap specifies the bitmap image ID. For example, if the prompt is displayed at the upper left, and the bitmap is a pointing hand, the fingers point down and to the right.

Bitmaps for all 6 prompt locations must be defined in order, starting with the upper left bitmap and proceding clockwise. The transparency bitmap must immediately follow each bitmap. *IdBitmap* should always contain the first bitmap ID (upper left).

Syntax

COACHPROMPT (*PromptName; Message; Font; FontSize; Library; IdBitmap; Location; Top; Left; Region*)

Parameters

PromptName: *Ansistring*
A unique name for the prompt.

Message: *Wpstring* (optional)
Message to display in the prompt. Formatting codes are optional:

Character	Result
\n	Insert Hard Return
\b	Toggle Bold
\u	Toggle Underline
\i	Toggle Italic

Font: *Ansistring* (optional)
Name of the Windows font for prompt text. If no font is specified, the default dialog font is used.

FontSize: *Word* (optional)
Point size of the prompt text. If no font is specified, the default font (MS Sans Serif 8pt) is used.

Library: *Dword* (optional)
The Windows handle of the DLL that contains the bitmap to display outside the prompt. DLLOAD retrieves the handle. DLLFREE removes the DLL from memory.

IdBitmap: *Word* (optional)
The identifier of the first bitmap image defined for COACHPROMPT.

In the bitmap DLL library each bitmap specified by *IdBitmap* must be immediately followed by a transparency bitmap. A transparency bitmap is a monochrome version of a bitmap. White areas in the transparency bitmap are transparent, black areas are not.

Location: *enum* (optional)
Specifies where to display the prompt in relation to the region (see Region).

UpperLeft!	Display prompt upper left.
UpperRight!	Display prompt upper right.
Right!	Display prompt right.
LowerRight!	Display prompt lower right.
LowerLeft!	Display prompt lower left.
Left!	Display prompt left.
Specify!	Display prompt as specified by *Top* and *Left*.

Top: *Word* (optional)
Specifies the percentage of the document window's vertical space above the center of the message box. Required only if *Location* is Specify!.

Left: *Word* (optional)
Specifies the percentage of the document window's horizontal space left of the center of the message box. Required only if *Location* is Specify!.

Region: *WPstring* (optional)
The named region the prompt points to. Not required if *Top* and *Left* are specified.

See Also
COACHPROMPTCLOSE
DLLLOAD
DLLFREE

COACHPROMPTCLOSE

Purpose
Closes a prompt.

Syntax
COACHPROMPTCLOSE (*PromptName*)

Parameters
PromptName: *Ansistring*
Name of the prompt to close. This is the same name specified in *PromptName* of COACHPROMPT.

See Also
COACHPROMPT

COACHREMOVEDIALOGFILTER

Purpose
Removes a dialog box filter from memory. (Not available in Shared Code 2.0.)

Syntax
COACHREMOVEDIALOGFILTER (*Dialog Name*)

Example: COACHREMOVEDIALOGFILTER ("WordPerfect.Tabs")

Parameters
Dialog Name
Specifies a dialog filter to remove.

See Also
COACHSETDIALOGFILTER

COACHSETDIALOGFILTER

Purpose

Defines a filter for a dialog box. The filter definition remains in memory until removed by CoachRemoveDialogFilter or until the macro ends. When the dialog box is initialized, macro execution moves to the dialog filter callback. (Not available in Shared Code 2.0.)

Syntax

COACHSETDIALOGFILTER (*DialogRegionName; Label; DefaultAction*)

Example: COACHSETDIALOGFILTER ("WordPerfect.Tabs"; Tabsetcb@)

Parameters

DialogRegionName
Specifies a dialog to filter.

Label
Specifies the name of a callback label to go to when the dialog box is displayed.

DefaultAction (optional)
Specifies whether the default allows an event or throws it away. If a default action is not specified, all events are allowed.

Allow!	Allow the event.
ThrowAway!	Do not allow the event.

Returns

When the dialog is invoked or a control on the dialog is activated, a variable array is created indicating the actions the user is taking. The dialog callback procedure can test each item in the array and perform appropriate operations.

The variable array's name is the same as the callback label. For example, the variable array of TabSetCB@ is TabSetCB[array item].

The variable array contains the following information depending on the message sent in Callback[5].

Element	Description
Callback[1]	Callback type (always 3 for a dialog callback)
Callback[2]	Dialog name
Callback[4]	Dialog window handle
Callback[5]	Message sent to the callback

The following messages are valid and require the additional information shown.

272 = WM_INITDIALOG. Sent when the dialog is first displayed. This message is for notification only and cannot be altered or prevented by the callback. If Callback[5] is 272, the following information is contained in the callback array:

Callback[6] = Window handle of the control that will receive the focus.

2 = WM_DESTROY. Sent when the dialog goes down. This message is for notification only and cannot be altered or prevented by the callback. No additional information is

contained in the callback array.

6 = WM_ACTIVATE. Sent when the dialog is activated or inactivated. This message is for notification only and cannot be altered or prevented by the callback. If Callback[5] is 6, the following information is contained in the callback array:

Callback[6] = Activation state of the dialog
0 = WA_INACTIVE (dialog becomes inactive)
1 = WA_ACTIVE (dialog becomes active)
2 = WA_CLICKACTIVE (dialog becomes active from a mouse click in the window)

Callback[8] = Minimized flag (A nonzero value means that the dialog is minimized.)

Callback[9] = Window handle of the window being activated or deactivated

3 = WM_MOVE. Sent when the dialog is moved to a new location on screen. This message is for notification only and cannot be altered or prevented by the callback. If Callback[5] is 3, the following information is contained in the callback array:

Callback[8] = New X position of the window

Callback[9] = New Y position of the window

273 = WM_COMMAND. Sent when the user interacts with a control or menu item owned by the dialog. WM_COMMAND events can be altered by the callback. To prevent the dialog from acting on the control or menu item, set the return variable to 1. To enable the dialog to act on the menu item, set the return variable to 0. The default is 0. The information in the callback array depends on the menu or control type. If Callback[5] is 273, the following information is contained in the array:

Callback[3] = Control name (blank if the notification is from a menu item).

Callback[6] = Control or menu item ID.

Callback[9] = Control window handle (blank if the notification is from a menu item).

Menu Items
No additional information in the array.

ListBoxes
Callback[8] = LBN_DBLCLK (2). List box item was double-clicked.

ComboBoxes
Callback[8] = CBN_DBLCLK (2). Combo box item was double-clicked.

Button
Callback[8] = BN_CLICKED (0). Button was clicked
Callback[8] = BN_DOUBLECLICKED (5). Button was double-clicked.

FileNameEntry
Callback[8] = FNEN_DLGDISMISS (0). Open dialog was dismissed.
Callback[8] = FNEN_DLGRAISE (1). Open dialog was displayed.

HotSpot

Callback[8] = BMPN_CLICKED (0). Bitmap was clicked.

Callback[8] = BMPN_DOUBLECLICKED (1). Bitmap was double-clicked.

See Also

COACHREMOVEDIALOGFILTER

COPYFILE

F

Purpose
Copies a file.

Syntax
boolean COPYFILE (SourceFilename: *string*; DestinationFilename: *string*; Prompts: *enum*)

Return Value
boolean True if successful, False if not.

Parameters
SourceFilename: *string*

DestinationFilename: *string*

Prompts: *enum* (optional)
Confirm replacement if destination file exists.
NoPrompts!
Prompts!

See Also
DELETEFILE
DOESFILEEXIST
RENAMEFILE

CREATEDIRECTORY

Purpose
Creates a directory.

Syntax
boolean CREATEDIRECTORY (DirectoryName: *string*; Prompts: *enum*)

Return Value
boolean True if successful, False if not.

Parameters
DirectoryName: *string*
Include full path.

Prompts: *enum* (optional)
Prompt if directory exists.
NoPrompts!
Prompts!

See Also
DELETEDIRECTORY
DOESDIRECTORYEXIST
GETCURRENTDIRECTORY
RENAMEDIRECTORY
SETCURRENTDIRECTORY

CTON

Purpose
Converts a keyboard character to its decimal equivalent.

Syntax
ReturnValue := CTON (*Character*)

Returns
The decimal equivalent of a keyboard character.

Parameters
Character: *string*
A keyboard character to convert. If a string is used, the decimal equivalent of the first character is returned.

See Also
NTOC

DDEEXECUTE

Purpose

Sends a command string to a client application.

See the client application's documentation for command names and syntax.

Syntax

DDEEXECUTE (*ConversationID*; *Command*)

Parameters

ConversationID: *numeric*
A number that links the server and client applications. DDEINITIATE returns the number in a variable.

Command: *string*
A command to execute.

See Also

DDEINITIATE
DDEEXECUTEEXT

DDEEXECUTEEXT

F

Purpose

Sends a command to a client application, and returns a value that indicates if the command was successfully processed.

See the client application's documentation for command names and syntax.

Syntax

ReturnValue := DDEEXECUTEEXT (*ConversationID*; *Command; TimeOut; Label*)

Returns

A number greater than 0 if successful, or 0 if not.

Parameters

ConversationID: *numeric*
A number that links the server and client applications. DDEINITIATE returns the number in a variable.

Command: *string*
A command to execute.

TimeOut: *numeric*
Asychronous DDE: Set *TimeOut* to 0 or a negative value. DDE returns immediately, and notifies the macro that the command was delivered. When the client application acknowledges receiving the command, DDE notifies the macro and the macro calls *label*. The return value is invalid for asychronous DDE, because the client application has not yet processed the command.

Synchronous DDE: Set *TimeOut* to a positive integer, which specifies the maximum number of milliseconds to wait while the client application processes the command. The result is returned when the command is processed or DDE times out (command is not processed within the specified time). See Return above.

Label: *label*
The LABEL to call when the client application acknowledges receiving the command (see TimeOut parameter). If the LABEL statement does not include RETURN, macro behavior is unpredictable.

See Also

DDEEXECUTE
DDEINITIATE
LABEL

DDEINITIATE

F

Purpose

Starts a DDE conversation with a client application.

A conversation must be started between server and client applications before you can use commands such as DDEPOKE, DDEREQUEST, and DDEEXECUTE.

Syntax

DDEINITIATE (*ConversationID; ServiceName*; *TopicName*)

Parameters

ConversationID: *variable*
A number that links the server and client applications is returned in this variable, or zero if the conversation is not started.

ServiceName: *string*
A DDE service name. See a client application's documentation. For GroupWise 4.1, ServiceName is "GroupWise".

TopicName: *string*
A Topic name. For example, GroupWise 4.1 for Windows is COMMAND. WordPerfect Office 4.0a for Windows has three topics names, depending on the type of action to initiate: COMMAND, GETOFFICEDATA and GETADDRESSBOOKDATA. See the client application's documentation.

See Also

DDEEXECUTE
DDEPOKE
DDEREQUEST
DDETERMINATE
DDETERMINATEALL

DDEPOKE

F

Purpose
Assigns data to an item in a client application.

Syntax
DDEPOKE (*ConversationID; ItemName; ItemData*)

Parameters
ConversationID: *numeric*
A number that links the server and client applications. DDEINITIATE returns the number in a variable.

ItemName: *string*
A client application item. See the client application's documentation for item names.

ItemData: *string*
The data to assign to a client application item.

See Also
DDEINITIATE
DDEREQUEST

DDEREQUEST

Purpose
Assigns the contents of a client application item to a macro variable.

Syntax
ReturnValue := DDEREQUEST (*ConversationID; ItemName*)

Returns
The contents of the client application item specified by *ItemName*.

Parameters
ConversationID: *numeric*
A number that links the server and client applications. DDEINITIATE returns the number in a variable.

ItemName: *string*
The name of a client application item.

See Also
DDEINITIATE
DDEPOKE
ONDDEADVISE CALL

DDETERMINATE

F

Purpose
Ends a DDE conversation with a client application.

Syntax
DDETERMINATE (*ConversationID*)

Parameters
ConversationID: *numeric*
A number that links the server and client applications. DDEINITIATE returns the number in a variable.

See Also
DDEINITIATE
DDETERMINATEALL

DDETERMINATEALL

Purpose
Ends all DDE conversations.

Syntax
DDETERMINATEALL

See Also
DDEINITIATE
DDETERMINATE

DECLARE

F

Purpose

Creates a local variable or array, which stores as many variables, constants, or expressions as memory allows.

An array can have up to 10 dimensions, and up to 32,767 elements per dimension depending on the amount of available memory. In the following example, DECLARE creates ArrayA with ten elements, ArrayB with five elements, and ArrayC with five elements. An element in each array is assigned a value. Parentheses are optional.

DECLARE(ArrayA[10]; ArrayB[5]; ArrayC[5])
ArrayA[3] := "John"
Explanation: Assigns "John" to the third element of ArrayA

ArrayB[5] := ArrayA[3]
Explanation: Assigns "John" to the fifth element of ArrayB.

ArrayC[] := {1; 2; 3; 4; 5}

or...

ASSIGN(ArrayC[]; {1; 3; 5; 7; 9})

Explanation: Assigns 1, 3, 5, 7, and 9 to the first, second, third, fourth, and fifth elements of ArrayC.

In the next example, DECLARE creates a two-dimensional array (ArrayC) with 100 elements (ten rows and ten columns). One element is assigned a value.

DECLARE ArrayC[10;10]
ArrayC[2;3] := 3+6
Explanation: Assigns 9 to the element at row 2, column 3.

Syntax

DECLARE (*VariableName[Elements]; ...*)

Parameters

VariableName: *variable*
The name of a local variable or array. Use a semicolon to separate names when declaring more than one variable or array.

Elements: *numeric*
The number of elements in an array, enclosed in brackets. Use a semicolon to separate elements in a multi-dimensional array.

See Also

GLOBAL
LOCAL
PERSIST
PERSISTALL

DEFAULTUNITS

F

Purpose

DEFAULTUNITS specifies the default unit of measure.

In the first example, Advance uses the default unit of measure (WP Units). In the second, Advance overrides the default unit of measure and uses inches. In the third, DEFAULTUNITS specifies centimeters as the default unit of measure.

Advance(AdvanceFromTop!; 4)
Explanation: Advance insertion point 4 WP Units.

Advance(AdvanceFromTop!; 4")
Explanation: Advance insertion point 4 inches.

DEFAULTUNITS(Centimeters!)
Advance(AdvanceFromTop!; 4)
Explanation: Advance insertion point 4 centimeters.

The default unit of measure is effective until changed by another DEFAULTUNITS command. If not specified by DEFAULTUNITS, the default unit of measure is WP Units.

Syntax

DEFAULTUNITS (*Units*)

Parameters

Units: *enum*
A unit of measure.

Inches!	
Centimeters!	
Millimeters!	
Points!	(One Point equals 1/72 of an inch)
WPUnits!	(One WordPerfect unit equals 1/1200 of an inch)
WP1200ths!	(Synonym for WPUnits!)

DELETEDIRECTORY

Purpose
Deletes a directory.

Syntax
boolean DELETEDIRECTORY (DirectoryName: *string*; Prompts: *enum*)

Return Value
boolean True if successful, False if not.

Parameters
DirectoryName: *string*
Include full path.

Prompts: *enum* (optional)
Prompt if directory does not exist.
NoPrompts!
Prompts!

See Also
CREATEDIRECTORY
DOESDIRECTORYEXIST
GETCURRENTDIRECTORY
RENAMEDIRECTORY
SETCURRENTDIRECTORY

DELETEFILE

Purpose
Deletes a file(s).

Syntax
boolean DELETEFILE (Filename: *string*; Prompts: *enum*)

Return Value
boolean True if successful, False if not.

Parameters
Filename: *string*
Include full path. If file not specified, deletes all files in directory!

Prompts: *enum* (optional)
Prompt if file does not exist.
NoPrompts!
Prompts!

See Also
COPYFILE
DOESFILEEXIST
RENAMEFILE

DIALOGADDCHECKBOX

F

Purpose

Adds a check box control to a dialog box.

Syntax

DIALOGADDCHECKBOX (*Dialog; Control; Left; Top; Width; Height; Text; MacroVar*)

Parameters

Dialog: *string*
The name or number of the dialog box to contain the check box control (see DIALOGDEFINE).

Control: *string*
A name or number that identifes the check box control (see DIALOGSHOW).

Left: *numeric*
Number of dialog units from the left side of the dialog box to the left side of the check box.

Top: *numeric*
Number of dialog units from the top of the dialog box to the top of the check box.

Width: *numeric*
Width of the check box control in dialog units.

Height: *numeric*
Height of the check box control in dialog units.

Text: *string*
Text displayed to the right of the check box.

MacroVar: *variable*
1 is returned in this variable when check box is checked, and 0 when not checked. The default value is 0. If *MacroVar* is assigned a value greater than 0 before the macro compiles, the check box is checked when the dialog box is displayed.

See Also

DIALOGDEFINE
DIALOGSHOW

DIALOGADDCOLORWHEEL

F

Purpose

Adds a color wheel control to a dialog box, and returns a number that represents the selected color.

The color selection consists of three color elements (red, green, and blue), separated into color values by the formula given below (see MacroVar parameter). These values are used in Windows programming to specify the color of a pen or brush.

Syntax

DIALOGADDCOLORWHEEL (*Dialog; Control; Left; Top; Width; Height; MacroVar*)

Parameters

Dialog: *string*
The name or number of the dialog box to contain the color wheel control (see DIALOGDEFINE).

Control: *string*
A name or number that identifes the color wheel control (see DIALOGSHOW).

Left: *numeric*
Number of dialog units from the left side of the dialog box to the left side of the color wheel control.

Top: *numeric*
Number of dialog units from the top of the dialog box to the top of the color wheel control.

Width: *numeric*
Width of the color wheel control in dialog units.

Height: *numeric*
Height of the color wheel control in dialog units.

MacroVar: *variable*
A number corresponding to a color selection is returned in this variable. This number can be divided into three other numbers which correspond to red, green, and blue. The formulas are:

vRed := *MacroVar* % 256
(vRed contains the red component, which is the remainder of *MacroVar* divided by 256)

vGreen := *MacroVar* / 256 % 256
(vGreen contains the green component, which is the remainder of *MacroVar* divided by 256 divided by 256)

vBlue := *MacroVar* / 256 / 256
(vBlue contains the blue component, which is the result of *MacroVar* divided by 256 divided by 256)

See Also

DIALOGDEFINE

DIALOGSHOW

DIALOGADDCOMBOBOX

F

Purpose

Adds a combo box control to a dialog box.

Displays an edit box and a list box. You enter text in the edit box, or double-click a list item to insert. Use DIALOGADDLISTITEM to create list box items.

Syntax

DIALOGADDCOMBOBOX (*Dialog; Control; Left; Top; Width; Height; Style; MacroVar; LimitText*)

Parameters

Dialog: *string*
The name or number of the dialog box to contain the combo box control (see DIALOGDEFINE).

Control: *string*
A name or number that identifes the combo box control (see DIALOGSHOW).

Left: *numeric*
Number of dialog units from the left side of the dialog box to the left side of the combo box.

Top: *numeric*
Number of dialog units from the top of the dialog box to the top of the combo box.

Width: *numeric*
Width of the combo box control in dialog units.

Height: *numeric*
Height of the combo box control in dialog units.

Style: *enum*
Combo box styles. Type | between enumerations to combine styles.

Simple!	Displays an edit control and an opened drop-down list box.
Sort!	Items are sorted alphabetically. If Sort! is used by itself, Simple! is the default combo box style. See DIALOGADDLISTITEM.
Dropdown!	Displays an edit control. Clicking the arrow to the right of the edit control displays a drop-down list box.
Droplist!	Displays a read-only edit control. Clicking the arrow to the right of the edit control displays a drop-down list box.
WPChars!	Non-keyboard characters allowed in a list item. See DIALOGADDLISTITEM.

Style combination examples

Sort! | Simple!

Explanation: Name search allowed on a sorted list box. A vertical scrollbar automatically appears when needed.

MacroVar: *variable*
The selected combo box item is returned in this variable. An item can be assigned to *MacroVar* before the macro compiles.

LimitText: *numeric*
Maximum number of characters the edit control accepts, up to the number of characters that fit in the edit box. This number has no effect if you choose style 16 (DropDown3D!).

See Also

DIALOGDEFINE
DIALOGSHOW
DIALOGADDLISTBOX
DIALOGADDLISTITEM
DIALOGADDPOPUPBUTTON

DIALOGADDCONTROL

F

Purpose

Adds a custom control to a dialog box.

Syntax

DIALOGADDCONTROL (*Dialog; Control; Left; Top; Width; Height; Class; Style; WindowName; MacroVar; Instance*)

Parameters

Dialog: *string*
The name or number of the dialog box to contain the custom control (see DIALOGDEFINE).

Control: *string*
A name or number that identifes the custom control (see DIALOGSHOW).

Left: *numeric*
Number of dialog units from the left side of the dialog box to the left side of the custom control.

Top: *numeric*
Number of dialog units from the top of the dialog box to the top of the custom control.

Width: *numeric*
Width of the custom control in dialog units.

Height: *numeric*
Height of the custom control in dialog units.

Class: *string*
The name of the control class. Standard Windows classes include Button, ComboBox, Edit, ListBox, ScrollBar, and Static. Other controls such as Meters and Spin buttons must be defined in a DLL (see Instance parameter). Class names for WordPerfect custom controls are in the TKSHWIN.H file on the diskette that comes with *GroupWise Software Developer's Kit*.

Style: *numeric*
A number that defines a class style. For exampe, an edit control includes styles such as left, multiline, and password. The values for standard Windows styles are in the WINDOWS.H file.

WindowName: *string*
Control text. For example, the text that appears on a button control. Set this value to "" if the control does not accept text.

MacroVar: *variable*
The value returned by a control. For example, a check box returns 1 if checked, and 0 if not checked.

Instance: *numeric*
The handle of the DLL that contains the custom control. DLLLOAD returns the handle of a custom DLL in a variable. The Shared Code default is 0 (see *GroupWise Software*

Developer's Kit for information about custom controls).

See Also

DIALOGDEFINE
DIALOGSHOW
MacroDialogResult

DIALOGADDCOUNTER

F

Purpose

Adds a counter control to a dialog box.

Click the counter button to insert a number in the edit box that is within a specified range (see CountMin and CountMax parameters).

Syntax

DIALOGADDCOUNTER (*Dialog; Control; Left; Top; Width; Height; Format; MacroVar; CountMin; CountMax; CountStep*)

Parameters

Dialog: *string*
The name or number of the dialog box to contain the counter control (see DIALOGDEFINE).

Control: *string*
A name or number that identifes the counter control (see DIALOGSHOW).

Left: *numeric*
Number of dialog units from the left side of the dialog box to the left side of the counter control.

Top: *numeric*
Number of dialog units from the top of the dialog box to the top of the counter control.

Width: *numeric*
Width of the counter control in dialog units.

Height: *numeric*
Height of the counter control in dialog units.

Format: *numeric*
Reserved for future use. Set parameter to 0.

MacroVar: *variable*
The selected counter number is returned in this variable. A number can be assigned to *MacroVar* before the macro compiles.

CountMin: *numeric*
The minimum number that you can insert by clicking the counter button. You can enter a number in the edit control beyond the minimum and maximum range.

CountMax: *numeric*
The maximum number that you can insert by clicking the counter button. You can enter a number in the edit control beyond the minimum and maximum range.

CountStep: *numeric*
The increment and decrement step value.

See Also

DEFAULTUNITS

DIALOGDEFINE

DIALOGADDEDITBOX

F

Purpose

Adds an edit control to a dialog box.

The text entered in the edit box is returned in MacroVar (see MacroVar parameter).

Syntax

DIALOGADDEDITBOX (*Dialog; Control; Left; Top; Width; Height; Style; MacroVar; LimitText*)

Parameters

Dialog: *string*
The name or number of the dialog box to contain the edit control (see DIALOGDEFINE).

Control: *string*
A name or number that identifes the edit control (see DIALOGSHOW).

Left: *numeric*
Number of dialog units from the left side of the dialog box to the left side of the edit control.

Top: *numeric*
Number of dialog units from the top of the dialog box to the top of the edit control.

Width: *numeric*
Width of the edit control in dialog units.

Height: *numeric*
Height of the edit control in dialog units.

Style: *enum*
Edit control styles. Type | between enumerations to combine styles. Not all combinations are possible.

Left!	Text is left justified.
Right!	Text is right justified if Multiline! style is set and WPChars! style is not set.
Center!	Text is centered if Multiline! style is set and WPChars! style is not set.
VScroll!	Edit control with a vertical scroll bar.
HScroll!	Edit control with a horizontal scroll bar.
WPChars!	Can display WP Character box (press Ctrl+W). Non-keyboard characters allowed.
Multiline!	Multiple lines of input automatically wrap to the next line. Press Enter to cause the default button action (see DIALOGADDPUSHBUTTON). If Enter2Hrtn! style is set in DIALOGDEFINE, press enter to move the insertion point to the next line.
Uppercase!	Text automatically converted to uppercase.
Lowercase!	Text automatically converted to lowercase.
Password!	Text automatically displayed with astersik (*) characters. The actual characters are stored by the edit control.
WordWrap!	Text wraps if Multiline! style is set.
SoftReturn!	Text contains Soft Return codes [SRt].
Attributes!	Underline (press Ctrl+U) and italics (press Ctrl+I) allowed if WPChars! style is set.

NoTabs! Tab codes not exported with output.
NoWPChar! Cannot display WP Character dialog box. Non-keyboard characters not allowed.

MacroVar: *variable*
The text entered in the edit box is returned in this variable. Text can be assigned to *MacroVar* before the macro compiles.

LimitText: *numeric*
The maximum number of characters the edit control accepts.

See Also
DIALOGDEFINE
DIALOGSHOW

DIALOGADDFILENAMEBOX

F

Purpose

Adds a filename edit control to a dialog box.

The filename edit control consists of an edit control and a button control. Clicking the button displays the Select Directory or Select File dialog box depending on the control style (see Style parameter).

Syntax

DIALOGADDFILENAMEBOX (*Dialog; Control; Left; Top; Width; Height; Style; MacroVar; DefaultDir; Template*)

Parameters

Dialog: *string*
The name or number of the dialog box to contain the filename edit control (see DIALOGDEFINE).

Control: *string*
A unique expression) or number that identifes the filename edit control (see DIALOGSHOW).

Left: *numeric*
Number of dialog units from the left side of the dialog box to the left side of the filename edit control.

Top: *numeric*
Number of dialog units from the top of the dialog box to the top of the filename edit control.

Width: *numeric*
Width of the filename edit control in dialog units.

Height: *numeric*
Height of the filename edit control in dialog units.

Style: *enum*
Filename edit control styles. Type | between enumerations to combine styles.

FilesAndDirs!	Directories and filenames allowed.
DirOnly!	Only directories allowed.
NoValidate!	No verification that a directory exists.

MacroVar: *variable*
The directory or filename entered in the edit box is returned in this variable. A name can be assigned to *MacroVar* before the macro compiles.

DefaultDir: *string* (optional)
The default directory when the Select File dialog box is displayed.

Template: *string* (optional)
The filenames to display in the Select File dialog box. For example, "*.*" (all files) and "*.bat" (all batch files).

See Also

DIALOGDEFINE
DIALOGSHOW

DIALOGADDFRAME

F

Purpose

Adds a frame to a dialog box.

You can combine frame styles by creating two frames the same size and at the same position. For example, a black frame and a white-filled frame (see Control and Style parameters). You can also create custom controls by combining frames and hot spots (see DIALOGADDHOTSPOT and DIALOGSHOW).

Syntax

DIALOGADDFRAME (*Dialog; Control; Left; Top; Width; Height; Style*)

Parameters

Dialog: *string*
The name or number of the dialog box to contain the frame control (see DIALOGDEFINE).

Control: *string*
A name or number that identifes the frame control (see DIALOGSHOW).

Left: *numeric*
Number of dialog units from the left side of the dialog box to the left side of the frame control.

Top: *numeric*
Number of dialog units from the top of the dialog box to the top of the frame control.

Width: *numeric*
Width of the frame control in dialog units.

Height: *numeric*
Height of the frame control in dialog units.

Style: *enum*
Frame styles. Type | between enumerations to combine styles. Not all combinations are possible.

Frame!	Frame only.
Black!	Black frame.
Gray!	Gray frame. Gray is the default DeskTop color of the Windows Default color scheme. To select a different color, choose Colors in the Windows Control Panel.
White!	White frame.
Filled!	Filled frame.

Style combination examples

Frame! | White!

Explanation: White frame displayed.

Filled! | White!

Explanation: White-filled frame displayed.

See Also

DIALOGADDGROUPBOX
DIALOGDEFINE
DIALOGSHOW

DIALOGADDGROUPBOX

F

Purpose

Draws a rectangle around related dialog controls such as buttons.

Grouping related controls helps narrow user options.

Syntax

DIALOGADDGROUPBOX (*Dialog; Control; Left; Top; Width; Height; Title*)

Parameters

Dialog: *string*
The name or number of the dialog box to contain the group box control (see DIALOGDEFINE).

Control: *string*
A name or number that identifes the group box control (see DIALOGSHOW).

Left: *numeric*
Number of dialog units from the left side of the dialog box to the left side of the group box control

Top: *numeric*
Number of dialog units from the top of the dialog box to the top of the group box control.

Width: *numeric*
Width of the group box control in dialog units.

Height: *numeric*
Height of the group box control in dialog units.

Title: *string*
The title displayed at the top left corner.

See Also

DIALOGADDFRAME
DIALOGADDHLINE
DIALOGADDVLINE
DIALOGDEFINE
DIALOGSHOW

DIALOGADDHLINE

F

Purpose
Adds a horizontal line to a dialog box.

Syntax
DIALOGADDHLINE (*Dialog; Control; Left; Top; Length*)

Parameters
Dialog: *string*
The name or number of the dialog box to contain the horizontal line control (see DIALOGDEFINE).

Control: *string*
A name or number that identifes the horizontal line control.

Left: *numeric*
Number of dialog units from the left side of the dialog box to the left side of the horizontal line control.

Top: *numeric*
Number of dialog units from the top of the dialog box to the top of the horizontal line control.

Length: *numeric*
Length of the horizontal line control in dialog units.

See Also
DIALOGADDGROUPBOX
DIALOGADDFRAME
DIALOGADDVLINE
DIALOGDEFINE
DIALOGSHOW

DIALOGADDHOTSPOT

F

Purpose

Creates an invisible area (hot spot) in a dialog box that closes the dialog box when clicked. Other responses are possible when you use DIALOGADDHOTSPOT with a callback function (see DIALOGSHOW).

To create a hot spot within a frame (see DIALOGADDFRAME), overlap size and position parameters.

Syntax

DIALOGADDHOTSPOT (*Dialog; Control; Left; Top; Width; Height; Style*)

Parameters

Dialog: *string*
The name or number of the dialog box to contain the hot spot control (see DIALOGDEFINE).

Control: *string*
A name or number that identifes the hot spot control (see DIALOGSHOW). The Control value is returned in the implicit variable MacroDialogResult (see DIALOGADDPUSHBUTTON or DIALOGDEFINE) if the user chooses the hot spot to dismiss the dialog box.

Left: *numeric*
Number of dialog units from the left side of the dialog box to the left side of the hot spot control.

Top: *numeric*
Number of dialog units from the top of the dialog box to the top of the hot spot control.

Width: *numeric*
Width of the hot spot control in dialog units.

Height: *numeric*
Height of the hot spot control in dialog units.

Style: *enum*
Hot spot styles.

Click!	Single click closes dialog box.
DblClick!	Double click closes dialog box.

See Also

DIALOGDEFINE

DIALOGADDICON

F

Purpose
Adds an icon to a dialog box.

Syntax
DIALOGADDICON (*Dialog; Control; Left; Top; Width; Height; IconName; Instance*)

Parameters
Dialog: *string*
The name or number of the dialog box to contain the icon control (see DIALOGDEFINE).

Control: *string*
A name or number that identifes the icon control (see DIALOGSHOW).

Left: *numeric*
Number of dialog units from the left side of the dialog box to the left side of the icon control.

Top: *numeric*
Number of dialog units from the top of the dialog box to the top of the icon control.

Width: *numeric*
Width of the icon control in dialog units.

Height: *numeric*
Height of the icon control in dialog units.

IconName: *string*
The name of an icon in a DLL. If an icon is identified by a number, precede the number with a pound sign (#). For example, "#160".

ModuleInstance: *numeric*
The handle of the DLL that contains the icon. The Shared Code default is 0. DLLLOAD returns the handle in a variable.

See Also
DIALOGDEFINE
DIALOGSHOW
DLLLOAD

DIALOGADDLISTBOX

F

Purpose

Adds a list box control to a dialog box.

Displays a list of options to choose from. The option you choose is returned in MacroVar (see MacroVar parameter). Use DIALOGADDLISTITEM to create list box items.

Syntax

DIALOGADDLISTBOX (*Dialog; Control; Left; Top; Width; Height; Style; MacroVar*)

Parameters

Dialog: *string*
The name or number of the dialog box to contain the list box control (see DIALOGDEFINE).

Control: *string*
A name or number that identifes the list box control (see DIALOGSHOW).

Left: *numeric*
Number of dialog units from the left side of the dialog box to the left side of the list box control.

Top: *numeric*
Number of dialog units from the top of the dialog box to the top of the list box control.

Width: *numeric*
Width of the list box control in dialog units.

Height: *numeric*
Height of the list box control in dialog units.

Style: *enum*
List box control styles. Type | between enumerations to combine styles.

Unsorted!	Items listed in order defined. Type a keyboard character to give the input focus to the first item that begins with that character. A vertical scrollbar automatically appears when needed. See DIALOGADDLISTITEM.
Sorted!	Default. Items sorted alphabetically. Type a keyboard character to give the input focus to the first item that begins with that character. See DIALOGADDLISTITEM.
NameSearch!	Item search allowed. Begin typing the name of an item to select. An edit control is displayed above the list box, and the input focus is given to the item that most closely matches the characters you type.
MultiColumn!	Items automatically roll into other columns as needed. A horizontal scrollbar automatically appears when needed. Column width is set by Windows.
WPChars!	Non-keyboard characters allowed in a list item. See DIALOGADDLISTITEM.

Style combination examples

Sort! | NameSearch!

Explanation: Name search allowed on a sorted list box. A vertical scrollbar automatically appears when needed.

Sort! | NameSearch! | MultiColumn!

Explanation: Name search allowed on a sorted list box. Items automatically roll in other columns as needed. A horizontal scrollbar automatically appears when needed.

MacroVar: *variable*
The selected list box item is returned in this variable. An item can be assigned to *MacroVar* before the macro compiles.

See Also

DIALOGDEFINE
DIALOGSHOW
DIALOGADDCOMBOBOX
DIALOGADDLISTITEM

DIALOGADDLISTITEM

F

Purpose

Creates a list item for DIALOGADDCOMBOBOX, DIALOGADDLISTBOX, or DIALOGADDPOPUPBUTTON.

Syntax

DIALOGADDLISTITEM (*Dialog; Control; Item*)

Parameters

Dialog: *string*
The name or number of the dialog box to contain the list item (see DIALOGDEFINE).

Control: *string*
The name or number of the control to contain the list item.

Item: *string*
A list box item. Use NTOC programming command to include non-keyboard characters in a list item for DIALOGADDCOMBOBOX and DIALOGADDLISTBOX, when the WPChars! style is set. For example,

```
DIALOGADDLISTITEM(1000; 100; "Sample" + NTOC(5;10))
```

See Also

DIALOGDEFINE
DIALOGSHOW
DIALOGADDCOMBOBOX
DIALOGADDLISTBOX
DIALOGADDPOPUPBUTTON

DIALOGADDPOPUPBUTTON

F

Purpose

Adds a popup button control to a dialog box.

Use DIALOGADDLISTITEM to create items for a popup button control. Unless *MacroVar* is assigned a default item, the item displayed on the popup button is the first item created by DIALOGADDLISTITEM.

Syntax

DIALOGADDPOPUPBUTTON (*Dialog; Control; Left; Top; Width; Height; MacroVar*)

Parameters

Dialog: *string*
The name or number of the dialog box to contain the popup button control (see DIALOGDEFINE).

Control: *string*
A name or number that identifes the popup button control (see DIALOGSHOW).

Left: *numeric*
Number of dialog units from the left side of the dialog box to the left side of the popup button control.

Top: *numeric*
Number of dialog units from the top of the dialog box to the top of the popup button control.

Width: *numeric*
Width of the popup button control in dialog units.

Height: *numeric*
Height of the popup button control in dialog units.

MacroVar: *variable*
The item displayed on the popup button is returned in this variable. An item can be assigned to *MacroVar* before the macro compiles.

See Also

DIALOGDEFINE
DIALOGSHOW
DIALOGADDCOMBOBOX
DIALOGADDLISTBOX
DIALOGADDLISTITEM

DIALOGADDPUSHBUTTON

F

Purpose

Adds a push button to a dialog box.

When you choose this button, the Control parameter value is returned in the implicit variable MacroDialogResult (see DIALOGDEFINE). If you choose another push button, its Control parameter value is returned: 1 for OK, 2 for Cancel, 2 for Close (system menu box), 2 if you double-click the system menu box, 2 if you press Alt+F4, the Control parameter value of another user-defined push button, or the Control parameter value of a hot spot.

Syntax

DIALOGADDPUSHBUTTON (*Dialog; Control; Left; Top; Width; Height; Style; ButtonText*)

Parameters

Dialog: *string*
A name or number that matches the Dialog parameter of the dialog box to contain the push button control (see DIALOGDEFINE).

Control: *string*
A unique string or number that identifes the push button control.

Left: *numeric*
Number of dialog units from the left side of the dialog box to the left side of the push button control.

Top: *numeric*
Number of dialog units from the top of the dialog box to the top of the push button control.

Width: *numeric*
Width of the push button control in dialog units.

Height: *numeric*
Height of the push button control in dialog units.

Style: *enum*
Push button styles.

NonDefaultBttn!	Creates a push button that dismisses the dialog box.
DefaultBttn!	Creates a default push button, with a heavy black border, that dismisses the dialog box. Press Enter to choose the default button. If Enter2HRtn! style is specified in DIALOGDEFINE, pressing Enter in a multiline edit control produces a hard return rather than the default button action.

ButtonText: *string*
The text displayed on the push button.

See Also

DIALOGDEFINE
DIALOGSHOW
DIALOGDESTROY

MacroDialogResult

DIALOGADDRADIOBUTTON

F

Purpose

Adds a radio button to a dialog box.

Radio buttons represent mutually exclusive choices. Selecting one radio button deselects another. Group them together by a control such as a group box or horizontal or vertical line. For multiple selections, see DIALOGADDCHECKBOX.

Syntax

DIALOGADDRADIOBUTTON (*Dialog; Control; Left; Top; Width; Height; ButtonText; MacroVar*)

Parameters

Dialog: *string*
The name or number of the dialog box to contain the radio button control (see DIALOGDEFINE).

Control: *string*
A name or number that identifes the radio button control (see DIALOGSHOW).

Left: *numeric*
Number of dialog units from the left side of the dialog box to the left side of the radio button control.

Top: *numeric*
Number of dialog units from the top of the dialog box to the top of the radio button control.

Width: *numeric*
Width of the radio button control in dialog units.

Height: *numeric*
Height of the radio button control in dialog units.

ButtonText: *string*
The text displayed to the right of the radio button.

MacroVar: *variable*
1 is returned in this variable if the radio button is selected, and 0 if not. The default value is 0. If *MacroVar* is assigned a value greater than 0 before the macro compiles, the radio button is automatically selected.

See Also

DIALOGADDCHECKBOX
DIALOGDEFINE
DIALOGSHOW

DIALOGADDSCROLLBAR

F

Purpose

Adds a horizontal or vertical scroll bar to a dialog box.

A callback function (see DIALOGSHOW) can receive Windows WM_HSCROLL and WM_VSCROLL messages from DIALOGADDSCROLLBAR. The wParam parameter of both messages indicates the mouse location when the scroll bar is clicked, or the thumb position when the thumb is dragged. The values are:

0 Right arrow clicked.
1 Left arrow clicked.
2 Area between the right arrow and thumb clicked.
3 Area between the left arrow and thumb clicked.
4 Position of thumb after it is dragged, passed as the low-order word of *lParam*.
5 The current position of thumb as it is dragged, passed as the low-order word of *lParam*.
6 Home key pressed while scroll bar has the input focus.
7 End key pressed while scroll bar has the input focus.
8 Scroll bar activity has ended.

See WM_HSCROLL and WM_VSCROLL in *Programmer's Reference, Volume 3: Messages, Structures, and Macros* (Windows 3.1 documentation).

Syntax

DIALOGADDSCROLLBAR (*Dialog; Control; Left; Top; Width; Height; Style; MacroVar; Minimum; Maximum*)

Parameters

Dialog: *string*
The name or number of the dialog box to contain the scroll bar control (see DIALOGDEFINE).

Control: *string*
A name or number that identifes the scroll bar control (see DIALOGSHOW).

Left: *numeric*
Number of dialog units from the left side of the dialog box to the left side of the scroll bar control.

Top: *numeric*
Number of dialog units from the top of the dialog box to the top of the scroll bar control.

Width: *numeric*
Width of the scroll bar control in dialog units (0 for vertical scroll bar).

Height: *numeric*
Height of the scroll bar control in dialog units (0 for horizontal scroll bar).

Style: *enum*
Scroll bar thumb position and type styles. Type | between enumerations to combine styles. Not all combinations are possible.

Left! Position the thumb at the left end of a horizontal scroll bar.

Initializing MacroVar parameter overrides this parameter.

Top!	Position the thumb at the top end of a vertical scroll bar. Initializing MacroVar parameter overrides this parameter.
Right!	Position the thumb at the right end of a horizontal scroll bar. Initializing MacroVar parameter overrides this parameter.
Bottom!	Position the thumb at the bottom end of a vertical scroll bar. Initializing MacroVar parameter overrides this parameter.
VScroll!	Create a vertical scroll bar. The Width parameter does not apply to this style.
HScroll!	Create a horizontal scroll bar. The Height parameter does not apply to this style.

Style combination examples

Left! | HScroll!

Explanation: Creates a horizontal scroll bar with the thumb at the left end.

Top! | VScroll!

Explanation: Creates a vertical scroll bar with the thumb at the top.

MacroVar: *variable*
The position of the thumb is returned in this variable. To pre-select a thumb position, initialize *MacroVar* to the desired position.

Minimum: *numeric*
The minimum thumb position. If the maximum position is 6 and the minimum position is 1, there are six positions on the scroll bar. If the maximum position is 6 and the minimum position is 2, there are five positions on the scroll bar.

Maximum: *numeric*
The maximum thumb position. See Minimum parameter.

See Also
DIALOGDEFINE
DIALOGDESTROY
DIALOGSHOW

DIALOGADDTEXT

F

Purpose
Displays a line of text in a dialog box.

Syntax
DIALOGADDTEXT (*Dialog; Control; Left; Top; Width; Height; Style; Text*)

Parameters
Dialog: *string*
The name or number of the dialog box to contain the text control (see DIALOGDEFINE).

Control: *string*
A name or number that identifes the text control (see DIALOGSHOW).

Left: *numeric*
Number of dialog units from the left side of the dialog box to the left side of the text control.

Top: *numeric*
Number of dialog units from the top of the dialog box to the top of the text control.

Width: *numeric*
Width of the text control in dialog units. This parameter determines how much horizontal space the control occupies, not the text width which is fixed by the system.

Height: *numeric*
Height of the text control in dialog units. This parameter determines how much vertical space the control occupies, not the text height which is fixed by the system.

Style: *enum*
Text control styles. Type | between enumerations to combine styles. Not all combinations are possible.

Left!	Left justified.
Right!	Right justified unless WPChars! style is set.
Center!	Centered inside Width parameter value unless WPChars! style is set.
RecessBox!	Recessed background.
ShadowBox!	Shadowed background.
WPChars!	Non-keyboard characters allowed. See Text parameter.
Multiline!	Text wraps if WPChars! is set.
Filename!	Ellipsis truncate path and filename to length of Width parameter.

Style combination examples

Left! | RecessBox!

Explanation: Text is left justified in a recess box.

Center! | ShadowBox!

Explanation: Text is centered in a shadow box.

Text: *string*
Text to display on the dialog box. Use NTOC programming command to display a non-keyboard charcter. For example,

Text: "A bullet character " + NTOC(4; 0)

See Also

DIALOGDEFINE
DIALOGSHOW
DIALOGDESTROY

DIALOGADDVIEWER

F

Purpose

Adds a read-only edit control to a dialog box. The edit control displays a file.

The edit control has horizontal and vertical scroll bars. The horizontal scroll bar contains text that identifies the text file format and indicates whether text wraps.

Syntax

DIALOGADDVIEWER (*Dialog; Control; Left; Top; Width; Height; Filename*)

Parameters

Dialog: *string*
The name or number of the dialog box to contain the viewer control (see DIALOGDEFINE).

Control: *string*
A name or number that identifes the viewer control (see DIALOGSHOW).

Left: *numeric*
Number of dialog units from the left side of the dialog box to the left side of the viewer control.

Top: *numeric*
Number of dialog units from the top of the dialog box to the top of the viewer control.

Width: *numeric*
Width of the viewer control in dialog units.

Height: *numeric*
Height of the viewer control in dialog units.

Filename: *string*
The path and name of a file to view.

See Also

DIALOGDEFINE

DIALOGADDVLINE

F

Purpose
Adds a vertical line to a dialog box.

Syntax
DIALOGADDVLINE (*Dialog; Control; Left; Top; Length*)

Parameters
Dialog: *string*
The name or number of the dialog box to contain the vertical line control (see DIALOGDEFINE).

Control: *string*
A name or number that identifes the vertical line control.

Left: *numeric*
Number of dialog units from the left side of the dialog box to the left side of the vertical line control.

Top: *numeric*
Number of dialog units from the top of the dialog box to the top of the vertical line control.

Length: *numeric*
Length of the vertical line in dialog units.

See Also
DIALOGADDGROUPBOX
DIALOGADDFRAME
DIALOGADDHLINE
DIALOGDEFINE
DIALOGSHOW

DIALOGDEFINE

F

Purpose

Creates a dialog box in memory.

Use DIALOGSHOW to display a dialog box created by DIALOGDEFINE.

The value of the control that dismisses a dialog box is returned in an implicit variable named MacroDialogResult: 1 for OK, 2 for Cancel, 2 for Close (system menu box), 2 if you double-click the system menu box, 2 if you press Alt+F4, or the value of the Control parameter of a user-defined push button or hot spot (see DIALOGADDPUSHBUTTON, DIALOGADDHOTSPOT).

MacroDialogResult is replaced or destroyed when you close a dialog box or call DIALOGDESTROY. To save its value, assign it to a variable. For example, vResult := MacroDialogResult.

Dialog boxes are positioned and sized in dialog units. A vertical unit equals 1/8 of the font height, and a horizontal unit equals 1/4 of the font width.

Syntax

DIALOGDEFINE (*Dialog; Left; Top; Width; Height; Style; Caption*)

Parameters

Dialog: *string*
A name or number that identifies the dialog box. This name or number is used by DIALOGSHOW and by DIALOGADD commands.

Left: *numeric*
Number of dialog units from the left side of the screen to the left side of the dialog box. When used with Percent! (see Style parameter), this number is a percentage of the screen width minus the width of the dialog box.

Top: *numeric*
Number of dialog units from the top of the screen to the top of the dialog box. When used with Percent! (see Style parameter), this number is a percentage of the screen height minus the height of the dialog box.

Width: *numeric*
Width of the dialog box in dialog units.

Height: *numeric*
Height of the dialog box in dialog units.

Style: *enum*
Dialog box styles. Type | between enumerations to combine styles. Not all combinations are possible.

OK!	Create an OK button.
Cancel!	Create a Cancel button.
Modeless!	Dialog box can lose input focus without closing. The WordPerfect command, InhibitInput, must be Off!. Default is modal dialog box.

Percent!	Sets Top and Left parameters to a percentage of the screen width or height minus the width or height of the dialog box. Use 50 to display the dialog box in the center of the screen.
NoFrame!	Create dialog box without a frame.
Sizeable!	Resizing of dialog box allowed.
NoTitle!	Remove the caption (title) bar. Prevents moving the dialog box.
Enter2HRtn!	Press Enter in a multiline edit control (see DIALOGADDEDITBOX) to move the insertion point to the next line. This does not cause the default button action (see DIALOGADDPUSHBUTTON).

Style combination example

OK! | Percent! | NoFrame!

Explanation: Creates a modal dialog box without a frame, and positions it in the center of the screen.

Caption: *string*
The text displayed in the caption (title) bar.

See Also

DIALOGDESTROY
DIALOGDISMISS
DIALOGSHOW

DIALOGDESTROY

Purpose
Removes an old-style dialog box from memory.

Syntax
DIALOGDESTROY (*Dialog*)

Parameters
Dialog: *string*
The name or number of a dialog box to destroy (see DIALOGDEFINE).

See Also
DIALOGDEFINE
DIALOGDISMISS
DIALOGSHOW

DIALOGDISMISS

Purpose
Hides a dialog box. Use DIALOGSHOW to display a hidden dialog box.

Syntax
DIALOGDISMISS (*Dialog; Control*)

Parameters
Dialog: *string*
The name or number of the dialog box to hide (see DIALOGSHOW).

Control: *string*
OKBttn or 1 for OK, CancelBttn or 2 for Cancel, or the control value of a user-defined push button or hot spot. CancelBttn cancels value assignments to control return variables. For example, an edit control does not return the text entered in the edit box. The value of Control is returned in MacroDialogResult as OKBttn, CancelBttn, or the control value of another push button or hot spot.

See Also
MacroDialogResult

DIALOGDISPLAY

Purpose

Displays a dialog box (backward compatibility only). Use DIALOGSHOW.

DIALOGHANDLE

Purpose
Returns the handle of a dialog box or dialog box control.

Syntax
DIALOGHANDLE (*MacroVar; Dialog; Control*)

Returns
The handle of a dialog box or dialog box control (see MacroVar parameter).

Parameters
MacroVar: *variable*
The handle of a dialog box or dialog box control is returned in this variable (see Dialog and Control parameters).

Dialog: *string*
The name or number of a dialog box (see DIALOGDEFINE).

Control: *string* (optional)
The name or number of a dialog box control (see Control parameter of any dialog control). If you omit this parameter, the handle of the dialog box is returned in MacroVar.

See Also
DIALOGADDCONTROL
DIALOGDEFINE

DIALOGSHOW

F

Purpose

Displays a dialog box, and, optionally, identifies a callback function. Use DIALOGDISMISS to hide a dialog box.

A callback function enables a macro to respond immediately and in specific ways to events such as selecting a radio button or check box, without waiting until the dialog box is dismissed. Without a callback function, user input is acted upon only after the dialog box is dismissed and the macro resumes execution.

To create a callback function, use the Label parameter of DIALOGSHOW, and a loop statement after DIALOGSHOW. For example,

```
DIALOGSHOW(1000; "WordPerfect"; Msgs)
WHILE(x = 0)
ENDWHILE

LABEL(Msgs)
    ... other statements ...
    RETURN
```

The value of the control that dismisses a dialog box is returned in an implicit variable named MacroDialogResult: 1 for OK, 2 for Cancel, 2 for Close (system menu box), 2 if you double-click the system menu box, 2 if you press Alt+F4, or the value of the Control parameter of a user-defined push button or hot spot (see DIALOGADDPUSHBUTTON, DIALOGADDHOTSPOT).

If DIALOGSHOW is used with DIALOG commands, such as DIALOGDEFINE and DIALGOADDEDITBOX (old-style dialog box), the first control that can return a value has the input focus.

Syntax

DIALOGSHOW (*Dialog; Parent; Label*)

Parameters

Dialog: *string*
The name of a dialog box to display (see DIALOGDEFINE for old-style dialog boxes).

Parent: *string*
The name of the parent window, such as "WordPerfect" for WordPerfect 6.1 for Windows. The parameter is required by new-style dialog boxes (see WordPerfect 6.1 for Windows, Dialog Editor Help), but optional for old-style dialog boxes.

Label: *label* (optional)
Identifies a callback function (LABEL to call) when the dialog box becomes active, or the user

-Chooses Close from the system menu.
-Presses Alt+F4.
-Double-clicks the system menu box.
-Chooses a push button, radio button, check box, hot spot, or scroll bar control.

The Windows messages are:

-WM_ACTIVATE
-WM_COMMAND
-WM_HSCROLL
-WM_SYSCOMMAND
-WM_VSCROLL

An array is automatically created for callback functions that contains Windows message information. The array is given the same name as the Label parameter.

If Label is named DoItNow, array elements are defined as DoItNow[1], DoItNow[2], DoItNow[3], and so forth. The element values are:

[1] *CallbackType* Dialog box (type 3).
[2] *Dialog* Name (character string) or number that matches the Dialog parameter of the dialog box that contains the callback function.
[3] *Control* Name (character string) or number that matches the Control parameter of the control that causes the callback function to be called. The value is "OKBttn" if OK is pressed; or, "CancelBttn" if Cancel is pressed.
[4] *DialogHandle* Handle of the dialog box that starts the callback function.
[5] *Message* The Windows message received from the dialog box. The values are:
6 The dialog box becomes active (WM_ACTIVATE).
273 The user chooses a window control, such as a hot spot, push button, or radio button (WM_COMMAND).
276 The user clicks a scroll bar (WM_HSCROLL).
274 The user chooses Close from the system menu, presses Alt+F4, or double-clicks the system menu box (WM_SYSCOMMAND).
277 The user clicks a scroll bar (WM_VSCROLL).
[6] *wParam* Contains a WORD (16-bit unsigned integer) value that is passed with *Message*. For example, if element 5 equals 273 (WM_COMMAND), the value of element 6 is 1 if you choose OK, or 2 if you choose Cancel. If element 5 equals 274 (WM_SYSCOMMAND), the value of element 6 is 61536 if you choose Close from the system menu, press Alt+F4, or double-click the system menu box.
[7] *lParam* Contains a DWORD (32-bit unsigned integer) value that is passed with *Message*. This value contains a high-order value and a low-order value, which you calculate with bitwise operators. The low-order value equals x := x & 65535 (mask hi-order value). The high-order value equals x := x >> 16 (shift right 16 bits).

Elements 6 (*wParam*) and 7 (*lParam*) have different meanings for every Windows message. See specified messages in *Programmer's Reference, Volume 3: Messages, Structures, and Macros* (Windows 3.1 documentation) for more information.

See Also
MacroDialogResult

DIALOGUNDISPLAY

Purpose

Hides a dialog box (backward compatibility only). Use DIALOGDISMISS.

DIMENSIONS

F

Purpose
Returns dimension information about an array variable.

Syntax
numeric DIMENSIONS (VariableName: *variable*; IndexNumber: *numeric*)

Return Value
numeric Number of array dimensions, total number of array elements, number of array elements in a single dimension, or 0 if the index number is out of range (see IndexNumber parameter).

Parameters
VariableName: *variable*
Array variable.

IndexNumber: *numeric* (optional)
If not specified, the number of array dimensions is returned. Use 0 to return the total number of array elements. Use a dimension number to return the number of elements in that dimension. Given the following declaration,

```
DECLARE array[4; 2; 5]
```

DIMENSIONS returns the following values:

```
3   DIMENSIONS(array[]) // number of dimensions
40  DIMENSIONS(array[]; 0) // total elements
4   DIMENSIONS(array[]; 1) // elements in dimension 1
2   DIMENSIONS(array[]; 2) // elements in dimension 2
5   DIMENSIONS(array[]; 3) // elements in dimension 3
0   DIMENSIONS(array[]; 4) // out of range
0   DIMENSIONS(c) // not an array
0   DIMENSIONS(b[]) // doesn't exist
```

DISCARD

F

Purpose

Removes LOCAL, GLOBAL, and PERSIST variables from memory, in that order.

DISCARD does not specify a variable table. If a variable by the same name exists in all three tables, call DISCARD three times.

```
WHILE(EXISTS(VariableName))
    DISCARD(VariableName)
ENDWHILE
```

Syntax

DISCARD (*VariableName; VariableName; ... VariableName*)

Parameters

VariableName: *variable*
Begins with a letter and can include any other combination of letters or numbers. Separate multiple variables with a semicolon.

See Also

EXISTS
GLOBAL
LOCAL
PERSIST
PERSISTALL

DLLCALL

F

Purpose

Calls a function in a Dynamic Link Library (DLL).

F Do not use this command unless you are familiar with Windows programming. Incorrect DLL calls can damage files and/or reset your computer.

Syntax

DLLCALL (*ModuleInstance; FunctionName; ReturnVariable; ReturnType; {Parameter; Parameter; ...}*)

Parameters

ModuleInstance: *numeric*
A DLL handle. DLLLOAD returns the handle in a variable.

FunctionName: *string*
A function name. See DLL documentation for function names.

ReturnVariable: *variable*
The DLL function may return a value in this variable. The ReturnType parameter defines the expected data type.

ReturnType: *enum*
The data type to return (see ReturnVariable parameter).

Data Type	Description
WPString	WordPerfect 6.0 wordstring
AnsiString	Windows type LPSTR
OemString	Windows type LPSTR
String	Windows type LPSTR
Bool	Windows type BOOL
DWord	Windows type DWORD
Integer	Windows type LONG
Word	Windows type WORD
Real	Microsoft C type double
Void	Nothing -- A function of type void can be called if the return variable name is not used and void is specified as the return type.

F ReturnType must match the data type returned in ReturnVariable (see DLL function documentation for return types). If it does not, an Unrecoverable Application Errror or system crash may result.

Parameter: *any*
Parameter information passed to a DLL function, enclosed in braces and separated by a semicolon. The type of information depends on the function being called (see DLL function documentation for parameter values to pass). Use LOWORD specifier to pass byte data.

Data Types	Description
Integer	LONG or DWORD
String	FAR pointer to an ANSI string
Boolean	BOOL
Real	Microsoft c-type double
ANSISTRING(*string*)	FAR pointer to an ANSI string
OEMSTRING(*string*)	FAR pointer to an OEM string
WPString(*string*)	FAR pointer to a WP 6.0 word string
LOWORD(*Integer*)	int, short, or WORD (low order bits passed to DLL function)
HIWORD(*Integer*)	int, short, or WORD (high order bits passed to DLL function)
ADDRESS(*Integer*)	LPDWORD
ADDRESS(WORD(*Integer*))	
ADDRESS(DWORD(*Integer*))	
ADDRESS(*Boolean*)	BOOL FAR*
ADDRESS(BOOLEAN(*Boolean*))	BOOL FAR*
ADDRESS(*string*)	FAR pointer to an ANSI string
ADDRESS(ANSISTRING(*string*))	FAR pointer to an ANSI string
ADDRESS(OEMSTRING(*string*))	FAR pointer to OEM string
ADDRESS(WPSTRING(*string*))	FAR pointer to a WP 6.0 word string.
ADDRESS(*Real*)	FAR pointer to a MicroSoft C-type double
ADDRESS(REAL(*Real*))	FAR pointer to a MicroSoft C-type double

See Also

DLLCALL PROTOTYPE
DLLFREE
DLLLOAD

DLLCALL PROTOTYPE

Purpose

Defines the calling format of a DLL routine (see example).

DLLCALL PROTOTYPE accepts only literal strings (no expressions), and must occur before calling a DLL function. The DLL library is loaded and freed each time the DLL function is called.

DLLCALL PROTOTYPE makes a macro easier to read, and provides safer DLL calls (routines can only be called as defined).

 Do not use this command unless you are familiar with Windows programming. Incorrect DLL calls can damage files and/or reset your computer.

DLLCALL PROTOTYPE *Name* (*ModuleFileName; FunctionName; ReturnType;* {*Parameter; Parameter; ...*})

Parameters

Name: *string*
The name used to call the DLL function.

ModuleFileName: *string*
DLL library file containing the DLL function.

FunctionName: *string*
A function name. See DLL documentation for function names.

ReturnType: *enum*
The data type returned by the DLL function. See DLLCALL.

- AnsiString!
- Bool!
- DWord!
- Integer!
- OemString!
- Real!
- String!
- Word!
- Void!

Parameter: *any*
Parameter information passed to a DLL function, enclosed in braces and separated by a semicolon. The type of information depends on the function being called (see DLL function documentation for parameter values to pass). See DLLCALL.

See Also

DLLFREE
DLLLOAD

DLLFREE

Purpose
Removes a dynamic link library (DLL) from memory.


If you use DLL commands incorrectly, they can damage files and/or reset your computer.

Syntax
DLLFREE (*ModuleInstance*)

Parameters
ModuleInstance: *numeric*
A DLL handle. DLLLOAD returns the handle in a variable.

See Also
DLLCALL
DLLLOAD

DLLLOAD

F

Purpose

Loads a dynamic link library (DLL) into memory.

You should respond to error values with code similar to the following (see ModuleInstance parameter):

```
DLLLOAD(vResult; Filename)
SWITCH(vResult)
    CASEOF 0: ...statement block...
    CASEOF 2: ...statement block...
ENDSWITCH
```

F If you use DLL commands incorrectly, they can damage files and/or reset your computer.

Syntax

DLLLOAD (*ModuleInstance; Filename*)

Parameters

ModuleInstance: *numeric*
A DLL handle if the DLL successfully loads, or an error value less than 32 if not. Partial list of error values:

0 Not enough system memory.
2 File not found.
3 Path not found.
10 Incorrect Windows version.
20 DLL is invalid.

See LoadLibrary function in *Programmer's Reference, Volume 2: Functions* (Windows 3.1 documentation), page 578, for a complete list of error values.

Filename: *string*
A DLL path and filename.

See Also

DLLFREE
DLLCALL

DOESDIRECTORYEXIST

Purpose
Returns True if a directory exists, False if not.

Syntax
boolean DOESDIRECTORYEXIST (DirectoryName: *string*)

Return Value
boolean True/False.

Parameters
DirectoryName: *string*
Include full path.

See Also
CREATEDIRECTORY
DELETEDIRECTORY
GETCURRENTDIRECTORY
RENAMEDIRECTORY
SETCURRENTDIRECTORY

DOESFILEEXIST

Purpose
Returns True if a file exists, False if not.

Syntax
boolean DOESFILEEXIST (Filename: *string*)

Return Value
boolean True/False.

Parameters
Filename: *string*
Include full path.

See Also
COPYFILE
DELETEFILE
RENAMEFILE

ENDAPP

F

Purpose

Identifies, for the compiler, an application that is no longer used in a macro (see APPLICATION).

ENDAPP is a non-executable statement that can occur anywhere in a macro. After ENDAPP, a product command to the application specified by ENDAPP creates a compile-time syntax error.

Syntax

ENDAPP (*ProductPrefix*)

Parameters

ProductPrefix: *string*
A unique identifier that matches the ProductPrefix parameter of an APPLICATION statement.

See Also

APPLICATION
NEWDEFAULT

ERROR

Purpose

Determines how a macro responds to an Error condition.

Create an Error condition with ASSERT(ErrorCondition!).

Syntax

ERROR (*State*)

Parameters

State: *enum*

Specifies the Error state. The default is ERROR(On!).

Off!	Overrides an Error condition.
On!	Stop a macro unless preceded by ONERROR, which directs macro execution to a specified LABEL.

See Also

ASSERT
CANCEL
LABEL
NOTFOUND
ONERROR

EXISTS

F

Purpose

Determines if a variable exists.

To exist, a variable must be declared and initialized. Given that variable w is declared and not initialized, and that LOCAL x is initialized to 5, GLOBAL y to 10, and PERSIST z to 15, the following statements are valid:

vNoInit := EXISTS(w)
Result: vNoInit equals 0

vLocal := EXISTS(x)
Result: vLocal equals 1

vGlobal := EXISTS(y)
Result: vGlobal equals 2

vPersist := EXISTS(z)
Result: vPersist = 3

Examples in shorthand notation

```
IF(EXISTS(vLocal))
    BEEP
ENDIF
```

Explanation: Computer beeps because EXISTS returns a value greater than 0.

```
IF(EXISTS(vLocal) = 3)
    BEEP
ENDIF
```

Explanation: The computer does not beep because EXISTS returns 1. (vLocal is a Local variable, not a Persist variable).

Syntax

ReturnValue := EXISTS (*VariableName*)

Returns

An integer. The values are:

0 Does not exist
1 Local variable
2 Global variable
3 Persist variable

Parameters

VariableName: *variable*
A variable declared as LOCAL, GLOBAL, or PERSIST. Variables begin with a letter and can include any other combination of letters or numbers.

See Also

DISCARD
LOCAL
GLOBAL
PERSIST
PERSISTALL

FILEERROR

Purpose

Returns the error associated with the last file command.

Syntax

numeric FILEERROR ()

Return Value

numeric A number greater than 0 (last file error), -1 (unknown error), or 0 (no error). Error numbers have loword and hiword values. The loword contains an error code generated by WordPerfect's I/O routines, described in the WordPerfect for Windows SDK (WERROR.H file). The hiword contains a Windows specific error value, described in *Programmer's Reference, Volume 3: Messages, Structures, and Macros* (Windows 3.1 documentation), page 344.

See Also

CLOSEFILE
FILEISEOF
FILEPOSITION
FILEREAD
FILESIZE
FILETRUNCATE
FILEWRITE
OPENFILE

FILEFIND

F

Purpose
Finds a file that matches user-defined search criteria (filename and DOS attributes).

Syntax
string FILEFIND (Filename: *string*; Attributes: *enum*; Context: *numeric*)

Return Value
string Filename, or an empty filename if no match is found.

Parameters
Filename: *string* (optional)
Include full path. Wild card specification is optional. Example: "*.WPD". If not specified, perform last search (see example under Context parameter).

Attributes: *enum* (optional)
Normal!
ReadOnly!
Hidden!
System!
Label!
Directory!
Archived!

Default: Normal!.

Context: *numeric* (optional)
User-defined number which identifies the search. If not specified, 0 is used. To find the next file to match the search criteria, call FILEFIND with an empty filename. For example,

```
vFilename = FILEFIND (Filename: "*.WPD"; Attributes: Normal!; Context: 1)
```

finds the first occurrence of a file with a WPD extension. The following statement finds the next file to match the same search criteria:

```
vFilename = FILEFIND (Filename: ""; Context: 1)

or...

vFilename = FILEFIND (Context: 1)
```

Use different Context values to perform multiple searches simultaneously. If you perform more than one search before the macro ends, you can repeat any search by using the appropriate Context value. The subsequent search begins where it left off.

See Also
COPYFILE
DELETEFILE
DOESFILEEXIST
FILESIZE
RENAMEFILE

FILEISEOF

Purpose

Returns True if the insertion point is at the end of a file, or False if not. The access mode must be Read! (see OPENFILE).

Syntax

boolean FILEISEOF (FileID: *numeric*)

Return Value

boolean True/False.

Parameters

FileID: *numeric*
See OPENFILE.

See Also

CLOSEFILE
FILEERROR
FILEPOSITION
FILEREAD
FILESIZE
FILETRUNCATE
FILEWRITE

FILEPOSITION

Purpose

Returns the current marker position or sets a new position.

A user can preallocate file size by setting the position marker past the end-of-file marker.

Syntax

numeric FILEPOSITION (FileID: *numeric*; NewPosition: *numeric*; PositionFrom: *enum*)

Return Value

numeric Position of old position marker, or negative number if error occurs.

Parameters

FileID: *numeric*
See OPENFILE.

NewPosition: *numeric* (optional)
Number of bytes to move position marker. A negative number moves the position marker left.

PositionFrom: *enum* (optional)
Move position marker relative to one of the following positions. If not specified, then position is relative to the start of the file.

- FromBeginning!
- FromCurrentPosition!
- FromEnd!

See Also

CLOSEFILE
FILEERROR
FILEISEOF
FILEREAD
FILESIZE
FILETRUNCATE
FILEWRITE

FILEREAD

F

Purpose
Reads text data from an open file.

Syntax
numeric FILEREAD (FileID: *numeric*; Data: *variable*)

Return Value
numeric Number of bytes read, or negative number if error occurs.

Parameters
FileID: *numeric*
See OPENFILE.

Data: *variable*
Starting from the position marker, one line of data is converted to a WP character string and returned in this variable.

See Also
CLOSEFILE
FILEERROR
FILEISEOF
FILEPOSITION
FILESIZE
FILETRUNCATE
FILEWRITE

FILESIZE

Purpose
Returns a file's size in bytes.

Syntax
numeric FILESIZE (Filename: *string*)

Return Value
numeric File size in bytes, or negative number if error occurs.

Parameters
Filename: *string*
Include full path.

See Also
COPYFILE
DELETEFILE
DOESFILEEXIST
FILEFIND
RENAMEFILE

FILETRUNCATE

Purpose
Removes all text data after the position marker (see FILEPOSITION).

Syntax
numeric FILETRUNCATE (FileID: *numeric*)

Return Value
numeric Truncated file size (bytes) if successful, or negative number if error occurs.

Parameters
FileID: *numeric*
See OPENFILE.

See Also
CLOSEFILE
FILEERROR
FILEISEOF
FILEREAD
FILESIZE
FILEWRITE

FILEWRITE

F

Purpose
Writes text data to a file.

Syntax
numeric FILEWRITE (FileID: *numeric*; Data: *string*; NewLine: *enum*; ParameterData: *string*)

Return Value
numeric Number of bytes written, or a negative number if error occurs.

Parameters
FileID: *numeric*
See OPENFILE.

Data: *string*
Text data is converted to the type specified when the file is opened. A caret (^) followed by a number is replaced by the corresponding character string in the ParameterData parameter. To write a caret, use two carets (^^).

NewLine: *enum*
NoNewLine!
NewLine!

Default: NewLine! (inserts an end-of-line marker).

ParameterData: *string*
Text data inserted in the data string (see Data parameter). Up to ten strings separated by semicolons are allowed. Numbering begins with 0. For example,

FILEWRITE (*FileID;* Data: "My friend ^0 is ^1 years old."; NewLine: NewLine!; ParameterData: {"Dan"; "41"})

writes "My friend Dan is 41 years old." followed by an end-of-line marker.

See Also
CLOSEFILE
FILEERROR
FILEISEOF
FILEPOSITION
FILEREAD
FILESIZE
FILETRUNCATE

FOR-ENDFOR

F

Purpose

A loop statement that executes a specified number of times (see Loop Statements).

The loop executes only if *TerminateExp* is true, and continues to execute until *ControlVariable's* value makes *TerminateExp* false.

ENDFOR closes a FOR statement.

Syntax

The general form of a FOR statement is,

```
FOR (ControlVariable; InitialValue; TerminateExp; IncrementExp)
    ...statement block...
ENDFOR
```

For example, the following statement initializes vTest to one, repeats the statement block while vTest is less than ten, and increments vTest by two after each loop.

```
FOR(vTest; 1; vTest < 10; vTest + 2)
    ...statement block...
ENDFOR
```

Parameters

ControlVariable: *variable*
Control variable.

InitialValue: *any*
Value assigned to *ControlVariable* at the start of a loop.

TerminateExp: *boolean*
The loop executes while *TerminateExp* is true.

IncrementExp: *any*
The value to increment *ControlVariable* after each loop.

See Also

FORNEXT
FOREACH
REPEAT
WHILE

FOREACH-ENDFOR

F

Purpose

A loop statement that executes a number of times equal to the number of specified expressions (see Loop Statements).

ControlVariable is assigned the first value before the first loop; the second value before the second loop; the third value before the third loop, and so forth. The statement block uses the value of *ControlVariable* to direct macro execution.

ENDFOR closes a FOREACH statement.

Syntax

The general form of a FOREACH statement is,

```
FOREACH (ControlVariable; {ValueList; ...})
    ...statement block...
ENDFOR
```

For example, the following statement repeats three times. The variable vTest is initialized to "Apples" before the first loop, to "Oranges" before the second, and to "Bananas" before the third.

```
FOREACH(vTest; {"Apples"; "Oranges"; "Bananas"})
    ... statement block ...
ENDFOR
```

Parameters

ControlVariable: *variable*
Control variable.

ValueList: *any*
Variables, constants, or expressions, enclosed in braces and separated by a semicolon.

See Also

FOR
FORNEXT
REPEAT
WHILE

FORNEXT-ENDFOR

F

Purpose

A loop statement that executes a specified number of times (see Loop Statements).

The loop executes if *ControlVariable* is less than or equal to *FinalValue,* and executes until *ControlVariable* is greater than *FinalValue.* If you use a negative value in *IncrementValue,* the loop executes until *ControlVariable* is less than *FinalValue*.

ENDFOR closes a FORNEXT statement.

Syntax

The general form of a FORNEXT statement is,

```
FORNEXT (ControlVariable; InitialValue; FinalValue; IncrementValue)
    ...statement block...
ENDFOR
```

The following statement initializes vTest to one, repeats the statement block while vTest is less than or equal to five, and increments vTest by two after each loop.

```
FORNEXT(vTest; 1; 5; 2)
    ...statement block...
ENDFOR
```

Parameters

ControlVariable: *variable*
Control variable.

InitialValue: *any*
The value assigned to *ControlVariable* at the start of a loop.

FinalValue: *any*
A variable, constant, or expression. The loop executes while *FinalValue* is greater than *ControlVariable,* or less than if *IncrementValue* is a negative value.

IncrementValue: *any* (optional)
The value to increment *ControlVariable* after each loop. Default is 1.

See Also

FOR
FOREACH
REPEAT
WHILE

FRACTION

Purpose
Returns the fractional portion of a real number.

Syntax
ReturnValue := FRACTION (*Value*)

Returns
A fraction.

vFraction := FRACTION(1.5)
Result: vFraction equals 0.5

vResult := FRACTION(1.77 * 2)
Result: vResult = 0.54

Parameters
Value: *numeric*
A real number.

See Also
INTEGER

FUNCTION-ENDFUNC

F

Purpose

Identifies a macro subroutine that can receive one or more values from a calling statement (see Calling Statements). FUNCTION also returns a value to the calling statement (caller).

Functions contain one or more statements that execute when the function is called. Unlike LABEL statements, functions do not execute unless called. A calling statement consists of the function's name, and can have one or more parameters that pass values to the function. RETURN, ENDFUNC, or ENDFUNCTION direct macro execution to the statement that follows the function's caller. RETURN also returns the result of a function operation, or 0 if no result is specified. For example,

```
y := 5
x := Add(y) // calling statement

FUNCTION Add(z)
    z := z + 5
    RETURN(z)
ENDFUNC
```

assigns the value 10 to variable x. Add is the name of the function, and y contains a value to pass. Variable z has no value until the function is called, when the value of z is 5 (the value of variable y). The first function statement assigns 10 to variable z. The next statement returns the value of z to the calling statement, which is then assigned to variable x. If the function did not contain a RETURN statement, or if RETURN did not return a value, x would be equal to 0.

Address Mode

In the above example, the value of y does not change. The function returns the value of z in variable x. To change the value of a variable passed to a function, precede the calling statement parameter and its corresponding function parameter with an ampersand (&). For example,

```
y := 5
x := Add(&y)

FUNCTION Add(&z)
    ASSIGN(z; z + 5)
    RETURN(z)
ENDFUNC
```

assigns the value 10 to both x and y. The ampersand before variable x means the variable's address (location in memory) is passed to the function, not the variable's value. Changes made at the address of x are made to the contents of x.

Arrays

You pass arrays to functions the same way you pass variables. If you are passing the entire array, every array element must be assigned a value. If not, a run-time error identifies a reference to the undefined element. For example,

```
DECLARE w[10]
FORNEXT(x; 1; 9; 1) // assign only 9 elements
    w[x] = x
ENDFOR

FUNCTION Test(z[])
    FORNEXT(x; 1; 10; 1)
        x[z] = x * 10
    ENDFOR
    RETURN(z[]) // the value of 10 elements returned
ENDFUNC

y[] = Test(w[]) // 10 elements returned in array y[]

Type(y[10]) // y[10] equals 100
Type(w[10]) // Run-time error: Undefined variable "W[10]"
```

In the previous example, if you preceed the calling statement parameter and the corresponding function parameter with an ampersand (&), 100 is returned in w[10]. No run-time error occurs. You are passing the address of array w[], not its value. The value is assigned inside the function (see Address Mode above). The two statements look like this:

```
FUNCTION Test(&z[])
```

and

```
y[] = Test(&w[])
```

Pass the value of an array element, the same way you pass a variable. For example,

```
DECLARE w[10]
FORNEXT(x; 1; 10; 1)
    w[x] = x
ENDFOR

FUNCTION Test(z)
    z := z * 10
    RETURN(z)
ENDFUNC

y = Test(w[1])

Type(y + " - ")
Type(w[1])
```

The value of y equals 10. The value of w[1] equals 1. If you preceed the calling statement parameter and the corresponding function parameter with an ampersand (&), the value of y equals 10 and the value of w[1] equals 10. Passing the address of w[1] causes its value to change inside the function. The two statements look like this:

```
FUNCTION Test(&z)
```

and

```
y = Test(&w[1])
```

Syntax

The general form of a FUNCTION statement is,

```
FUNCTION Name (Parameter; Parameter; ... Parameter)
    ...statement block...
ENDFUNC
```

The syntax accepts ENDFUNC or ENDFUNCTION.

Parameters

Name: *label*
The name of a function. It begins with a letter and consists of one or more letters or numbers.

Parameter: *variable*
Receives a value from a calling statement (see Calling Statements). If an ampersand precedes the calling statement variable, an ampersand must precede the corresponding function variable (see Address Mode above). Multiple variables are separated by a semicolon.

See Also

CALL
LABEL
PROCEDURE
RETURN
USE

FUNCTION PROTOTYPE

Purpose

Verifies the syntax of a FUNCTION statement (see FUNCTION).

If the syntax of a function contained in a USE macro file is incorrect and the function is called from the main macro, you get a run-time error but not a compile-time error (see Macro Errors). Using FUNCTION PROTOTYPE at the start of a main macro ensures that you get a compile-time error.

Syntax

FUNCTION PROTOTYPE *Name* (*Parameter; Parameter; ... Parameter*)

Parameters

Name: *label*

The name of the function. It begins with a letter and consists of one or more letters or numbers.

Parameter: *variable*

Receives a value from a calling statement (see Calling Statements). If an ampersand precedes the calling statement variable, an ampersand must precede the corresponding function variable (see Address Mode under FUNCTION). Multiple variables are separated by a semicolon.

GETCURRENTDIRECTORY

Purpose
Returns name of current directory.

Syntax
string GETCURRENTDIRECTORY ()

Return Value
string Name of current directory.

See Also
CREATEDIRECTORY
DELETEDIRECTORY
DOESDIRECTORYEXIST
RENAMEDIRECTORY
SETCURRENTDIRECTORY

GETFILEATTRIBUTES

Purpose
Returns DOS file attributes.

Syntax
numeric GETFILEATTRIBUTES (Filename: *string*)

Return Value
numeric One of the following values, or a negative number if error occurs.

0	Error!
1	ReadOnly!
2	Hidden!
4	System!
8	Label!
16	Directory!
32	Archived!

Archived! file has changed since last save.

Parameters
Filename: *string*
Include full path.

See Also
SETFILEATTRIBUTES

GETFILEDATEANDTIME

Purpose
Returns a file's creation date and time.

Syntax
numeric GETFILEDATEANDTIME (Filename: *string*)

Return Value
numeric File date and time (number of seconds since January 1, 1980), or -1 if error occurs.

Parameters
Filename: *string*
Include full path.

See Also
SETFILEDATEANDTIME

GETNUMBER

F

Purpose

Displays a dialog box that contains an edit control to enter an integer or real number.

Syntax

GETNUMBER (*MacroVar; Prompt; Title*)

Parameters

MacroVar: *variable*
The number entered in the edit box is returned in this variable.

Prompt: *string* (optional)
Text displayed above the edit control. Use NTOC(0F90Ah) to insert a hard return code in the prompt string. For example,

```
GETNUMBER(vAns; "First line" + NTOC(0F90Ah) + "Second line"; "Title")
```

creates two lines above the edit box. The next example creates three lines, including one blank line:

```
ASSIGN(HdReturn; NTOC(0F90Ah))
GETNUMBER(vAns; "First line" + HdReturn + HdReturn + "Second line"; "Title")
```

Title: *string* (optional)
Text displayed in the title bar.

See Also

GETSTRING
GETUNITS

GETSTRING

F

Purpose

Displays a dialog box that contains an edit control to enter a character string.

Syntax

GETSTRING (*MacroVar; Prompt; Title; Length*)

Parameters

MacroVar: *variable*
The character string entered in the edit box is returned in this variable.

Prompt: *string* (optional)
The text displayed above the edit control. Use NTOC(0F90Ah) to insert a hard return code in the prompt string. For example,

```
GETSTRING(vAns; "First line" + NTOC(0F90Ah) + "Second line"; "Title")
```

creates two lines above the edit box. The next example creates three lines, including one blank line:

```
ASSIGN(HdReturn; NTOC(0F90Ah))
GETSTRING(vAns; "First line" + HdReturn + HdReturn + "Second line"; "Title")
```

Title: *string* (optional)
The text displayed in the title bar.

Length: *numeric* (optional)
The maximum number of characters the edit control accepts. Default is unlimited.

See Also

GETNUMBER
GETUNITS

GETUNITS

F

Purpose

Displays a dialog box that contains an edit control to enter a number and unit of measure.

The units of measure are " (inches); i (inches); c (centimeters); m (millimeters); p (points—72 per inch); and w (WordPerfect units—1200 per inch). If a unit of measure is not specified, the default is WordPerfect units. To change the default use DEFAULTUNITS.

Syntax

GETUNITS (*MacroVar; Prompt; Title*)

Parameters

MacroVar: *variable*
The number and unit of measure entered in the edit box is returned in this variable. If no unit of measure is specified, the default is returned.

Prompt: *string* (optional)
The text displayed above the edit control. Use NTOC(0F90Ah) to insert a hard return code in the prompt string. For example,

```
GETUNITS(vAns; "First line" + NTOC(0F90Ah) + "Second line"; "Title")
```

creates two lines above the edit box. The next example creates three lines, including one blank line:

```
ASSIGN(HdReturn; NTOC(0F90Ah))
GETUNITS(vAns; "First line" + HdReturn + HdReturn + "Second line"; "Title")
```

Title: *string* (optional)
The text displayed in the title bar.

See Also

DEFAULTUNITS
GETSTRING
GETNUMBER
UNITSTR

GLOBAL

F

Purpose

Declares global variables and arrays, and assigns them to the global variable table.

Global variables can be used in RUN or CHAIN macros.

Syntax

GLOBAL (*VariableName; VariableName; ... VariableName*)

Parameters

VariableName: *variable*

One or more user-defined variables separated by a semicolon. Variables must begin with a letter and can include any other combination of letters or numbers. Parentheses are optional. For example,

```
GLOBAL VariableName; VariableName
GLOBAL(VariableName; VariableName)
```

are both valid statements. The following example declares global arrays with 10 elements and 20 elements,

```
GLOBAL FirstArray[10]; NextArray[20]
```

The last example simultaneously declares a variable and assigns a value:

```
GLOBAL x := 2
```

or...

```
GLOBAL x
x := 2
```

If you create two variables with the same name (for example, GLOBAL x and PERSIST x), the following statement

```
GLOBAL x := 5
```

specifies which variable *x* is assigned the value 5.

See Also

CHAIN
DECLARE
DISCARD
EXISTS
LOCAL
PERSIST
PERSISTALL
RUN

GO

Purpose

Jumps to a LABEL statement and does not return.

Go is generally used to exit multiple layers of nested statements, or to create a loop.

Syntax

Go (*label*)

Parameters

Label: *label*

The name of a label. It begins with a letter and consists of one or more letters or numbers.

See Also

CALL
INDIRECT
LABEL

IF-ELSE-ENDIF

F

Purpose

A conditional statement that determines whether a statement (or statement block) is executed. See Conditional Statements.

If *Test* is true, the statements between IF and ELSE are executed. If false, the statements between ELSE and ENDIF are executed.

ELSE is optional. If *Test* is false and ELSE is not used, the macro skips the statements between IF and ENDIF, and executes the first statement after ENDIF.

ENDIF closes an IF statement.

Syntax

The general form of an IF statement is,

```
IF (Test)
    ...statement block...
ELSE
    ...statement block...
ENDIF
```

Parameters

Test: *boolean*
Evaluates to true or false.

See Also

CASE
CASE CALL
SWITCH

IFPLATFORM-ENDIFPLATFORM

F

Purpose

A conditional statement that specifies a platform or platforms for which subsequent statements are compiled.

If the current platform matches one of the specified platforms, the statements between IFPLATFORM and ENDIFPLATFORM are compiled and executed.

ENDIFPLATFORM closes an IFPLATFORM statement.

Syntax

The general form of an IFPLATFORM statement is,

IFPLATFORM (*PlatformID; PlatformID; ... PlatformID*)
...statement block...
ENDIFPLATFORM

Parameters

PlatformID: *enum*
One or more platform identifiers. Separate multiple platforms with a semicolon. The identifiers for DOS and Windows are DOS and WIN. See platform documentation for other identifiers.

INCLUDE

F

Purpose

Specifies a macro file with executable statements, functions, and/or procedures. The functions and procedures can be called from another macro (see Macro File Libraries).

INCLUDE is a non-executable statement that can occur anywhere in a macro. INCLUDE macros are automatically compiles when the parent macro compiles. All statements in an INCLUDE macro are executed before macro execution returns to the parent macro. RETURN or QUIT statements in an INCLUDE macro end all macro execution.

Syntax

INCLUDE (*FileName*)

Parameters

FileName: *string*
The path and name of a macro file. You cannot substitute *string* with a variable.

See Also

USE
FUNCTION
PROCEDURE

INDIRECT

F

Purpose

Creates variable and label names out of a combination of character strings and/or numbers.

You can use INDIRECT wherever you use a variable. The actual variable must be declared and initialized before INDIRECT. You can use INDIRECT to call a LABEL, but not to create one.

```
State1 := "Utah"
x := INDIRECT("State" + 1) // x equals Utah
```

Variable example in shorthand notation

```
State1 := "Utah"
State2 := "Idaho"
State3 := "Arizona"

FORNEXT(Nmbr; 1; 3; 1)
     Type("State of " + INDIRECT("State" + Nmbr))
     HardReturn
ENDFOR
```

Explanation: Types three lines, State of Utah, State of Idaho, and State of Arizona.

LABEL example

```
FORNEXT(x; 1; 3; 1)
    CALL(INDIRECT("Lab" + x))
ENDFOR
QUIT

LABEL(Lab1)
    ...statement block...
    RETURN

LABEL(Lab2)
    ...statement block...
    RETURN

LABEL(Lab3)
    ...statement block...
    RETURN
```

Explanation: Call three Labels with a single CALL statement in a FORNEXT loop.

Syntax

ReturnValue := INDIRECT (*VariableName*)

Returns

The contents of *VariableName*.

Parameters

VariableName: *string*

Character strings and/or numbers are concatenated to form variable and Label names. You cannot use INDIRECT to create a subroutine name.

See Also

ASSIGN

INTEGER

Purpose

Returns the integer portion of a real number.

If *Value* does not contain an integer, 0 is returned. A negative real number is rounded up to the next integer value, and a positive real number is rounded down.

The following metasymbols specify a key or keystring in COACHFILTERADD and COACHANIMATE.

```
{VKnnn}                 //   nnn = ANSI character number

{Alt}
{Ctrl}
{Control}
{Shift}

{0}...{9}               // Digits
{A}...{Z}               // Alphabet
{F1}...{F16}            // Function keys

{NumLock}
{NumAdd}
{NumSubtract}
{NumMultiply}
{NumDivide}
{NumDecimal}
{NumEnter}
{Num0}...{Num9}         // Numpad numbers

{Left}
{Right}
{Up}
{Dn}
{Down}

{PgDn}
{PageDown}
{PgUp}
{PageUp}

{Bksp}
{Backspace}
{Break}                 // Cancel
{CapsLock}
{Clear}
{Del}
{Delete}
{End}
{Enter}
{Esc}
```

{Escape}
{Help} // VK Help key
{Home}
{Ins}
{Insert}
{Minus}
{Pause}
{ScrLock}
{ScrollLock}
{PrintScrn}
{PrintScreen}
{Space}
{Tab}

{LeftBrace} // "{"
{RightBrace} // "}"

{LeftButton} // Mouse clicks
{MiddleButton}
{RightButton}

Metasymbols are translated into keydown messages (WM_KEYDOWN or WM_SYSKEYDOWN) and keyup messages (WM_KEYUP or WM_SYSKEYUP).

Metasymbols are not case-sensitive. Spaces, plus (+) and minus (—) symbols are delimiters. For example, "A B C Enter" in COACHANIMATE sends "ABC[Enter]" to the application. A filter that specifies this keystring waits for the user to press "ABC[Enter]".

Plus (+) and minus (—) operators combine metasymbols. The + delays the keyup message of the first keystroke until the second keystroke is processed. For example, {Shift + Tab} presses and holds Shift while Tab is pressed and released. The Shift key is then released.

If a metasymbol does not follow +, then keys previously combined with + remain down until released. For example, {Shift+Control+} leaves Shift and Control down.

The — releases keys. For example, {—Control—Shift} releases Control, then Shift.

The + and — can be combined. For example, {Shift+Del—Del+Ins} cuts selected text to the Clipboard (Shift+Del), then pastes it at the insertion point (Shift+Ins).

```
{Home Shift+End}{Control+B End}{Enter}{Alt+F O} *.doc {Enter}{Escape}
{Ctrl+Home}{Shift+}{Ctrl+End Del}{—Shift}
```

Result:

1 Move to the beginning of the line and select the entire line.

2 Bold the line and move to the end of the line (cancelling the selection).

3 Create a new line.

4 Call the File Open dialog, list all *.DOC files, and then exit the dialog.

5 Go to the top of the document.

6 Hold down Shift.

7 Go to the end of the text and copy the selection to the Clipboard.

8 Release Shift.

Syntax

ReturnValue := INTEGER (*Value*)

Returns

An integer.

vInteger := INTEGER(1.5)
Result: vInteger equals 1

vResult := INTEGER(1.77 * 2)
Result: vResult equals 3

vZero := INTEGER(.7)
Result: vZero equals 0

vNegative := INTEGER(-1.77)
Result: vNegative equals -2

Parameters

Value: *numeric*
A real number.

See Also

FRACTION

KEYSTRING

Purpose

Specifies one or more keystrokes in COACHANIMATE and COACHFILTERADD. In COACHANIMATE, a keystring specifies keys to press during animation. In COACHFILTERADD, a keystring specifies keystrokes to filter.

LABEL

F

Purpose

Identifies a macro subroutine, which generally includes a statement block followed by RETURN or QUIT. See Subroutines.

You can pass a LABEL to a function or procedure. LABEL must be preceeded by @. For example,

```
CALL Macro(@Test)

PROCEDURE Macro(@x)
    CALL(x)
ENDPROC

LABEL(Test)
    ...statements...
    RETURN
```

LABEL is used by CALL, CASE, CASE CALL, GO, ONCANCEL, ONCANCEL CALL, ONERROR, ONERROR CALL, ONNOTFOUND, ONNOTFOUND CALL, ONDDEADVISE CALL, DDEEXECUTEEXT.

Syntax

LABEL (*label*)

Parameters

Label: *label*

The name of a label. It begins with a letter and consists of one or more letters or numbers.

See Also

CALL
CASE
CASE CALL
DDEEXECUTEEXT
GO
ONCANCEL
ONCANCEL CALL
ONDDEADVISE CALL
ONERROR
ONERROR CALL
ONNOTFOUND
ONNOTFOUND CALL

LOCAL

F

Purpose

Declares local variables and arrays, and assigns them to the local variable table.

Variables are LOCAL by default. The Local variable table is removed from memory when the macro ends.

Syntax

LOCAL (*VariableName; VariableName; ... VariableName*)

Parameters

VariableName: *variable*

One or more user-defined variables separated by a semicolon. Variables must begin with a letter and can include any other combination of letters or numbers. Separate multiple variables with a semicolon. Parentheses are optional. For example,

```
LOCAL VariableName; VariableName
LOCAL(VariableName; VariableName)
```

are both valid statements. The next example declares local arrays with 10 elements and 20 elements:

```
LOCAL FirstArray[10]; NextArray[20]
```

The last example simultaneously declares a variable and assigns a value:

```
LOCAL x := 2
```

or...

```
LOCAL x
x := 2
```

If you create two variables with the same name (for example, GLOBAL x and LOCAL x), the following statement

```
LOCAL x := 5
```

specifies which variable *x* is assigned the value 5.

See Also

DECLARE
DISCARD
EXISTS
GLOBAL
PERSIST
PERSISTALL

MENU

Purpose

Displays a menu of user-defined items.

MENU is used with statements such as CASE, CASE CALL, and IF to execute a statement that corresponds to a selected menu item. To dismiss a menu, press Alt or Esc.

Syntax

MENU (*MacroVar; MenuType; HorizPosition; VertiPosition;* {*MenuText; MenuText; ...*})

Parameters

MacroVar: *variable*
A number is returned in this variable, whether numbers or letters are used to reference menu items (see MenuType parameter). For numbers, the range is 1 to 9. For letters, the range is 1 to 26. A returns 1, B returns 2, and so forth. 0 is returned if Alt or Esc is pressed.

MenuType: *enum*
Determines whether a letter or number is displayed to the left of a menu item.

Digit!	Numbers (1-9)
Letter!	Alphabetic characters (A-Z)

HorizPosition: *numeric* (optional)
The number of pixels from the left side of the main window to the left side of the menu. To center the menu horizontally, leave this parameter blank.

VertiPosition: *numeric* (optional)
The number of pixels from the top of the main window to the top of the menu. To center the menu vertically, leave this parameter blank.

MenuText: *string*
The menu item text. You may list up to nine menu items referenced by numbers, or up to twenty-six referenced by letters. Enclose menu items in braces ({}) separated by a semicolon.

```
MENU(vSelection; Letter!; 50; 50; {"Option 1"; "Option 2"; "Option 3"})
```

To select an item: click it, select it with Tab and press Enter, or press the corresponding letter or number.

See Also

CASE
CASE CALL
IF

MENULIST

F

Purpose

Displays a menu of user-defined items.

This command is included for DOS compatibility. The Windows equivalent is MENU.

MENULIST is used with statements such as CASE, CASE CALL, and IF to execute a statement that corresponds to a selected menu item. To dismiss a menu, press Alt or Esc.

Syntax

MENULIST (*VariableName; {MenuChoice; MenuChoice; ...}*; *Title*; *HorizPosition*; *VertiPosition*)

Parameters

VariableName: *variable*
A number is returned in this variable, whether numbers or letters are used to reference menu items (see MenuChoice parameter). For numbers, the range is 1 to 9. For letters, the range is 1 to 26. A returns 1, B returns 2, and so forth. 0 is returned if Alt or Esc is pressed.

MenuChoice: *string*
The menu item text, enclosed in braces ({}) and separated by a semicolon.

```
MENULIST(vSelection; ; ; {"Option 1"; "Option 2"; "Option 3"})
```

To select an item: click it, select it with Tab and press Enter, or press the corresponding letter or number. Nine items or less are displayed with a number to the left of the menu item. More than nine are displayed with a letter. The range is 1 to 26.

Title: *string*
The text displayed in the title bar.

See Also

CASE
CASE CALL
IF
MENU

MESSAGEBOX

F

Purpose

Displays a message box.

Provides a limited set of buttons and icons, and an option to include up to ten different parameter strings in the message.

Syntax

MESSAGEBOX (*MacroVar; Title; Message; Style; ParameterData*)

Parameters

MacroVar: *variable*

The control value of the button that dismisses the message box is returned in this variable. The return button values are:

1 OK
2 Cancel
3 Abort
4 Retry
5 Ignore
6 Yes
7 No

If there is not enough memory to create the message box, 0 is returned.

Title: *string*

The title bar text.

Message: *string*

The message box text. Icons appear to the left of the message. Control buttons are centered below the message. If HasParameters! style is used, a caret (^) followed by a number inserts the corresponding ParameterData message in its place (see ParameterData parameter below). Use two carets (^^) to insert a caret as part of the message string. Use NTOC(0F90Ah) to insert a hard return code in the message string. For example,

```
MESSAGEBOX(vAns; "Title"; "First line" + NTOC(0F90Ah) + "Second line";
IconInformation!)
```

creates two lines above the control button. The next example creates three lines, including one blank line:

```
ASSIGN(HdReturn; NTOC(0F90Ah))
MESSAGEBOX(vAns; "Title"; "First line" + HdReturn + HdReturn + "Second line";
IconInformation!)
```

Style: *enum*

Message box styles. Type | between enumerations to combine styles. Select only one style from each group. Button enumerations are:

AbortRetryIgnore!	Abort, Retry, and Ignore
OK!	OK (default if no button is specified)
OKCancel!	OK and Cancel

RetryCancel!	Retry and Cancel
YesNo!	Yes and No
YesNoCancel!	Yes, No, and Cancel

Icon enumerations are:

IconNone!	No icon (default if no icon is specified)
IconExclamation!	Exclamation point in a yellow circle
IconInformation!	Lowercase "i" in a blue circle
IconAsterisk!	Lowercase "i" in a blue circle
IconQuestion!	Question mark in a green circle
IconStop!	Stop sign in a red circle
IconHand!	Stop sign in a red circle

Modality enumerations are:

ApplicationModal!	You can switch to another application, but you must dismiss the message box (click a control button) before the macro resumes. ApplicationModal! is the default if either SystemModal! or TaskModal! is not specified.
SystemModal!	You must dismiss the message box before the macro resumes, and before you can switch to another application.
TaskModal!	You can switch to another application, but you must dismiss the message box (click a control button) before the macro resumes.

Miscellaneous enumerations are:

Beep!	Beep when the message box is displayed.
HasParameters!	Notifies MessageBox box that a caret (^) followed by a number in the Message parameter is to be replaced by the corresponding character string in the ParameterData parameter.

ParameterData: *string*
The text for the Message parameter (numbering begins with 0). Up to ten parameters separated by semicolons are allowed. For example,

```
v0 := "Yes to continue, or"
v1 := "No to quit."
MessageBox(x; "Title"; "Select ^0 ^1"; YesNo! | IconStop! | HasParameters!; {v0; v1})
```

fills the Message parameter with "Select Yes to continue, or No to quit.". If HasParameters! is not used, *Message* contains "Select ^0 ^1".

See Also
PROMPT

MMPLAY

Purpose

Plays a sound file.

Plays a wave audio file, an Audio-Visual Interleave (AVI) movie file, or a Musical Instrument Digitial Interface (MIDI) file, or speaks an ASCII text file (see MMSPEAK). Microsoft Video drivers must be installed to play AVI files. For WAV files, install Windows Audio drivers. MIDI files require Windows MIDI drivers. Speech requires text-to-speech drivers like those shipped with some sound boards.

Syntax

MMPLAY (*Filename*)

Parameters

Filename: *string*
The path and name of a sound file. The appropriate driver for each file type must be installed.

See Also

MMSPEAK
MMSTOPSPEECH
MMSPEAKCLIPBOARD

MMSPEAK

Purpose
Speaks an ASCII file.

Syntax
MMSPEAK (*Filename*)

Parameters
Filename: *string*
The path and name of an ASCII file. A text-to-speech driver must be installed.

See Also
MMPLAY
MMSTOPSPEECH
MMSPEAKCLIPBOARD

MMSPEAKCLIPBOARD

Purpose

Speaks an ASCII file saved to the Clipboard.

A text-to-speech driver must be installed.

Syntax

MMSPEAKCLIPBOARD ()

See Also

MMPLAY
MMSPEAK
MMSTOPSPEECH

MMSTOPSPEECH

Purpose

Stops a speech (text) file started with MMPLAY (ASCII file), MMSPEAK, or MMSPEAKCLIPBOARD.

After a text file begins to speak, control immediately returns to the macro. Use MMSTOPSPEECH to stop speaking a text file.

Syntax

MMSTOPSPEECH ()

See Also

MMPLAY
MMSPEAK
MMSPEAKCLIPBOARD

MOUSE STRING

Purpose

Represents a mouse event or action. COACHANIMATE specifies the animated mouse actions. COACHFILTERADD specifies the filtered mouse actions.

Syntax

The syntax for mouse events for COACHANIMATE is:

[Ctrl|Control] [Shift] [Alt]
[Left|Middle|Right]
Click|DoubleClick|Press|Release|Move
[To|In|Inside|On NamedRegion]
[At (x,x)|(x,x,x,x)]

"Outside," "OutOf," and "Off" are supported in COACHFILTERADD and "Move" is not supported.

[Ctrl|Control] [Shift] [Alt]
[Left|Middle|Right]
Click|DoubleClick|Press|Release
[To|In|Inside|On|Outside|OutOf|Off NamedRegion]
[At (x,x)|(x,x,x,x)]

Arguments are not case-sensitive. Brackets ([]) enclose optional values. A vertical bar | separates values when only one can be chosen. A space separates arguments.

MOUSE STRING Arguments

Keys (optional) ([Ctrl|Control] [Shift] [Alt])
Specifies keys to press with the mouse event, such as holding down Shift and Control when the mouse is clicked. These values can occur in any order.

Button (optional) ([Left|Middle|Right])
Indicates whether to use the left, middle, or right button. Left is the default.

Action (Click|DoubleClick|Press|Release|Move)
Indicates how to click or press the mouse button, or whether to move the mouse pointer.

Argument	Meaning
Click	One click.
DoubleClick	Double-click.
Press	Press and hold button down.
Release	Release button after it is pressed. Use Press with Release for *drag* (use two mouse strings).
Move	Move pointer to specified location. Valid only in COACHANIMATE.

Relative Location ([To|In|Inside|On|Outside|OutOf|Off])
Specifies a screen position for a mouse event.

Argument	Meaning

To	If the Action argument is Move, To moves the pointer to the named region. Otherwise, To means In.
In	Mouse action occurs within the specified named region (see Named Region, below).
Inside	Same as In.
On	Same as In.
Outside	Mouse action occurs outside the specified named region. Supported only in COACHFILTERADD.
OutOf	Same as Outside. Supported only in COACHFILTERADD.
Off	Same as Outside. Supported only in COACHFILTERADD.

Named Region ([NamedRegion])
Specifies a window or location where the mouse action occurs. Required only if a relative location is specified. For example, you may want to specify a mouse click in the Open button of the Open Dialog. The named region specifies a name for the Open button.

Named regions are defined by the application. The region consists of the application name, followed by a period (.), followed by additional words that narrow the named region to the appropriate window. For example, the named region of the Open button on the Open dialog of WordPerfect is: WordPerfect.FileOpenDlg.Open.

Use the WordPerfect Help system to find a region name. First edit your BIF file to enable Help Information. The BIF editor, BIFED20.EXE, should be in the WPC20 directory.

1 Exit WordPerfect and go to Program Manager.
2 Choose File, then Run.
3 In the BIF Edit dialog box, choose File, then Open.
4 Select the desired file.
5 Select WP Shared Code.
6 Choose Insert to add a new section.
7 Type Help in the Section Name entry box.
8 Type Help Information in the Item entry box. The Item Type is Boolean.
9 Select True for the Value.
10 Choose File, then Save.
11 Choose File, then Exit.
12 Start WordPerfect.

To find a named region:

1 Press Shift+F1.
2 Click the desired menu item or dialog box.

A Help Information screen is displayed. The Application and Keyword items display the named region.

Help Information for dialog boxes returns application and dialog box names, but not named regions for controls on the dialog. See the list of dialog box named regions in the Appendix for control named regions.

Buttons are named for the text on them, plus Bttn. For example, the Help button in the HeadersFooters dialog box is WordPerfect.HeadersFooters.HelpBttn. Named region names are case-sensitive.

Narrowed Location (optional) ([At (x,x)|(x,x,x,x)])
Specifies an exact point (x,x), or rectangle (x,x,x,x) where a mouse event occurs. Point

and rectangle positions are relative to the window specified in *Named Region*. Pixels are the default unit of measure. For example, in COACHANIMATE,

```
Move To WordPerfect.FileOpenDlg.Open At (10,5)
```

moves the pointer to a point 10 pixels right and 5 pixels down from the upper left corner of the Open button.

You can specify the percentage from the upper left corner of the window by placing a percent sign % after values. For example,

```
Move To WordPerfect.FileOpenDlg.Open At (50%, 50%)
```

moves the pointer to the center of the Open button.

To specify dialog units, place the letter d after values.

Examples

Three mouse string examples follow. Named regions do not represent actual WordPerfect names.

```
Shift Right DblClick
```

Result: Shift is held down while the right mouse button is double-clicked (the mouse pointer may be anywhere on the screen).

```
Move at (50%, 50%)
```

Result: This mouse string is invalid in COACHFILTERADD. When this string is specified in COACHANIMATE, the mouse pointer moves to the center of the display.

```
Press on WordPerfect.Ruler.Tab1
Release on WordPerfect.EditWindow
```

Result: These mouse strings must be specified in two consecutive commands. The first tab setting on the ruler is dragged off the ruler (deleted).

NEST

F

Purpose

Calls (starts) a nested macro.

This command is included for DOS compatibility. The Windows equivalent is RUN.

A nested macro starts immediately when called. When a nested macro ends, the first statement following NEST is executed. A macro must be compiled before it is called by NEST.

Syntax

NEST (*MacroFileName*; {*Parameter*})

Parameters

MacroFileName: *string*
The path and name of a compiled macro.

Parameter: *any* (optional)
Enclose multiple parameters in braces ({}), separated by a semicolon. For example: NEST ("macro"; {"a"; "b"; "c"}). Parameter values are passed to a special array variable named MacroArgs.

See Also

CHAIN
RUN

NEWDEFAULT

Purpose

Specifies a macro's new default application.

The compiler verifies that each product command is valid for the default application. Invalid product commands produce a syntax error during compilation.

Product commands to the default application do not require a product prefix. A product command to a non-default application, without a product prefix, creates a compile-time syntax error.

Syntax

NEWDEFAULT (*ProductPrefix*)

Parameters

ProductPrefix: *string*
A unique identifier that matches the ProductPrefix parameter of an APPLICATION statement.

See Also

APPLICATION
ENDAPP

NEXT

Purpose

Advances a loop iteration before the end of the loop.

Use NEXT inside a conditional statement such as IF or SWITCH in a loop to advance a loop iteration when a specified condition is met. Statements after NEXT are ignored.

Syntax

NEXT

See Also

BREAK
FOR
FOREACH
FORNEXT
REPEAT
WHILE

NOTFOUND

Purpose

Determines how a macro responds to a Not Found condition.

A Not Found condition is generally the result of a failed search. Create a Not Found condition with ASSERT(NotFoundCondition!).

Syntax

NOTFOUND (*State*)

Parameters

State: *enum*

Specifies the Not Found state. The default is NOTFOUND(On!).

Off!	Overrides a Not Found condition.
On!	Stops a macro unless preceded by ONNOTFOUND, which directs macro execution to a specified LABEL.

See Also

ASSERT
CANCEL
ERROR
LABEL
ONNOTFOUND

NTOC

Purpose
Converts a decimal value to its character equivalent.

Syntax
ReturnValue := NTOC (*CharSet; CharValue*)

Returns
The character equivalent of a decimal value.

Parameters
CharSet: *numeric* (optional)
The number of a WordPerfect character set. If this parameter is not used, the default is 0 (ASCII set).

CharValue: *numeric*
The number of a character in a WordPerfect character set.

See Also
CTON

NUMSTR

Purpose
Converts a number to a character string of numbers.

Syntax
ReturnValue := NUMSTR (*Value; RightDigits; LeftDigits*)

Returns
A character string of numbers.

```
vText := NUMSTR(999; 0) // vText = "999"
vText := NUMSTR(3+7; 0) // vText = "10"
```

Parameters
Value: *numeric*
A number or expression that results in a number.

RightDigits: *numeric* (optional)
The number of digits to the right of the decimal returned as a character string. The range is 0-16. The default is 6. If the value of this parameter is greater than the number of decimal places, NUMSTR rounds up the entire expression. The decimal point returned by NUMSTR is defined by the sDecimal setting in the [intl] section of the Windows WIN.INI file.

LeftDigits: *numeric* (optional)
The number of digits to the left of the decimal returned as a character string. Larger numbers are expressed in exponential notation.

See Also
STRNUM

ONCANCEL

Purpose

Executes a LABEL statement when a Cancel condition occurs.

A macro with more than one ONCANCEL executes the last ONCANCEL before a Cancel condition occurs. A Cancel condition stops a macro unless preceded by ONCANCEL.

Syntax

ONCANCEL (*label*)

Parameters

Label: *label* (optional)
A LABEL to execute when a Cancel condition occurs. If a label is not specified, the macro halts when a Cancel condition occurs. For example: ONCANCEL().

See Also

ASSERT
INDIRECT
LABEL
ONCANCEL CALL
ONERROR
ONNOTFOUND

ONCANCEL CALL

F

Purpose

Calls a LABEL-RETURN statement when a Cancel condition occurs. RETURN directs macro execution to the first statement after the Cancel condition.

A macro with more than one ONCANCEL CALL executes the last ONCANCEL CALL before a Cancel condition occurs.

F A Cancel condition stops a macro unless preceded by ONCANCEL CALL.

Syntax

ONCANCEL CALL (*label*)

Parameters

Label: *label*
A LABEL to call when a Cancel condition occurs.

See Also

ASSERT
INDIRECT
LABEL
ONCANCEL
ONCANCEL CALL
ONDDEADVISE CALL
ONERROR
ONERROR CALL
ONNOTFOUND
ONNOTFOUND CALL
RETURN

ONDDEADVISE CALL

Purpose

Establishes a link with an application and tests for changes in the value of the DDEREQUEST ItemName parameter.

Syntax

ONDDEADVISE CALL (*MacroVar; ConversationID; ItemName; Label*)

Parameters

MacroVar: *variable*
0 is returned in this variable if the ONDDEADVISE CALL link is successfully established, or a number greater than 0 if a link is not established.

ConversationID: *numeric*
The value of the DDEREQUEST ConversationID parameter.

ItemName: *string*
The value of the DDEREQUEST ItemName parameter.

Label: *label*
The LABEL to call when the value of ItemName changes.

See Also

DDEINITIATE
DDEREQUEST
INDIRECT
ONCANCEL CALL
ONERROR CALL
ONNOTFOUND CALL

ONERROR

Purpose

Executes a LABEL statement when an Error condition occurs.

A macro with more than one ONERROR statement executes the last ONERROR before an Error condition occurs. An Error condition stops a macro unless preceded by ONERROR.

Syntax

ONERROR (*label*)

Parameters

Label: *label* (optional)
A LABEL to execute when an Error condition occurs. If a label is not specified, the macro halts when an Error condition occurs. For example: ONERROR().

See Also

ASSERT
INDIRECT
LABEL
ONCANCEL
ONERROR CALL
ONNOTFOUND

ONERROR CALL

F

Purpose

Calls a LABEL-RETURN statement when an Error condition occurs. RETURN directs macro execution to the first statement after the Error condition.

A macro with more than one ONERROR CALL statement executes the last ONERROR CALL before an Error condition occurs.

F An Error condition stops a macro unless preceded by ONERROR CALL.

Syntax

ONERROR CALL (*label*)

Parameters

Label: *label*
A LABEL to call when an Error condition occurs.

See Also

ASSERT
INDIRECT
LABEL
ONCANCEL
ONCANCEL CALL
ONERROR
ONNOTFOUND
ONNOTFOUND CALL
RETURN

ONNOTFOUND CALL

F

Purpose

Calls a LABEL-RETURN statement when a Not Found condition occurs. RETURN directs macro execution to the first statement after the Not Found condition.

A macro with more than one ONNOTFOUND CALL statement executes the last ONNOTFOUND CALL before a Not Found condition occurs.

F A Not Found condition stops a macro unless preceded by ONNOTFOUND CALL.

Syntax

ONNOTFOUND CALL (*label*)

Parameters

Label: *label*
A LABEL to call when a Not Found condition occurs.

See Also

ASSERT
INDIRECT
LABEL
ONCANCEL
ONCANCEL CALL
ONERROR
ONERROR CALL
ONNOTFOUND
RETURN

ONNOTFOUND

Purpose

Executes a LABEL statement when a Not Found condition occurs.

A macro with more than one ONNOTFOUND statement executes the last ONNOTFOUND before a Not Found condition occurs. A Not Found condition stops a macro unless preceded by ONNOTFOUND.

Syntax

ONNOTFOUND (*label*)

Parameters

Label: *label* (optional)
A LABEL to execute when a Not Found condition occurs. If a label is not specified, the macro halts when a Not Found condition occurs. For example: ONNOTFOUND().

See Also

ASSERT
INDIRECT
LABEL
ONCANCEL
ONERROR
ONNOTFOUND CALL

OPENFILE

Purpose

Opens a file for reading and writing text data.

Close file when no longer needed, or system resources will be lost. The file is closed automatically when the macro ends. See CLOSEFILE.

Syntax

numeric OPENFILE (Name: *string*; AccessMode: *enum*; ShareMode: *enum*; DataType: *enum*)

Return Value

numeric File ID, or a negative number if error occurs. The following file I/O (input/output) commands require a file ID:

CLOSEFILE
FILEISEOF
FILEPOSITION
FILEREAD
FILETRUNCATE
FILEWRITE

Parameters

Name: *string*
Include full path. If path not included, file searched for in the following directories: current, Windows, Windows system, current program directory, DOS path, network (mapped).

AccessMode: *enum* (optional)
Read!
ReadWrite!
Write!
WriteNew!
Append!
Exists!

Default: Read!. Write! overrides existing data if position marker not at end of file (see FILEPOSITION). WriteNew! deletes existing data and creates a new file. Exists! opens and closes file to test if file exists. If file does not exist: Read! and ReadWrite! cause macro to fail; Write!, WriteNew!, and Append! create a new file.

ShareMode: *enum* (optional)
None!
Compatibility!
Exclusive!
DenyNone!
DenyRead!
DenyWrite!

Default: None!. Compatibility! allows one or more macros to open a file any number of

times. Exclusive! denies read and write access to more than one macro. DenyNone! does not deny other macros read and write access. DenyRead! denies other macros read access. DenyWrite! denies other macros write access.

DataType: *enum* (optional)
AnsiText!
OEMText!
WPText!

Default: AnsiText!. WPText! reads or writes WP character strings. It does not create a WP document.

See Also
FILEERROR

PAUSE

Purpose

Pauses a macro, returning computer control to the user.

To resume macro execution, press Enter, or choose Tools, Macro, Pause. To stop a macro, choose Tools, Macro, Stop. When PAUSE follows a PROMPT statement, choose OK to resume macro execution or Cancel to create a Cancel condition.

Syntax

PAUSE

See Also

CANCEL
ONCANCEL
PROMPT

PERSIST

F

Purpose

Declares Shared Code variables and arrays, and assigns them to the persist variable table.

PERSIST variables are available to any Shared Code application while Shared Code is running.

Syntax

PERSIST (*VariableName; VariableName; ... VariableName*)

Parameters

VariableName: *variable*

One or more user-defined variables separated by a semicolon. Variables must begin with a letter and can include any other combination of letters or numbers. Separate multiple variables with a semicolon. Parentheses are optional. For example,

```
PERSIST VariableName; VariableName
PERSIST(VariableName; VariableName)
```

are both valid statements. The next example declares persist arrays with 5 elements and 15 elements:

```
PERSIST FirstArray[5]; NextArray[15]
```

The last example simultaneously declares a variable and assigns a value:

```
PERSIST x := 2
```

or...

```
PERSIST x
x := 2
```

If you create two variables with the same name (for example, GLOBAL x and PERSIST x), the following statement

```
PERSIST x := 5
```

specifies which variable *x* is assigned the value 5.

See Also

DECLARE
DISCARD
EXISTS
GLOBAL
LOCAL
PERSISTALL

PERSISTALL

F

Purpose

Assigns a macro's variables and arrays to the Persist variable table.

PERSIST variables are available to any Shared Code application while Shared Code is running.

Only variables assigned values after PERSISTALL are Persist variables. Variables assigned values before PERSISTALL, and variables declared LOCAL or GLOBAL, are not Persist variables. For example, only var6 and var7 are Persist variables:

```
APPLICATION(A1; "WordPerfect"; Default; "US")
LOCAL var1; var2
ASSIGN(var3; 0)
GLOBAL var4
PERSIST var7
ASSIGN(var7; "Persist Variable")

PERSISTALL

LOCAL var5
ASSIGN(var4; 2)
ASSIGN(var6; "Persist Variable")
```

Syntax

PERSISTALL

See Also

DISCARD
GLOBAL
LOCAL
PERSIST

PROCEDURE-ENDPROC

F

Purpose

Identifies a macro subroutine that can receive one or more values from a calling statement (see Calling Statements).

Procedures contain one or more statements that execute when the procedure is called. Unlike LABEL subroutines, procedures do not execute unless called. A calling statement consists of the procedure's name, and can have one or more parameters that contain values passed to the procedure. RETURN, ENDPROC, or ENDPROCEDURE direct macro execution to the statement that follows the procedure's caller.

Address Mode

To change the value of a variable passed to a procedure, precede the calling statement parameter and its corresponding procedure parameter with an ampersand (&). In the following example, the value of x is 5 after calling Test:

```
x := 5
Test(x) // calling statement

PROCEDURE Test(z)
    z := z + 5
ENDPROC
```

In the next example, the value of x is 10 after calling Test:

```
x := 5
Test(&x) // calling statement

PROCEDURE Test(&z)
    z := z + 5
ENDPROC
```

The ampersand before variable x means the variable's address (location in memory) is passed to the procedure, not the variable's value. Changes made at the address of x are made to the contents of x.

Syntax

The general form of a PROCEDURE statement is,

PROCEDURE *Name* (*Parameter; Parameter; ... Parameter*)
  ...statement block...
ENDPROC

The syntax accepts ENDPROC or ENDPROCEDURE.

Parameters

Name: *label*
The name of the procedure. It begins with a letter and consists of one or more letters or numbers.

Parameter: *variable*

Receives a value from a calling statement (see Calling Statements). If an ampersand precedes the calling statement variable, an ampersand must precede the corresponding procedure variable (see Address Mode above). Multiple variables are separated by a semicolon.

See Also

CALL
FUNCTION
LABEL
PROCEDURE PROTOTYPE
RETURN
USE

PROCEDURE PROTOTYPE

F

Purpose

Verifies the syntax of a PROCEDURE statement (see PROCEDURE).

If the syntax of a procedure contained in a USE macro file is incorrect and the procedure is called from the main macro, you get a run-time error but not a compile-time error (see Macro Errors). Using PROCEDURE PROTOTYPE at the start of a main macro ensures that you get a compile-time error.

Syntax

PROCEDURE PROTOTYPE *Name* (*Parameter; Parameter; ... Parameter*)

Parameters

Name: *label*

The name of the procedure. It begins with a letter and consists of one or more letters or numbers.

Parameter: *variable*

Receives a value from a calling statement (see Calling Statements). If an ampersand precedes the calling statement variable, an ampersand must precede the corresponding procedure variable (see Address Mode under PROCEDURE). Multiple variables are separated by a semicolon.

PROMPT

F

Purpose

Displays a message box with OK and Cancel button controls.

PAUSE following PROMPT displays a PROMPT message box until the user selects OK or Cancel. OK removes the prompt and resumes macro execution. Cancel removes the prompt and creates a Cancel condition (see ONCANCEL).

If PROMPT is not followed by PAUSE, the message box is displayed until an ENDPROMPT occurs or the macro ends.

F You can display only one message box at a time. Calling a second message box replaces the first.

Syntax

The general form of a PROMPT statement is,

PROMPT (*Title; Prompt; Icon; HorizPosition; VertiPosition*)
PAUSE

A PROMPT statement can have another form,

PROMPT (*Title; Prompt; Icon; HorizPosition; VertiPosition*)

...other statements...

ENDPROMPT

Parameters

Title: *string*
The text displayed in the title bar.

Prompt: *string* (optional)
The text displayed in the prompt box. If not used, the title is displayed as the prompt and "WordPerfect Macro Facility" as the title.

Icon: *numeric* (optional)
The number of an icon displayed to the left of *Prompt*. The default value is zero. The values are:

0 (no icon)
1 (stop sign)
2 (question mark)
3 (exclamation point)
4 (information icon)

HorizPosition: *numeric* (optional)
The number of pixels from the left side of the main window to the left side of the message box. If you omit this parameter, the prompt is centered horizontally in the main window.

VertiPosition: *numeric* (optional)
The number of pixels from the top of the main window to the top of the message box. If you omit this parameter, the prompt is centered vertically in the main window.

See Also

CANCEL
MESSAGEBOX
ONCANCEL
PAUSE

QUIT

Purpose

Ends a macro.

CHAIN does not execute if QUIT occurs before CHAIN.

Syntax

QUIT

See Also

CHAIN

REGIONADDLISTITEM

Purpose
Adds an item to a list box, combination box, or popup button.

Syntax
REGIONADDLISTITEM (NamedRgn: *string*; Item: *string*)

Parameters
NamedRgn: *string*
Dialog box and control IDs. Enclose IDs in double quotation marks, separated by a period.

Item: *string*
A list item.

See Also
DIALOGADDLISTITEM
DIALOGADDLISTBOX
DIALOGADDCOMBOBOX
DIALOGADDPOPUPBUTTON
DIALOGDEFINE
DIALOGSHOW
REGIONREMOVELISTITEM
REGIONRESETLIST
REGIONSELECTLISTITEM

REGIONENABLEWINDOW

Purpose

Enables or disables mouse and keyboard input to a dialog box or control.

Syntax

REGIONENABLEWINDOW (NamedRgn: *string*; State: *enum*)

Parameters

NamedRgn: *string*

Dialog box and control IDs. Enclose IDs in double quotation marks, separated by a period. If a control ID is not specified, State affects the entire dialog box.

State: *enum*

Mouse and keyboard input state.

0 Disable!
1 Enable!

See Also

DIALOGDEFINE
DIALOGSHOW
REGIONSHOWWINDOW

REGIONGETCHECK

F

Purpose
Reports the state of a check box control.

Syntax
REGIONGETCHECK (MacroVar: *variable*; NamedRgn: *string*)

Parameters
MacroVar: *variable*
Output variable is assigned 1 if the check box is selected, and 0 if not.

NamedRgn: *string*
Dialog box and check box control IDs. Enclose IDs in double quotation marks, separated by a period.

See Also
DIALOGADDCHECKBOX
DIALOGDEFINE
DIALOGSHOW
REGIONSETCHECK

REGIONGETSELECTEDTEXT

F

Purpose

Assigns selected text to an output variable.

The selected text is in a control such as a list box, combination box, or counter.

Syntax

REGIONGETSELECTEDTEXT (MacroVar: *variable*; NamedRgn: *string*)

Parameters

MacroVar: *variable*
Output variable contains the selected text.

NamedRgn: *string*
Dialog box and control IDs. Enclose IDs in double quotation marks, separated by a period.

See Also

DIALOGDEFINE
DIALOGSHOW
DIALOGADDLISTBOX
DIALOGADDCOMBOBOX
DIALOGADDCOUNTER

REGIONGETWINDOWTEXT

Purpose
Gets caption bar or static text.

Syntax
REGIONGETWINDOWTEXT (MacroVar: *variable*; NamedRgn: *string*)

Parameters
MacroVar: *variable*
Output variable contains the text.

NamedRgn: *string*
Dialog box and control IDs. Enclose IDs in double quotation marks, separated by a period. If a control ID is not specified, the command gets the caption bar text.

See Also
DIALOGDEFINE
DIALOGSHOW
REGIONSETWINDOWTEXT

REGIONISVISIBLE

F

Purpose

Determines the visibility state of a region or window.

Syntax

REGIONISVISIBLE (MacroVar: *variable*; NamedRgn: *string*)

Parameters

MacroVar: *variable*

The visibility state is returned in this variable: a nonzero number if the window is visible, or 0 if not. Window's WS_VISIBLE flag determines the return value, which can be nonzero even if the window is only hidden by other windows. For more information, see IsWindowVisible in *Programmer's Reference, Volume 2: Functions* (Windows 3.1 documentation), page 558.

NamedRgn: *string*

Dialog box and control IDs. Enclose IDs in double quotation marks, separated by a period.

See Also

REGIONSHOWWINDOW

REGIONMOVEWINDOW

F

Purpose

Moves and/or resizes a dialog box or control.

Syntax

REGIONMOVEWINDOW (NamedRgn: *string*; Left: *numeric*; Top: *numeric*; Width: *numeric*; Height: *numeric*)

Parameters

NamedRgn: *string*
Dialog box and control IDs. Enclose IDs in double quotation marks, separated by a period. If a control ID is not specified, the dialog box is moved and/or resized.

Left: *numeric*
Number of dialog units from the left side of the screen to the left side of the dialog box; or, from the left side of the dialog box to the left side of the control.

Top: *numeric*
Number of dialog units from the top of the screen to the top of the dialog box; or, from the top of the dialog box to the top of the control.

Width: *numeric*
Width of the dialog box or control in dialog units.

Height: *numeric*
Height of the dialog box or control in dialog units.

See Also

DIALOGDEFINE

REGIONREMOVELISTITEM

Purpose

Removes a list item from a list box, combination box, or popup button.

Syntax

REGIONREMOVELISTITEM (NamedRgn: *string*; Item: *string*)

Parameters

NamedRgn: *string*
Dialog box and control IDs. Enclose IDs in double quotation marks, separated by a period.

Item: *string*
A list item.

See Also

DIALOGADDLISTITEM
DIALOGADDLISTBOX
DIALOGADDCOMBOBOX
DIALOGADDPOPUPBUTTON
DIALOGDEFINE
DIALOGSHOW
REGIONADDLISTITEM
REGIONRESETLIST
REGIONSELECTLISTITEM

REGIONRESETLIST

Purpose
Clears the contents of a list box, combination box, or popup button.

Syntax
REGIONRESETLIST (NamedRgn: *string*)

Parameters
NamedRgn: *string*
Dialog box and control IDs. Enclose IDs in double quotation marks, separated by a period.

See Also
DIALOGADDLISTITEM
DIALOGADDLISTBOX
DIALOGADDCOMBOBOX
DIALOGADDPOPUPBUTTON
DIALOGDEFINE
DIALOGSHOW
REGIONADDLISTITEM
REGIONREMOVELISTITEM
REGIONSELECTLISTITEM

REGIONSELECTLISTITEM

F

Purpose

Selects an item in a list box, combination box, or popup button.

Syntax

REGIONSELECTLISTITEM (NamedRgn: *string*; Item: *string*)

Parameters

NamedRgn: *string*
Dialog box and control IDs. Enclose IDs in double quotation marks, separated by a period.

Item: *string*
A list item.

See Also

DIALOGADDLISTITEM
DIALOGADDLISTBOX
DIALOGADDCOMBOBOX
DIALOGADDPOPUPBUTTON
DIALOGDEFINE
DIALOGSHOW
REGIONADDLISTITEM
REGIONREMOVELISTITEM
REGIONRESETLIST

REGIONSETCHECK

Purpose
Selects/deselects a check box.

Syntax
REGIONSETCHECK (NamedRgn: *string*; State: *enum*)

Parameters
NamedRgn: *string*
Dialog box and check box control IDs. Enclose IDs in double quotation marks, separated by a period.

State: *enum*
Sets the state of a check box control.
0 Unchecked!
1 Check!

See Also
DIALOGADDCHECKBOX
DIALOGDEFINE
DIALOGSHOW
REGIONGETCHECK

REGIONSETFOCUS

Purpose
Gives input focus to a specified control.

Syntax
REGIONSETFOCUS (NamedRgn: *string*)

Parameters
NamedRgn: *string*
Dialog box and control IDs. Enclose IDs in double quotation marks, separated by a period.

See Also
DIALOGSHOW

REGIONSETWINDOWTEXT

F

Purpose
Replaces caption bar or static text.

Syntax
REGIONSETWINDOWTEXT (NamedRgn: *string*; Item: *string*)

Parameters
NamedRgn: *string*
Dialog box and control IDs. Enclose IDs in double quotation marks, separated by a period. If a control ID is not specified, the command replaces the caption bar text.

Item: *string*
Replacement text.

See Also
DIALOGDEFINE
DIALOGSHOW
REGIONGETWINDOWTEXT

REGIONSHOWWINDOW

F

Purpose
Shows/hides a dialog box or control.

Syntax
REGIONSHOWWINDOW (NamedRgn: *string*; State: *enum*)

Parameters
NamedRgn: *string*
Dialog box and control IDs. Enclose IDs in double quotation marks, separated by a period. If you do not use the control ID, the command hides the dialog box.

State: *enum*
The display state.
- Hide!
- Show!
- ShowMinimized!
- ShowMaximized!
- Maximize!
- ShowNoActivate!
- Normal!
- ShowNormal!
- Minimize!
- ShowMinNoActive!
- ShowNA!
- ShowRestore!

See Also
DIALOGSHOW
REGIONISVISIBLE

RENAMEDIRECTORY

Purpose
Renames a directory.

Syntax
boolean RENAMEDIRECTORY (OldDirectoryName: *string*; NewDirectoryName: *string*; Prompts: *enum*)

Return Value
boolean True if successful, False if not.

Parameters
OldDirectoryName: *string*
Include full path.

NewDirectoryName: *string*
Include full path.

Prompts: *enum* (optional)
Prompt if directory does not exist.
NoPrompts!
Prompts!

See Also
CREATEDIRECTORY
DELETEDIRECTORY
DOESDIRECTORYEXIST
GETCURRENTDIRECTORY
SETCURRENTDIRECTORY

RENAMEFILE

F

Purpose
Renames and/or moves a file.

Syntax
boolean RENAMEFILE (OldFilename: *string*; NewFilename: *string*; Prompts: *enum*)

Return Value
boolean True if successful, False if not.

Parameters
OldFilename: *string*
Include full path.

NewFilename: *string*
Include full path.

Prompts: *enum*
Prompts if old file does not exist, or path is incorrect.
NoPrompts!
Prompts!

See Also
COPYFILE
DELETEFILE
DOESFILEEXIST

REPEAT-UNTIL

Purpose

A loop statement that executes until the expression at the bottom of the loop is true (see Loop Statements).

The loop executes at least once, because it is not tested until the bottom of the loop. When *Test* is true, the first statement after UNTIL is executed.

Syntax

The general form of a REPEAT statement is,

```
REPEAT
    ...statement block...
UNTIL (Test)
```

Parameters

Test: *boolean*
Evaluates to true or false.

See Also

FOR
FOREACH
FORNEXT
WHILE

RETURN

F

Purpose

Ends LABEL, FUNCTION, and PROCEDURE subroutines, or a macro (see RUN), and then directs macro execution to the statement that follows the subroutine or macro's caller, or creates a Cancel, Error, or Not Found condition.

RETURN generally ends a LABEL statement called by statements such as CALL or CASE CALL.

```
CALL(StartMacro)
CALL(QuitMacro)

LABEL(StartMacro)
    ... statement block...
    RETURN // directs macro execution to CALL(QuitMacro)

LABEL(QuitMacro)
    QUIT
```

If there is no caller to return to, and the macro containing RETURN is nested (called by another macro), RETURN directs macro execution to the statement that follows the macro's caller (see RUN). RETURN ends a macro if there is no caller to return to, and the macro containing the RETURN statement is not nested.

Syntax

RETURN (*Condition; Value*)

Parameters

Condition: *enum* (optional)
Creates a Cancel, Error, or Not Found condition (see ASSERT). *Condition* stops a macro unless preceded by ONCANCEL, ONERROR, or ONNOTFOUND, which direct macro execution to a specified LABEL. *Condition* has no effect when preceded by CANCEL(Off!), ERROR(Off!), or NOTFOUND(Off!)

The enumerations (return types) are:

CancelCondition!	Stops a macro unless preceded by ONCANCEL.
ErrorCondition!	Stops a macro unless preceded by ONERROR.
NotFoundCondition!	Stops a macro unless preceded by ONNOTFOUND.

Value: *any* (optional)
Returns the result of a function operation (see FUNCTION).

See Also

ASSERT
CALL
CASE CALL
FUNCTION
LABEL
PROCEDURE
ONCANCEL
ONERROR

ONNOTFOUND
RUN

RUN

F

Purpose

Calls (starts) a nested macro.

A nested macro starts immediately when it is called. When a nested macro ends, control returns to the calling macro. If QUIT ends a nested macro, control does not return to the calling macro. A macro must be compiled before it is called by RUN.

Syntax

RUN (*MacroFile*; {*Parameter*})

Parameters

MacroFile: *string*
The path and name of a compiled macro.

Parameter: *any* (optional)
Enclose multiple parameters in braces ({}), separated by a semicolon. For example: RUN ("macro"; {"a"; "b"; "c"}) For example: RUN ("macro"; {"a"; "b"; "c"}). Parameter values are passed to a special array variable named MacroArgs.

See Also

CHAIN

SENDKEYS WAIT

Purpose

Sends keystrokes to the current application. WAIT is optional (see Description below)

Keys such as Alt or F1 must be enclosed in braces to specify a single keystroke. For example,

```
SENDKEYS("{Alt}LLH")
```

sends four keystrokes (Alt, L, L, and H) which opens the Line Height dialog box. The following commands are equivalent to the previous example:

```
SENDKEYS("{Alt + L + L + H}")
SENDKEYS("{Alt}{L}{L}{H}")
```

If you combine keystrokes in a single set of braces, you must separate them with a plus operator. Otherwise, the result is unpredictable. Enclosing single character keystrokes in braces is optional.

Assign frequently used keystrokes to a variable. For example,

```
ASSIGN(KS_MarginsDlg; "{Alt}LM")
SENDKEYS(KS_MarginsDlg)
```

opens the Margins dialog box, and is equivalent to the programming command FormatMarginsDlg. Do not use variable names that are identical to a macro command name.

A minus operator releases a keystroke. For example,

```
SENDKEYS("{Shift+Del-Del+Ins}")
```

cuts selected text to the clipboard (Shift+Del), then pastes it at the insertion point (Shift+Ins).

With the WAIT modifier, SENDKEYS is processed before the next macro statement. Otherwise, SENDKEYS is processed after the next macro statement. For example,

```
APPLICATION(A1; "WordPerfect"; Default; "US")
Display(On!)
InhibitInput(Off!)
SENDKEYS("C:\WPWIN60\MACROS\*.WCM {Enter}")
FileOpenDlg
```

displays the Open File dialog box, enters C:\WPWIN60\MACROS*.WCM in the Filename edit box, and presses Enter. The directory changes to C:\WPWIN60\MACROS, and files with a .WCM extension appear in the Filename list box. The same example with the WAIT modifier enters C:\WPWIN60\MACROS*.WCM in the document before the Open File dialog box is displayed.

Set InhibitInput(Off!) for SENDKEYS to operate correctly.

Syntax

SENDKEYS WAIT (*KeyCode*; *MarkupLanguage*)

Parameters

KeyCode: *string*
Keystrokes to send.

{VKnnn}	nnn = ANSI character number
{Alt}	
{Ctrl}	
{Control}	
{Shift}	
{0} - {9}	Digits
{A} - {Z}	Alphabet
{F1} - {F16}	Function keys
{NumLock}	
{NumAdd}	
{NumSubtract}	
{NumMultiply}	
{NumDivide}	
{NumDecimal}	
{Num0} - {Num9}	Numpad numbers
{Left}	
{Right}	
{Up}	
{Dn}	
{Down}	
{PgDn}	
{PageDown}	
{PgUp}	
{PageUp}	
{Bksp}	
{Backspace}	
{Break}	Cancel
{CapsLock}	
{Clear}	
{Del}	
{Delete}	
{End}	
{Enter}	
{Esc}	
{Escape}	
{Help}	VK Help key
{Home}	
{Ins}	
{Insert}	
{Minus}	
{Pause}	
{ScrLock}	
{ScrollLock}	
{PrintScrn}	
{PrintScreen}	
{Space}	

{Tab}
{LeftBrace} {
{RightBrace} }

MarkupLanguage: Numeric Expression
Specifies which language interprets the key codes. The values are,
1 WPWin 6.0 keystring language.

SETCURRENTDIRECTORY

Purpose
Specifies new default directory.

Syntax
boolean SETCURRENTDIRECTORY (DirectoryName: *string*)

Return Value
boolean True if successful, False if not.

Parameters
DirectoryName: *string*
Include full path.

See Also
CREATEDIRECTORY
DELETEDIRECTORY
DOESDIRECTORYEXIST
GETCURRENTDIRECTORY
RENAMEDIRECTORY

SETFILEATTRIBUTES

F

Purpose
Changes file attributes.

Syntax
boolean SETFILEATTRIBUTES (Filename: *string*; Attributes: *enum*; Prompts: *enum*)

Return Value
boolean True if successful, False if not.

Parameters
Filename: *string*

Attributes: *enum* (optional)
- Normal!
- ReadOnly!
- Hidden!
- System!
- Label!
- Directory!
- Archived!

Prompts: *enum* (optional)
Prompts if attribute does not exist.
- NoPrompts!
- Prompts!

See Also
GETFILEATTRIBUTES

SETFILEDATEANDTIME

Purpose
Changes a file's creation date and time.

Syntax
boolean SETFILEDATEANDTIME (Filename: *string*; DateAndTime: *numeric*)

Return Value
boolean True if successful, False if not.

Parameters
Filename: *string*
Include full path.

DateAndTime: *numeric*
Number of seconds since January 1, 1980.

See Also
GETFILEDATEANDTIME

SPEED

F

Purpose

Slows macro execution.

Time is measured in tenths of a second. SPEED(0) runs at maximum speed. SPEED(5) delays a macro one-half second between statements. The maximum delay is one minute, or SPEED(600).

```
SPEED(10) // delay one second between beeps
    BEEP
    BEEP
    BEEP
```

Syntax

SPEED (*TenthsOfSeconds*)

Parameters

TenthsOfSeconds: *numeric*

A number from zero to 600. Divide the number by 10 to calculate the number of seconds.

See Also

WAIT

STRLEN

F

Purpose

Determines the number of characters in a string.

The string can be a variable, constant, character string, or result of an expression.

vWord := "WordPerfect"
vNumber := STRLEN(vWord)
Result: vNumber equals 11

vNumber := STRLEN(45899)
Result: vNumber equals 5

vNumber := STRLEN("WordPerfect")
Result: vNumber equals 11

vNumber := STRLEN(9 + 9)
Result: vNumber equals 2

Syntax

ReturnValue := STRLEN (*string*)

Returns

The number of characters in a string.

Parameters

String: *string*
A variable, constant, character string, or result of an expression.

See Also

CHARLEN
NUMSTR
STRNUM
STRPOS
STRUNIT
SUBSTR

STRNUM

F

Purpose
Converts a character string of numbers to a numeric equivalent.

Syntax
ReturnValue := STRNUM (*string*)

Returns
A number.

vNum := STRNUM("123")
Result: vNum = 123

vNum := STRNUM("123.5")
Result: vNum = 123.5

vNum := STRNUM("9 + 9")
Result: vNum = 9

Parameters
String: *string*
Contains a character string of numbers. STRNUM recognizes the decimal point defined by the sDecimal setting in the [intl] section of the Windows WIN.INI file. Alphabetic characters, operators, and punctuation marks, and everything that follows them are ignored.

See Also
NUMSTR
STRLEN
STRPOS
STRUNIT
SUBSTR

STRPOS

Purpose
Determines whether a character string is also a substring.

Syntax
ReturnValue := STRPOS (*String; SubString*)

Returns
The beginning position of a substring, or zero if a substring is not found.

vPos := STRPOS("WordPerfect"; "Perfect")
Result: vPos = 5

vPos := STRPOS("WordPerfect"; "Scott")
Result: vPos = 0

Parameters
String: *string*
A character string to evaluate.

Substring: *string*
A substring to locate in String parameter.

See Also
NUMSTR
STRLEN
STRNUM
STRUNIT
SUBSTR

STRUNIT

F

Purpose

Converts a string of numbers to a measurement.

The string may contain a number, an arithmetic expression that results in a number, or a character string of numbers. If a character string is used, alphabetic characters (except units of measure), operators, and punctuation marks are ignored. The default unit of measure is WordPerfect units. You can change the default with DEFAULTUNITS, or by specifying a unit of measure as part of the character string.

DEFAULTUNITS(Centimeters)
vUnit := STRUNIT(9)
Result: vUnit equals 9.C

vUnit :equals STRUNIT(4+3)
Result: vUnit equals 7.C

vUnit :equals STRUNIT("10")
Result: vUnit equals 10.C

vUnit :equals STRUNIT("10abc")
Result: vUnit equals 10.C

vUnit :equals STRUNIT("10i")
Result: vUnit equals 10I

The units of measure are:

"	inches
i	inches
c	centimeters
m	millimeters
p	points (72 per inch)
w	WP units (1200 per inch)

Syntax

ReturnValue := STRUNIT (*string*)

Returns

A unit of measure.

Parameters

String: *string*
A number or character string of numbers.

See Also

NUMSTR
STRLEN
STRNUM
STRPOS
SUBSTR
UNITSTR

SUBCHAR

Purpose
Extracts a substring from a character string.

Use CHARPOS to locate a substring.

Syntax
ReturnValue := SUBCHAR (*String; Beginning; NumberOfChars*)

Returns
A character string.

```
vSub := SUBCHAR("WordPerfect"; 1; 4) // vSub = "Word"
```

Parameters
String: *string*
A character string to evaluate.

Beginning: *numeric*
The starting position of a substring.

NumberOfChars: *numeric*
The number of characters to extract.

See Also
CHARLEN
CHARPOS
SUBSTR

SUBSTR

Purpose
Extracts a substring from a character string.

Use STRPOS to locate a substring.

Syntax
ReturnValue := SUBSTR (*String; Beginning; NumberOfChars*)

Returns
A character string.

```
vSub := SUBSTR("WordPerfect"; 1; 4) // vSub = "Word"
```

Parameters
String: *string*
A character string to evaluate.

Beginning: *numeric*
The starting position of a substring.

NumberOfChars: *numeric*
The number of characters to extract.

See Also
NUMSTR
STRLEN
STRNUM
STRPOS
STRUNIT
SUBCHAR

SWITCH-ENDSWITCH

F

Purpose

A conditional statement that tests for matching expressions. If a match is found, a statement (or statement block) is executed. See Conditional Statements.

If Test matches Selector, the statement block that follows Selector is executed and no other evaluation is made. Test and Selector are case sensitive and must match exactly.

If CONTINUE follows an executed statement block, the next statement block is automatically executed. CONTINUE is optional.

The DEFAULT statement block is executed if no Selector matches Test. DEFAULT is optional. If no match is found and DEFAULT is not used, the macro continues to the first statement after ENDSWITCH.

If BREAK occurs in a state block, the macro continues to the first statement after ENDSWITCH.

ENDSWITCH closes a SWITCH statement.

Syntax

The general form of a SWITCH statement is,

```
SWITCH (Test)
    CASEOF Selector1:
        ...statement block...
    CONTINUE
    CASEOF Selector2; Selector3:
        ...statement block...
    DEFAULT:
        ...statement block...
ENDSWITCH
```

Parameters

Test: *any*

The control expression. Variables are assigned values by commands such as GETSTRING, GETNUMBER, or MENU.

Selector: *any*

An expression (variable, constant, character) with a value that is usually assigned before the macro is compiled. It is possible to assign the value at run-time. *Selector* always follows a CASEOF statement.

See Also

BREAK
CASE
CASE CALL
IF

TOLOWER

F

Purpose

Converts uppercase letters to lowercase.

The value returned by TOLOWER can be assigned to a variable or used directly by a statement such as IF.

```
vAnswer := "YES"
vAnswer := TOLOWER(vAnswer)
```
Result: vAnswer equals lowercase yes

Shorthand notation example

```
IF(TOLOWER(vAnswer) = "yes")
    BEEP
ELSE
    QUIT
ENDIF
```

Explanation: Beep if vAnswer equals lowercase yes.

Syntax

ReturnValue := TOLOWER (*String*)

Returns

The lowercase equivalent of uppercase letters.

Parameters

String: *string*
A string to convert.

See Also

TOUPPER

TOUPPER

F

Purpose

Converts lowercase letters to uppercase.

The value returned by TOUPPER can be assigned to a variable or used directly by a statement such as IF.

```
vAnswer := "yes"
vAnswer := TOUPPER(vAnswer)
```
Result: vAnswer equals uppercase YES

Shorthand notation example

```
IF(TOUPPER(vAnswer) = "YES")
    BEEP
ELSE
    QUIT
ENDIF
```

Explanation: Beep if vAnswer equals uppercase YES.

Syntax

ReturnValue := TOUPPER (*String*)

Returns

The uppercase equivalent of lowercase letters.

Parameters

String: *string*
A string to convert.

See Also

TOLOWER

UNITSTR

F

Purpose

Converts a measurement to a character string.

If a unit of measure is not specified, the default is WordPerfect units. To change the default use DEFAULTUNITS. The following example converts WP units to centimeters and returns the result as a string:

```
vUnits := UNITSTR(1200; Centimeters!)
```
Result: vUnits equals "2.540000C"

The next example converts centimeters to millimeters and returns the result as a string:

```
DEFAULTUNITS(Centimeters!)
vUnits := UNITSTR(1.0; Millimeters!)
```
Result: vUnits equals "10.000000M"

The next example converts inches to centimeters and returns the result as a string:

```
vUnits := UNITSTR(2.0i; Centimeters!)
```
Result: vUnits equals "5.080000C"

See STRUNIT to convert a string of numbers to a measurement expression.

Syntax

ReturnValue := UNITSTR (*Value; Units*)

Returns

A measurement expressed as a string.

Parameters

Value: *numeric*

A real number measurement. Unless you specifiy a unit of measure, the default is WP Units. To change the default use DEFAULTUNITS.

Units: *enum*

The unit of measure that *Value* converts to.

Inches!	
Centimeters!	
Millimeters!	
Points!	(72 per inch)
WPUnits!	(1200 per inch)
WP1200ths!	(1200 per inch)

See Also

GETUNITS
NUMSTR
STRLEN
STRNUM
STRPOS
SUBSTR

STRUNIT

USE

F

Purpose

Specifies a macro file with functions and/or procedures, or dialogs that can be called from another macro.

USE is a non-executable statement that can occur anywhere in a macro. A macro with a calling statement to a function or procedure in another macro file must include a USE statement that identifies the file. A USE macro file that includes only function and/or procedure statements must be compiled like any macro file (see Macro File Libraries). A USE macro file that does not contain command statements, such as PROCEDURE or FUNCTION, does not need to be compiled.

Macro files identified by multiple USE statements are searched from first to last. Thus a parent macro always calls the first occurrence of a function or procedure with the same name in different USE files.

Syntax

USE (*MacroFile*)

Parameters

MacroFile: *string*

The path and name of a macro file. You cannot substitute *string* with a variable.

See Also

FUNCTION
INCLUDE
PROCEDURE

VARERRCHK

Purpose

Determines how a macro responds to unitialized variables (variables not assigned a value).

Syntax

VARERRCHK (*State*)

Parameters

State: *enum*

Specifies the state of variable checking. The default is VARERRCHK(On!).

Off!	Ignore unitialized variables by temporarily assigning a value of 0.
On!	Display a run-time Error message when a macro attempts to use an unitialized variable, and end the macro.

See Also

GLOBAL
LOCAL
PERSIST
PERSISTALL

WAIT

F

Purpose

Pauses macro execution.

Time is measured in tenths of a second. The maximum pause is one minute, or WAIT(600).

Syntax

WAIT (*TenthsOfSeconds*)

Parameters

TenthsOfSeconds: *numeric*
A number from zero to 600. Divide the number by 10 to calculate the number of seconds.

See Also

SPEED

WHILE-ENDWHILE

Purpose

A loop statement that executes while the expression at the top of the loop is true (see Loop Statements).

The loop does not execute the first time unless Test is true. When Test is false, the first statement after ENDWHILE is executed.

Syntax

The general form of a WHILE statement is,

```
WHILE (Test)
    ...statement block...
ENDWHILE
```

Parameters

Test: *boolean*
Evaluates to true or false.

See Also

FOR
FOREACH
FORNEXT
REPEAT

For Example...

```
APPLICATION (A1; "WPOffice"; Default; "US")
// Execute Windows Cardfile three times: full screen, normal size, and to an icon
// FORNEXT decrements initial value
// Demonstrate APPEXECUTEEXT command

FORNEXT(vCtrl; 3; 1; -1)
   ASSIGN(vReturn; APPEXECUTEEXT("c:\windows\cardfile.exe"; vCtrl))
   IF(vReturn < 32)
      BEEP
      MESSAGEBOX(vStatus; "ERROR"; "Unable to execute Cardfile.exe"; IconExclamation!)
      QUIT
   ENDIF
 WAIT(3)
ENDFOR
```

For Example...

```
APPLICATION(A1; "WPOffice"; Default; "US")
// Execute Windows Calculator
// Locate and activate 3 times
// Demonstrate APPEXECUTE command

ASSIGN(hWP; APPLOCATE("*"))
APPEXECUTE("c:\windows\calc.exe")
ASSIGN(hCalc; APPLOCATE("Calculator"))
FORNEXT(vCtrl; 1; 2; 1)
   WAIT(10)
   APPACTIVATE(hWP)
   WAIT(10)
   APPACTIVATE(hCalc)
ENDFOR
```

For Example...

```
APPLICATION(A1; "WPOffice"; Default; "US")
// Display message box on ASSERT condition
// Demonstrate ASSERT command

ONCANCEL(Condition)
ONERROR(Condition)
ONNOTFOUND(Condition)

LABEL(Start)
   MENU(vPick; Digit!; ; ; {"Assert Cancel"; "Assert Error"; "Assert Not Found"; "Quit"})
   SWITCH(vPick)
      // If the MenuType parameter were Letter! instead of Digit!,
      // the following CASEOF selectors would still be numeric.
      CASEOF 1: ASSERT(CancelCondition!)
      CASEOF 2: ASSERT(ErrorCondition!)
      CASEOF 3: ASSERT(NotFoundCondition!)
      CASEOF 4: QUIT
      DEFAULT: GO(Condition)
   ENDSWITCH

LABEL(Condition)
   SWITCH(ErrorNumber)
      CASEOF 1: vMsg := "1 (Cancel condition)"
      CASEOF 2: vMsg := "2 (Error condition)"
      CASEOF 7: vMsg := "3 (Not Found condition)"
      DEFAULT: vMsg := "?"
   ENDSWITCH
   MESSAGEBOX(vStatus; "ASSERT COMMAND"; "You selected " + vMsg; IconInformation!)
   GO(Start)
```

For Example...

```
APPLICATION(A1; "WPOffice"; Default; "US")
// Beep vCount times - BREAK when vCount equals 4
// Remember: functions do not execute unless called
// Demonstrate BEEP command

FUNCTION BeepBeep(vCount)
   REPEAT
      BEEP
      WAIT(3)
      vCount := vCount - 1
   UNTIL(vCount = 0)
   RETURN
ENDFUNC

FOREACH(vCount; {1; 2; 3; 4; 5})
   IF(vCount = 4)
      BREAK
   ENDIF
   BeepBeep(vCount)
   WAIT(5)
ENDFOR

MESSAGEBOX(x; "BREAK"; "Variable vCount equals 4"; IconInformation!)
QUIT
```

For Example...

```
APPLICATION(A1; "WPOffice"; Default; "US")
// Test user input
// Remember: procedures do not execute unless called
// Demonstrate GETNUMBER, PROCEDURE, and PROCEDURE PROTOTYPE commands

PROCEDURE PROTOTYPE ErrorMsg(x)

REPEAT
   GETNUMBER(vNum; "Enter a number from 1 to 5"; "GETNUMBER EXAMPLE")
   IF((vNum < 1) OR (vNum > 5))
      CALL ErrorMsg(vNum)
   ELSE
      MESSAGEBOX(vStatus; "YES!"; "You entered " + vNum; IconInformation!)
   ENDIF
UNTIL((vNum > 0) AND (vNum < 6))

PROCEDURE ErrorMsg(x)
   BEEP BEEP BEEP
   IF(x < 1)
      vMessage := x + " is less than 1"
   ELSE
      vMessage := x + " is greater than 5"
   ENDIF
   MESSAGEBOX(vStatus; "OUT-OF-RANGE ERROR" ; vMessage + " - please try again";
   IconExclamation!)
ENDPROC
```

For Example...

```
APPLICATION(A1; "WPOffice"; Default; "US")
// Test user input for Y or N
// For Cancel condition: chose Cancel, click Close, press Alt+F4, or double-click system menu
box
// Demonstrate CHARLEN command

CANCEL(On!)
ONCANCEL CALL(CancelMessage)
PROCEDURE Message(vPrompt; vIcon)
   IF(vIcon = "!")
      BEEP
      MESSAGEBOX(y; "Message Box"; vPrompt; IconExclamation!)
   ELSE
      MESSAGEBOX(y; "Message Box"; vPrompt; IconInformation!)
   ENDIF
ENDPROC

vStr := ""
REPEAT
   GETSTRING(vStr; "Press one character: ""Y"" for Yes or ""N"" for no"; "GETSTRING
   EXAMPLE"; 50)
   IF(vStr = "")
      vStr := "<Enter>"
   ENDIF

   vTest := TOUPPER(vStr)
   IF(CHARLEN(vStr) > 1)
         CALL Message("You typed or pressed """ + vStr + """"; "!")
   ELSE
      IF((vTest = "Y") OR (vTest = "N"))
         CALL Message("Well done - You pressed " + vTest; "i")
      ELSE
         CALL Message("You pressed """ + vStr + """"; "!")
      ENDIF
   ENDIF
UNTIL(((vTest = "Y") OR (vTest = "N")) AND NOT(CHARLEN(vStr) > 1))
QUIT

LABEL(CancelMessage)
   vStr := "a Cancel control"
   RETURN
```

For Example...

```
APPLICATION(A1; "WPOffice"; "US")
APPLICATION(A1; "WordPerfect"; Default; "US")
// Specify a unit of measure or accept WP units default
// Plays macro in WordPerfect for Windows 6.0a
// Launches WordPerfect if not running
// Demonstrate GETUNITS command

MENU(vChoice; Digit; ; ; {"Advance down"; "Advance up"; "Advance left"; "Advance right"})
SWITCH(vChoice)
   CASEOF 1: vPrompt := "Advance down:"
   CASEOF 2: vPrompt := "Advance up:"
   CASEOF 3: vPrompt := "Advance left:"
   CASEOF 4: vPrompt := "Advance right:"
ENDSWITCH
DEFAULTUNITS(Inches!)
GETUNITS(vUnit; vPrompt; "GETUNITS EXAMPLE")
SWITCH(vChoice)
   CASEOF 1: ADVANCE(AdvanceDown!; vUnit)
   CASEOF 2: ADVANCE(AdvanceUp!; vUnit)
   CASEOF 3: ADVANCE(AdvanceLeft!; vUnit)
   CASEOF 4: ADVANCE(AdvanceRight!; vUnit)
ENDSWITCH
```

For Example...

```
APPLICATION(A1; "WPOffice"; Default; "US")
// Execute Windows Calculator, Calendar, and Cardfile
// Demonstrate CASE CALL inside a REPEAT-UNTIL loop

ASSIGN(hWP; APPLOCATE("*"))
REPEAT
   MENU(vChoice; Digit; ; ; {"Calculator"; "Calendar"; "Card File"; "Quit"})
   CASE CALL(vChoice; {1; Program; 2; Program; 3; Program; 4; QuitMacro}; QuitMacro)
UNTIL(vChoice = 4)

LABEL(Program)
   SWITCH(vChoice)
      CASEOF 1: vProgram := "c:\windows\calc.exe"
      CASEOF 2: vProgram := "c:\windows\calendar.exe"
      CASEOF 3: vProgram := "c:\windows\cardfile.exe"
   ENDSWITCH
   APPEXECUTE(vProgram)
   RETURN

LABEL(QuitMacro)
   MESSAGEBOX(vStatus; "EXIT"; "Are you finished?"; IconQuestion! | YesNo!)
   IF(vStatus = 6)
      APPACTIVATE(hWP)
      QUIT
   ELSE
      vChoice := ""
      RETURN
   ENDIF
```

For Example...

```
APPLICATION(A1; "WPOffice"; "US")
APPLICATION(A1; "WordPerfect"; Default; "US")
// Create personal or business letterhead
// Plays macro in WordPerfect for Windows 6.0a
// Launches WordPerfect if not running

MENU(vChoice; Digit; ; ; {"Personal letterhead"; "Business letterhead"; "Quit"})
FileNew
CASE(vChoice; {1; Personal; 2; Business; 3; QuitMacro}; QuitMacro)

LABEL(Personal)
   Center
   AttributeAppearanceOn(Bold!)
   FontSize(16.0p)
   Type("From Dan's Desk")
   AttributeAppearanceOff(Bold!) HardReturn
   Center FontSize(12.0p)
   DateCode
   HardReturn HardReturn HardReturn
   Type("Dear ")
   QUIT

LABEL(Business)
   DateCode HardReturn HardReturn
   Type("Mr. John Doe") HardReturn
   Type("Executive Vice President") HardReturn
   Type("XYZ Company") HardReturn
   Type("4399 Oak View Drive") HardReturn
   Type("Orem, UT 84057") HardReturn HardReturn
   Type("Dear Mr. Doe:")
   QUIT

LABEL(QuitMacro)
   QUIT
```

For Example...

```
// There are four macros in this example
// Compile separately

APPLICATION(A1; "WPOffice"; Default; "US")
// macro1.wcm
MESSAGEBOX(vStatus; vTitle; "Macro 1 - " + vPrompt; IconInformation!)

APPLICATION(A1; "WPOffice"; Default; "US")
// macro2.wcm
MESSAGEBOX(vStatus; vTitle; "Macro 2 - " + vPrompt; IconInformation!)

APPLICATION(A1; "WPOffice"; Default; "US")
// macro3.wcm
MESSAGEBOX(vStatus; vTitle; "Macro 3 - " + vPrompt; IconInformation!)

APPLICATION(A1; "WPOffice"; Default; "US")
// Calling Macro
// Demonstrate CHAIN command and GLOBAL variables

GLOBAL vTitle; vPrompt
MENU(vChoice; Letter; ; ; {"Macro 1"; "Macro 2"; "Macro 3"})
ASSIGN(vPrompt; "Computer beeps before chain macro called")
ASSIGN(vMacrosPath; EnvPrefMacroPath)
SWITCH(vChoice)
   CASEOF 1:
      ASSIGN(vTitle; "MACRO ONE")
      CHAIN(vMacrosPath + "macro1.wcm")
   CASEOF 2:
      ASSIGN(vTitle; "MACRO TWO")
      CHAIN(vMacrosPath + "macro2.wcm")
   CASEOF 3:
      ASSIGN(vTitle; "MACRO THREE")
      CHAIN(vMacrosPath + "macro3.wcm")
ENDSWITCH
// beep sounds before chain macro called
BEEP WAIT(5)
```

For Example...

```
// There are four macros in this example
// Compile separately

APPLICATION(A1; "WPOffice"; Default; "US")
// macro1.wcm
MESSAGEBOX(vStatus; vTitle; "Macro 1 - " + vPrompt; IconInformation!)
RETURN

APPLICATION(A1; "WPOffice"; Default; "US")
// macro2.wcm
MESSAGEBOX(vStatus; vTitle; "Macro 2 - " + vPrompt; IconInformation!)
RETURN

APPLICATION(A1; "WPOffice"; Default; "US")
// macro3.wcm
MESSAGEBOX(vStatus; vTitle; "Macro 3 - " + vPrompt; IconInformation!)
RETURN

APPLICATION(A1; "WPOffice"; Default; "US")
// Calling Macro
// Demonstrate RUN command and GLOBAL variables

GLOBAL vTitle; vPrompt
ASSIGN(vPrompt; "RUN macros can return to their caller")
ASSIGN(vMacrosPath; EnvPrefMacroPath)
REPEAT
   MENU(vChoice; Letter; ; ; {"Macro 1"; "Macro 2"; "Macro 3"; "Quit"})
   SWITCH(vChoice)
      CASEOF 1:
          ASSIGN(vTitle; "MACRO ONE")
          RUN(vMacrosPath + "macro1.wcm")
      CASEOF 2:
          ASSIGN(vTitle; "MACRO TWO")
          RUN(vMacrosPath + "macro2.wcm")
      CASEOF 3:
          ASSIGN(vTitle; "MACRO THREE")
          RUN(vMacrosPath + "macro3.wcm")
      DEFAULT: QuitMacro
   ENDSWITCH
UNTIL(vChoice = 4)

LABEL(QuitMacro)
   BEEP
   MESSAGEBOX(vStatus; "QUIT"; "This example uses global variables"; IconExclamation!)
   DISCARD vTitle; vPrompt
   QUIT
```

For Example...

```
APPLICATION(A1; "WPOffice"; Default; "US")
// Create a loop with GO command
// Demonstrate CASE, MESSAGEBOX, and system variables

LABEL(Start)
   MENU(vChoice; Digit; ; ; {"IconExclamation!"; "IconInformation!"; "IconQuestion";
   "IconStop!"; "Quit"})
   CASE(vChoice; {1; Exclaim; 2; Info; 3; Question; 4; Stop; 5; QuitMacro}; QuitMacro)

LABEL(Exclaim)
   MESSAGEBOX(x; "EXCLAMATION"; "Example of an exclamation icon"; IconExclamation!)
   GO(LoopToStart)

LABEL(Info)
   ASSIGN(vID; EnvNetworkLoginID)
   MESSAGEBOX(x; "INFORMATION"; "Example of an information icon - Network login ID is " +
   vID + "."; IconInformation!)
   GO(LoopToStart)

LABEL(Question)
   Yes := 6
   MESSAGEBOX(vAns; "QUESTION"; "Example of a question icon." + NTOC(0F90Ah) +
   "Would you like to know the default macro path?"; IconQuestion! | YesNo!)
   IF(vAns = Yes)
      vPrompt := "The default macro path is " + EnvPrefMacroPath
      MESSAGEBOX(x; "MACRO PATH"; vPrompt; IconInformation!)
      GO(LoopToStart)
   ELSE
      GO(LoopToStart)
   ENDIF

LABEL(Stop)
   MESSAGEBOX(z; "STOP"; "Example of a stop icon"; IconStop!)
   GO(LoopToStart)

LABEL(LoopToStart)
   MESSAGEBOX(vAns; "Message"; "Would you like to continue?"; IconQuestion! | YesNo!)
   IF(vAns = 7)
      GO(QuitMacro)
   ELSE
      GO(Start)
   ENDIF

LABEL(QuitMacro)
   BEEP
   QUIT
```

For Example...

```
APPLICATION(A1; "WPOffice"; Default; "US")
// Extract numbers from a character string
// Demonstrate CHARLEN command

vNumbers := ""
vPrompt := "Enter text that includes both letters and numbers:"

LABEL(Start)
GETSTRING(vStr; vPrompt; "EXTRACT NUMBERS")
IF(vStr = "")
   BEEP
   MESSAGEBOX(x; "ERROR"; "You pressed return without entering any text"; IconStop!)
   GO(Start)
ENDIF
vLen := CHARLEN(vStr)
vPos := 1
WHILE(NOT(vPos = vLen + 1))
   vTest := SUBCHAR(vStr; vPos; 1)
   IF((CTON(vTest) > 47) AND (CTON(vTest) < 58))
      vNumbers := vNumbers + vTest
   ENDIF
   vPos := vPos + 1
ENDWHILE
IF(CHARLEN(vNumbers) > 0)
   MESSAGEBOX(x; "NUMBER STRING"; "You entered the following numbers: " + vNumbers;
   IconInformation!)
ELSE
   BEEP
   MESSAGEBOX(x; "ERROR"; "You didn't enter any numbers"; IconExclamation!)
ENDIF
```

For Example...

```
APPLICATION(A1; "WPOffice"; Default; "US")
// Extract "Perfect" from WordPerfect
// Test if "program" was entered
// Demonstrate SUBCHAR command

vNumbers := ""
vPrompt := "Enter the following text: ""WordPerfect program"""

LABEL(Start)
GETSTRING(vStr; vPrompt; "USING SUBCHAR")
IF(vStr = "")
   BEEP
   MESSAGEBOX(x; "ERROR"; "You pressed return without entering any text"; IconStop!)
   GO(Start)
ENDIF
vSub := SUBCHAR(vStr; 5; 7)
IF(vSub = "Perfect")
   MESSAGEBOX(x; "SUBCHAR EXAMPLE"; "WordPerfect is simply """ + vSub + """";
   IconInformation!)
   IF(CHARPOS(vStr; "program") = 0)
      BEEP
      MESSAGEBOX(x; "CHARPOS EXAMPLE"; "You entered ""WordPerfect"" but not
      ""program"""; IconExclamation!)
   ENDIF
ELSE
   BEEP
   MESSAGEBOX(x; "ERROR"; """WordPerfect"" wasn't the first word"; IconExclamation!)
   GO(Start)
ENDIF
```

For Example...

```
APPLICATION(A1; "WordPerfect"; Default; "US")
// Start conversation with GroupWise 4.1 from WordPerfect 6.0a
// Execute GroupWise 4.1 command
// Demonstrate DDEEXECUTE and DDEINITIATE commands

ASSIGN(hConv; DDEINITIATE("GROUPWISE"; "COMMAND"))
IF(hConv) // IF hConv not equal to 0
   DDEEXECUTE(hConv; "ViewOpenNamed(ViewName:""Mail""; ViewType:Mail!)")
   DDETERMINATEALL
ELSE
   BEEP
   MESSAGEBOX(x; "ERROR"; "Is GroupWise running?   Conversation not started.";
   IconQuestion!)
ENDIF
```

For Example...

```
APPLICATION(A1; "WordPerfect"; Default; "US")
// Start conversation with GroupWise 4.1 from WordPerfect 6.0a
// Execute GroupWise 4.1 command
// Demonstrate DDEEXECUTEEXT command

ASSIGN(hConv; DDEINITIATE("GROUPWISE"; "COMMAND"))
IF(hConv) // IF hConv not equal to 0
   vReturn := DDEEXECUTEEXT(hConv; "ViewOpenNamed(""Mail""; Mail!)"; 0; Message)
   IF(vReturn = 0)
      MESSAGEBOX(x; "ERROR"; "Command not received by GroupWise"; IconExclamation!)
   ENDIF
ELSE
   BEEP
   MESSAGEBOX(x; "ERROR"; "Is GroupWise running?   Conversation not started.";
   IconQuestion!)
ENDIF
DDETERMINATEALL
QUIT

LABEL(Message)
   ASSIGN(vMsg; "GroupWise acknowledges receipt of ViewOpenNamed command")
   MESSAGEBOX(z; "DDEEXECUTEEXT EXAMPLE"; vMsg; IconInformation!)
   RETURN
```

For Example...

```
APPLICATION(A1; "WordPerfect"; Default; "US")
// Request GroupWise 4.1 version number from WordPerfect 6.0a
// Demonstrate DDEREQUEST command
// This example is included for backward compatibility
// See Contents page, Using DDE

ASSIGN(hConv; DDEINITIATE("WPOFFICE40"; "GETOFFICEDATA"))
IF(hConv)
   ASSIGN(vMajor; DDEREQUEST(hConv; "GetOfficeData(ID; MajorVersion!)"))
   ASSIGN(vMinor; DDEREQUEST(hConv; "GetOfficeData(ID; MinorVersion!)"))
   MESSAGEBOX(x; "DDEREQUEST EXAMPLE"; "GroupWise version is " + vMajor + "." +
   vMinor; IconInformation!)
   DDETERMINATE(hConv)
ELSE
   BEEP
   MESSAGEBOX(x; "ERROR"; "Is GroupWise running?   Conversation not started.";
   IconQuestion!)
ENDIF
```

For Example...

```
APPLICATION(A1; "WPOffice"; Default; "US")
// Assign fax number to ExpressFax+
// Demonstrate DDEPOKE command

ASSIGN(hConv; DDEINITIATE("FAXMNG"; "Transmit"))
IF(hConv)
   GETSTRING(vFax; "Enter FAX Number"; "ExpressFax+")
   DDEPOKE(hConv;"FAX Number"; vFax)
   DDETERMINATE(hConv)
ELSE
   BEEP
   MESSAGEBOX(x; "ERROR"; "Is ExpressFax+ running?   Conversation not started.";
   IconQuestion!)
ENDIF
```

For Example...

```
APPLICATION(A1; "WPOffice"; Default; "US")
// Declare an array of months
// Demonstrate DECLARE command, recursive LABEL statement

DECLARE Months[12]
x := 1

FOREACH(vMonth; {"January"; "Februrary"; "March"; "April"; "May"; "June"; "July"; "August";
"September"; "October"; "November"; "December"})
   Months[x] := vMonth
   x := x + 1
ENDFOR

LABEL(Start)
   MENU(vNmbr; Letter; ; ; {"January"; "Februrary"; "March"; "April"; "May"; "June"; "July";
   "August"; "September"; "October"; "November"; "December"; "QUIT"})
   IF(vNmbr = 13)
      BEEP
      QUIT
   ELSE
      MESSAGEBOX(vStatus; "MONTHS OF THE YEAR"; "You selected " + Months[vNmbr];
      IconInformation!)
      GO(Start)
   ENDIF
```

For Example...

```
APPLICATION(A1; "WPOffice"; Default; "US")
// Change current working directory
// Demonstrate DLLLOAD, DLLCALL, and DLLFREE
// Assign hard return code to a variable, insert in prompt string
// For information about WfsSetCurDir and WfsGetCurDir, see WordPerfect 6.0 for Windows
Software Developer's Kit

DLLLOAD(hInstance; "shwin20.dll")
DLLCALL(hInstance; "WfsGetCurDir"; vMsgResult: WORD; {ADDRESS (ANSISTRING(CurDir))})

ASSIGN(HdReturn; NTOC(0F90Ah)) // hard return code
LABEL(Start)
   GETSTRING(vNewDir; "The current working directory is " + CurDir + HdReturn + HdReturn
   + "Enter a new working directory:"; "DLL EXAMPLE")
   IF(vNewDir = "")
      BEEP
      MESSAGEBOX(x; "ERROR"; "You pressed return without entering a path"; IconStop!)
      GO(Start)
   ENDIF
   DLLCALL(hInstance; "WfsSetCurDir"; vMsgResult: WORD; {vNewDir})
   DLLCALL(hInstance; "WfsGetCurDir"; vMsgResult: WORD; {ADDRESS
   (ANSISTRING(CurDir))})
   MESSAGEBOX(vAns; "DLL EXAMPLE"; "The new working directory is " + CurDir;
   IconInformation! | RetryCancel!)
      IF(vAns = 2)
          BEEP
          QUIT
      ELSE
          GO(Start)
      ENDIF

   DLLFREE(hInstance)
   QUIT
```

For Example...

```
APPLICATION(A1; "WPOffice"; Default; "US")
// LABEL(ErrorMsg) executes with ERROR(On!)
// Demonstrate ERROR command

ONERROR(ErrorMsg)

REPEAT
   MENU(vChoice; Digit; ; ; {"ERROR(Off!)"; "ERROR(On!)"; "Quit"})
   CASE CALL(vChoice; {1; OffMsg; 2; OnMsg; 3; QuitMacro}; QuitMacro)
UNTIL(vChoice = 3)

LABEL(OffMsg)
   ERROR(Off!)
   ASSERT(ErrorCondition!) // ignore Error condition
   MESSAGEBOX(x; "ERROR(Off!)"; "Error condition ignored"; IconInformation!)
   RETURN

LABEL(OnMsg)
   ERROR(On!)
   ASSERT(ErrorCondition!) // LABEL(ErrorMsg) executed
   RETURN

LABEL(ErrorMsg)
   BEEP
   MESSAGEBOX(x; "ERROR CONDITION"; "An error condition causes LABEL(ErrorMsg) to
   execute"; IconInformation!)
   RETURN

LABEL(QuitMacro)
   QUIT
```

For Example...

```
APPLICATION(A1; "WPOffice"; Default; "US")
// LABEL(NotFoundMsg) executes with NOTFOUND(On!)
// Demonstrate NOTFOUND command

ONNOTFOUND(NotFoundMsg)

REPEAT
   MENU(vChoice; Digit; ; ; {"NOTFOUND(Off!)"; "NOTFOUND(On!)"; "Quit"})
   CASE CALL(vChoice; {1; OffMsg; 2; OnMsg; 3; QuitMacro}; QuitMacro)
UNTIL(vChoice = 3)

LABEL(OffMsg)
   NOTFOUND(Off!)
   ASSERT(NotFoundCondition!) // ignore Not Found condition
   MESSAGEBOX(x; "NOTFOUND(Off!)"; "Not Found condition ignored"; IconInformation!)
   RETURN

LABEL(OnMsg)
   NOTFOUND(On!)
   ASSERT(NotFoundCondition!) // LABEL(NotFoundMsg) executed
   RETURN

LABEL(NotFoundMsg)
   BEEP
   MESSAGEBOX(x; "NOTFOUND CONDITION"; "A Not Found condition causes
   LABEL(NotFoundMsg) to execute"; IconInformation!)
   RETURN

LABEL(QuitMacro)
   QUIT
```

For Example...

```
APPLICATION(A1; "WPOffice"; Default; "US")
// Declare and test variable types
// Demonstrate BREAK, EXISTS, LOCAL, GLOBAL, and PERSIST commands

DISCARD var5; var1; var7
LOCAL var2; var9; var4
GLOBAL var3; var10; var6
PERSIST var5; var1; var7
ASSIGN(var8; "")

FOR(x; 1; x < 11; x + 1)
   ASSIGN(INDIRECT("var" + x); x)
   MESSAGEBOX(vStatus; "VAR" + x; "Continue to next variable?"; IconQuestion! | YesNo!)
   IF(vStatus = 7)
      BREAK
   ENDIF
ENDFOR

FOR(z; 1; z < (x + 1); z + 1)
   SWITCH(EXISTS(INDIRECT("var" + z)))
      CASEOF 1: Msg("Local"; z)
      CASEOF 2: Msg("Global"; z)
      CASEOF 3: Msg("Persist"; z)
   ENDSWITCH
ENDFOR

FUNCTION Msg(w; y)
   MESSAGEBOX(z; "VARIABLE TYPES"; "var" + y + " = " + w)
ENDFUNC
```

For Example...

```
APPLICATION(A1; "WPOffice"; Default; "US")
// Calculate fractions and integers
// Demonstrate FRACTION and INTEGER commands

ONCANCEL(QuitMacro)
ASSIGN(HdReturn; NTOC(0F90Ah)) // hard return code

REPEAT
   GETNUMBER(vNmbr; "Enter a real number:"; "FRACTION EXAMPLE")

   ASSIGN(vInteger; INTEGER(vNmbr))
   ASSIGN(vFraction; FRACTION(vNmbr))

   vMsg1 := "The integer portion of " + vNmbr + " is " + vInteger
   vMsg2 := "The fractional portion of " + vNmbr + " is " + vFraction

   MESSAGEBOX(x; "INTEGER - FRACTION"; vMsg1 + HdReturn + HdReturn + vMsg2 +
   HdReturn; IconInformation! | RetryCancel!)
UNTIL(x = 2)

LABEL(QuitMacro)
   QUIT
```

For Example...

```
APPLICATION(A1; "WPOffice"; Default; "US")
// Remember: functions must be called to execute
// Demonstrate FUNCTION and FUNCTION PROTOTYPE commands

FUNCTION PROTOTYPE Check(vBeep; HdReturn)

HdReturn := NTOC(0F90Ah)
x := 4
vBeep := 1

WHILE(x = 4)
   BEEP
   x := Check(vBeep; HdReturn)
   vBeep := vBeep + 1
ENDWHILE

MESSAGEBOX(z; "RETURN"; "The value of variable vStatus (" + x + ") is returned to variable
x, which ends the loop."; IconExclamation!)

FUNCTION Check(vBeep; HdReturn)
   MESSAGEBOX(vStatus; "FUNCTION EXAMPLE"; "Beeps: " + vBeep + HdReturn + HdReturn
   + "Choose Retry to beep again." + HdReturn; IconInformation! | RetryCancel!)
   RETURN(vStatus)
ENDFUNC
```

For Example...

```
APPLICATION(A1; "WPOffice"; Default; "US")
// Play .wav file
// Assume Windows sound system and that appropriate driver is installed
// Demonstrate MMPLAY and WAIT commands

No := 7
REPEAT
   MENU(vWav; Digit; ; ; {"Siren"; "Crickets"; "Snoring"; "Thunder"; "Dog bark"; "Elephant";
   "Quit"})
   SWITCH(vWav)
      CASEOF 1: PlaySound("siren.wav"; 115)
      CASEOF 2: PlaySound("crickets.wav"; 50)
      CASEOF 3: PlaySound("snoring.wav"; 30)
      CASEOF 4: PlaySound("thunder.wav"; 30)
      CASEOF 5: PlaySound("dogbark.wav"; 15)
      CASEOF 6: PlaySound("elephant.wav"; 15)
      CASEOF 7: QUIT
   ENDSWITCH
   MESSAGEBOX(vStatus; "MMPLAY"; "Would you like to hear another sound?"; IconQuestion!
   | YesNo!)
UNTIL(vStatus = No)

PROCEDURE PlaySound(vSound; vWait)
   MMPLAY("c:\sndsys\sounds\" + vSound)
   WAIT(vWait)
ENDPROC
```

For Example...

```
APPLICATION(A1; "WPOffice"; Default; "US")
// Speak an ASCII text file
// A text-to-speech driver must be installed
// Demonstrate MMSPEAK, MMSPEAKCLIPBOARD, MMSTOPSPEECH commands

REPEAT
   MENU(vChoice; Digit; ; ; {"File"; "Clipboard"; "Quit"})
   SWITCH(vChoice)
      CASEOF 1:
          GETSTRING(vStr; "Enter the path and name of a file to speak:"; "ASCII TEXT FILE")
          IF(vStr = "")
              BREAK
          ELSE
              MMSPEAK(vStr)
              CancelMessage()
          ENDIF
      CASEOF 2:
          MMSPEAKCLIPBOARD
          CancelMessage()
   ENDSWITCH
UNTIL(vChoice = 3)
QUIT

PROCEDURE CancelMessage()
   vCancel := 2
   MESSAGEBOX(vStatus; "SPEAK FILE"; "Choose Cancel any time to stop speech"; IconStop!
   | RetryCancel!)
   IF(vStatus = vCancel)
      MMSTOPSPEECH
   ENDIF
ENDPROC
```

For Example...

```
APPLICATION(A1; "WPOffice"; Default; "US")
APPLICATION(A2; "WordPerfect"; "US")

// GroupWise is the default application
// AboutDlg does not require a product prefix

AboutDlg()

NEWDEFAULT(A2)

// WordPerfect is not the default application
// AboutDlg requires a product prefix (A1)
A2.AboutDlg()
```

For Example...

```
APPLICATION(A1; "WPOffice"; Default; "US")
// Convert integer to Character equivalent
// NTOC(32) returns "space" character
// Demonstrate NTOC command

ViewOpenNamed(ViewName: "Expanded Mail"; ViewType: Mail!)
FocusSet(Place: Message!)

FOR(int; 33; int < 127; int + 1)
   Type(NTOC(int) + (NTOC(32)))
ENDFOR

Enter Enter

// Convert character set value to character equivalent
FOR(int; 0; int < 102; int + 1)
   Type(NTOC(4; int) + (NTOC(32)))
ENDFOR

MESSAGEBOX(vAns; "NTOC command"; "Close Expanded Mail view?"; YesNo! |
IconQuestion!)
IF(vAns = 6)
   CloseWindow ()
ENDIF

// The first FOR loop outputs the following characters:
// ! " # $ % & ' ( ) * + , - . / 0 1 2 3 4 5 6 7 8 9 : ; < = > ? @ A B C D E F G H I J K L M N O P Q
R S T U V W X Y Z [ \ ] ^ _ ` a b c d e f g h i j k l m n o p q r s t u v w x y z { | } ~

// The second FOR loop outputs the following typographic symbols (character set 4):
//              ¶ § ¡ ¿ « » £ ¥       ª º ½ ¼ ¢ ²     ® © ¤ ¾ ³     ’ ‘     ” “ — –                          •
 ¹
```

For Example...

```
APPLICATION(A1; "WPOffice"; Default; "US")
// Most type conversions are handled automatically
// The following type statements output identical character strings
// Demonstrate NUMSTR command

HdReturn := NTOC(0F90Ah)
TotalCharNumber:= EnvTextLineCharCount
TotalCharString:= NUMSTR(TotalCharNumber)

MESSAGEBOX(x; "AUTOMATIC TYPE CONVERSION"; "Variable TotalCharNumber: There are " +
TotalCharNumber + " characters in the current line." + HdReturn + HdReturn + "Variable
TotalCharString: There are " + TotalCharString + " characters in the current line." +
HdReturn; IconInformation!)
```

For Example...

```
APPLICATION(A1; "WPOffice"; Default; "US")
// Test Error condition
// Demonstrate ONERROR CALL command

ONERROR CALL(ErrorTest)
No := 7
REPEAT
   GETNUMBER(vNmbr; "Enter 3 and press <Enter>:"; "ONERROR CALL EXAMPLE")
   IF(vNmbr = 3)
      MESSAGEBOX(vStatus; "YES!"; "You entered 3 as requested.   Do you want to try
      again?"; IconInformation! | YesNo!)
   ELSE
      ASSERT(ErrorCondition!)
   ENDIF
UNTIL(vStatus = No)
QUIT

LABEL(ErrorTest)
   BEEP
   MESSAGEBOX(vStatus; "ERROR"; "You didn't enter 3 as requested"; IconExclamation!)
   RETURN
```

For Example...

```
APPLICATION(A1; "WPOffice"; Default; "US")
// Test Not Found condition
// Open view and retrieve text into the message box
// Demonstrate ONNOTFOUND command

NOTFOUND(On!)
ONNOTFOUND CALL(NotFoundTest)
No := 7
REPEAT
   GETSTRING(vStr; "Enter a string to search for and press <Enter>:"; "ONNOTFOUND CALL
   EXAMPLE")
   IF(vStr = "")
      NEXT
   ENDIF
   vError := False
   PosTextTop ()
   Find(MessageSearchString: vStr; SearchDir: Forward!; MessageMatch: BeginText!)
   IF(vError = False)
      vMessage := "Success!   Do you want to try again?"
   ELSE
      vMessage := "Not Found condition.   Do you want to try again?"
   ENDIF
   MESSAGEBOX(vStatus; "Search"; vMessage; IconInformation! | YesNo!)
UNTIL(vStatus = No)
QUIT

LABEL(NotFoundTest)
   BEEP
   MESSAGEBOX(vError; "ERROR"; """ + vStr + """ not found"; IconExclamation!)
   vError := True
   RETURN // return to IF statement following SearchNext command
```

For Example...

```
APPLICATION(A1; "WPOffice"; Default; "US")
// Demonstrate PAUSE command

ONCANCEL(EndLoop)
InhibitInput(Off!)
InformationIcon := 4
x := 1

ViewOpenNamed(ViewName: "Expanded Mail"; ViewType: Mail!)
FocusSet(Place: Message!)

PROMPT("PAUSE EXAMPLE"; "Choose OK to dismiss prompt and continue loop, or Cancel to
end loop."; InformationIcon)
REPEAT
   Type(x + NTOC(32))
   IF(x MOD 10 = 0)
      Type(" - Press OK to continue or Esc to end")
      PAUSE
      Enter
   ENDIF
   x := x + 1
UNTIL(x = 0)

LABEL(EndLoop)
   QUIT
```

For Example...

```
// There are two macros in this example
// Compile separately

APPLICATION(A1; "WPOffice"; Default; "US")
// Name of macro: Persist.wcm
// Demonstrate PERSIST command
// vNmbr is a Persist variable

MESSAGEBOX(x; "PERSISTALL EXAMPLE"; "The value of variable vNmbr is " + vNmbr;
IconInformation!)
ASSIGN(vNmbr; 999)
RETURN

APPLICATION(A1; "WPOffice"; Default; "US")
// Run PERSIST.WCM macro
// Demonstrate PERSISTALL command

LOCAL vStatus; vCancel
vStatus := 4
vCancel := 2

PERSISTALL // vNmbr Persist variable

REPEAT
   IF(EXISTS(vNmbr))
      DISCARD vNmbr
   ENDIF
   GETNUMBER(vNmbr; "Enter a number:"; "PERSISTALL EXAMPLE")
   vMacrosPath := EnvPrefMacroPath
   RUN(vMacrosPath + "persist.wcm")
   MESSAGEBOX(vStatus; "PERSISTALL EXAMPLE"; "After running PERSIST.WCM, the value of
   variable vNmbr is " + vNmbr; IconInformation! | RetryCancel!)
UNTIL(vStatus = vCancel)
```

For Example...

```
APPLICATION(A1; "WPOffice"; Default; "US")
// Saves message box text to file
// Demonstrate SENDKEYS WAIT command

ViewOpenNamed("Expanded Mail"; Mail!)
FocusSet(Message!)
Type("Save this message as ""test.doc"" in the default Save Directory (see
PrefLocationOfFiles).")
WAIT(10)
SENDKEYS WAIT("{Alt+F}{S}test.doc{Enter}")
```

For Example...

```
APPLICATION(A1; "WPOffice"; Default; "US")
// Demonstrate SPEED command

ONCANCEL(QuitMacro)

ViewOpenNamed("Expanded Mail"; Mail!)
FocusSet(Message!)
Type("Press Escape to Cancel")
Enter Enter

FOR(x; 1; x < 51; x + 1)
   Type(x + NTOC(32))
   SPEED(x/10)
   IF(((x MOD 10) = 0) AND (x < 50))
      Slower
   ENDIF
ENDFOR
QuitMacro

LABEL(Slower)
   BEEP
   Enter
   Type("Slower ... ")
   RETURN

LABEL(QuitMacro)
   MESSAGEBOX(x; "SPEED EXAMPLE"; "Counting to 50, SPEED increased the delay between
   counts."; IconInformation!)
   QUIT
```

For Example...

```
APPLICATION(A1; "WPOffice"; Default; "US")
// Test user input for Y or N
// For Cancel condition: chose Cancel, click Close, press Alt+F4, or double-click system menu
box
// Demonstrate STRLEN command

CANCEL(On!)
ONCANCEL CALL(CancelMessage)
PROCEDURE Message(vPrompt; vIcon)
   IF(vIcon = "!")
      BEEP
      MESSAGEBOX(y; "Message Box"; vPrompt; IconExclamation!)
   ELSE
      MESSAGEBOX(y; "Message Box"; vPrompt; IconInformation!)
   ENDIF
ENDPROC

vStr := ""
REPEAT
   GETSTRING(vStr; "Press one character: ""Y"" for Yes or ""N"" for no"; "GETSTRING - STRLEN
   EXAMPLE"; 50)
   IF(vStr = "")
      vStr := "<Enter>"
   ENDIF

   vTest := TOUPPER(vStr)
   IF(STRLEN(vStr) > 1)
         CALL Message("You typed or pressed """ + vStr + """"; "!")
   ELSE
      IF((vTest = "Y") OR (vTest = "N"))
         CALL Message("Well done - You pressed " + vTest; "i")
      ELSE
         CALL Message("You pressed """ + vStr + """"; "!")
      ENDIF
   ENDIF
UNTIL(((vTest = "Y") OR (vTest = "N")) AND NOT(STRLEN(vStr) > 1))
QUIT

LABEL(CancelMessage)
   vStr := "a Cancel control"
   RETURN
```

For Example...

```
APPLICATION(A1; "WPOffice"; Default; "US")
// Extract string of numbers and convert to numeric equivalent
// Demonstrate STRNUM command

vNumbers := ""
vPrompt := "Enter text that includes both letters and numbers:"
LABEL(Start)
GETSTRING(vStr; vPrompt; "EXTRACT NUMBERS")
IF(vStr = "")
   BEEP
   MESSAGEBOX(x; "ERROR"; "You pressed return without entering any text"; IconStop!)
   GO(Start)
ENDIF
vLen := STRLEN(vStr)
vPos := 1
REPEAT
   vTest := SUBSTR(vStr; vPos; 1)
   IF((CTON(vTest) > 47) AND (CTON(vTest) < 58))
      vNumbers := vNumbers + vTest
   ENDIF
   vPos := vPos + 1
UNTIL(vPos = vLen + 1)
IF(STRLEN(vNumbers) > 0)
   vConvert := STRNUM(vNumbers)
   vConvert := vConvert * 5
   MESSAGEBOX(x; "STRNUM COMMAND"; vNumbers + " * 5 equals " + vConvert;
   IconInformation!)
ELSE
   BEEP
   MESSAGEBOX(x; "ERROR"; "You didn't enter any numbers"; IconExclamation!)
ENDIF
```

For Example...

```
APPLICATION(A1; "WPOffice"; Default; "US")
// Extract string of numbers and convert to measurement equivalent
// Demonstrate STRUNIT command

Retry := 4
LABEL(Start)
   vNumbers := ""
   GETSTRING(vStr; "Enter text that includes both letters and numbers:"; "EXTRACT
   NUMBERS")
   IF(vStr = "")
      BEEP
      MESSAGEBOX(x; "ERROR"; "You pressed return without entering any text"; IconStop!)
      GO(Start)
   ENDIF
   vLen := STRLEN(vStr)
   vPos := 1
   REPEAT
      vTest := SUBSTR(vStr; vPos; 1)
      IF((CTON(vTest) > 47) AND (CTON(vTest) < 58))
          vNumbers := vNumbers + vTest
      ENDIF
      vPos := vPos + 1
   UNTIL(vPos = vLen + 1)
   IF(STRLEN(vNumbers) > 0)
      CALL(DefaultUnit)
      vConvert := STRUNIT(vNumbers)
      vConvert := vConvert * 5
      MESSAGEBOX(x; "STRUNIT COMMAND"; vNumbers + " * 5 equals " + vConvert;
      IconInformation! | RetryCancel!)
      IF(x = Retry)
          GO(Start)
      ENDIF
   ELSE
      BEEP
      MESSAGEBOX(x; "ERROR"; "You didn't enter any numbers"; IconExclamation!)
   ENDIF
   QUIT

LABEL(DefaultUnit)
   MENU(vChoice; Digit; ; ; {"Inches"; "Centimeters"; "Millimeters"; "Points"; "WP Units"})
   SWITCH(vChoice)
      CASEOF 1: DEFAULTUNITS(Inches!)
      CASEOF 2: DEFAULTUNITS(Centimeters!)
      CASEOF 3: DEFAULTUNITS(Millimeters!)
      CASEOF 4: DEFAULTUNITS(Points!)
      CASEOF 5: DEFAULTUNITS(WPUnits!)
   ENDSWITCH
   RETURN
```

For Example...

```
APPLICATION(A1; "WPOffice"; Default; "US")
// Extract "Perfect" from WordPerfect and test if "program" was typed
// Demonstrate STRPOS and SUBSTR commands

vNumbers := ""
LABEL(Start)
GETSTRING(vStr; "Enter the following text: ""WordPerfect program"""; "SUBSTR AND STRPOS
EXAMPLES")
IF(vStr = "")
   BEEP
   MESSAGEBOX(x; "ERROR"; "You pressed return without entering any text"; IconStop!)
   GO(Start)
ENDIF
vSub := SUBSTR(vStr; 5; 7)
IF(vSub = "Perfect")
   MESSAGEBOX(x; "SUBSTR EXAMPLE"; "WordPerfect is simply """ + vSub + """";
   IconInformation!)
   IF(STRPOS(vStr; "program") = 0)
      BEEP
      MESSAGEBOX(x; "STRPOS EXAMPLE"; "You entered ""WordPerfect"" but not
      ""program"""; IconExclamation!)
   ENDIF
ELSE
   BEEP
   MESSAGEBOX(x; "ERROR"; """WordPerfect"" wasn't the first word"; IconExclamation!)
   GO(Start)
ENDIF
```

For Example...

```
APPLICATION(A1; "WPOffice"; Default; "US")
// Demonstrate TOLOWER and VARERRCHK command

HdReturn := NTOC(0F90Ah)
VARERRCHK(Off!)

FOR(x; 65; x < 91; x + 1)
   // convert uppercase character to lowercase
   // vLongStr not declared - no value assigned
   vLongStr := vLongStr + TOLOWER(NTOC(x)) + NTOC(32)
ENDFOR

MESSAGEBOX(x; "VARERRCHK and TOLOWER"; "With VARERRCHK(On!), you would receive a
run-time error message." + HdReturn + HdReturn + "vLongStr: "    + vLongStr + HdReturn;
IconInformation!)
```

For Example...

```
APPLICATION(A1; "WPOffice"; Default; "US")
// Demonstrate UNITSTR and DIALOG commands

vNmbr := 1
vStatus := 4
DIALOGDEFINE(1000; 50; 50; 100; 120; Percent! | NoTitle!| NoFrame!; "")
   DIALOGADDTEXT(1000; -1; 10; 15; 75; 15; Left!; "Convert " + vNmbr + " inch to:")
   DIALOGADDCHECKBOX(1000; 101; 15; 30; 100; 13; "Centimeters"; vCB1)
   DIALOGADDCHECKBOX(1000; 102; 15; 45; 100; 13; "Millimeters"; vCB2)
   DIALOGADDCHECKBOX(1000; 103; 15; 60; 100; 13; "Points"; vCB3)
   DIALOGADDCHECKBOX(1000; 104; 15; 75; 100; 13; "WP Units"; vCB4)
   DIALOGADDTEXT(1000; -2; 10; 100; 100; 15; Left!; "(Press Alt+F4 to cancel)")
DIALOGDISPLAY(1000; -1; Msg)

WHILE(vStatus = 4)
ENDWHILE
DIALOGDESTROY(1000)

LABEL(Msg)
   IF(Msg[5] = 274)
      BEEP
      vStatus := 2
   ENDIF
   DEFAULTUNITS(Inches!)
   SWITCH(Msg[3])
      CASEOF 101: vResult := UNITSTR(vNmbr; Centimeters!)
            Message
      CASEOF 102: vResult := UNITSTR(vNmbr; Millimeters!)
            Message
      CASEOF 103: vResult := UNITSTR(vNmbr; Points!)
            Message
      CASEOF 104: vResult := UNITSTR(vNmbr; WPUnits!)
            Message
   ENDSWITCH
   RETURN

LABEL(Message)
   DIALOGUNDISPLAY(1000; "CancelBttn")
   MESSAGEBOX(vStatus; "UNITSTR COMMAND"; vNmbr + " inch is converted to " + vResult;
   IconInformation! | RetryCancel!)
   IF(vStatus = 4)
      DIALOGDISPLAY(1000; -1; Msg)
   ELSE
      BEEP
   ENDIF
   RETURN
```

For Example...

```
// There are two macros in this example
// Compile separately: name the first MACROLIB.WCM
// A Warning Error displayed when macros compiled for the first time
//  For each warning, choose Continue Compilation.
// Demonstrate USE command

APPLICATION(A1; "WPOffice"; Default; "US")
FUNCTION GetMsgBoxResult(x)
   VARERRCHK(Off!)
   BEEP
   SWITCH(x)
      CASEOF 1:     MESSAGEBOX(vChoice; "Macro File Library"; "Choose a button to dismiss
                    the message box"; IconStop! | AbortRetryIgnore!)
      CASEOF 2:     MESSAGEBOX(vChoice; "Macro File Library"; "Choose a button to dismiss
                    the message box"; IconStop! | Ok!)
      CASEOF 3:     MESSAGEBOX(vChoice; "Macro File Library"; "Choose a button to dismiss
                    the message box"; IconStop! | OkCancel!)
      CASEOF 4:     MESSAGEBOX(vChoice; "Macro File Library"; "Choose a button to dismiss
                    the message box"; IconStop! | RetryCancel!)
      CASEOF 5:     MESSAGEBOX(vChoice; "Macro File Library"; "Choose a button to dismiss
                    the message box"; IconStop! | YesNo!)
      CASEOF 6:     MESSAGEBOX(vChoice; "Macro File Library"; "Choose a button to dismiss
                    the message box"; IconStop! | YesNoCancel!)
   ENDSWITCH
   RETURN(vChoice)
ENDFUNC

PROCEDURE DisplayMsgBoxResult(x)
   MESSAGEBOX(x; "Macro File Library"; "You chose the """ + x + """ button";
   IconInformation!)
ENDPROC

APPLICATION(A1; "WPOffice"; Default; "US")
// Demonstrate USE command

VARERRCHK(Off!)
// replace default macros directory if necessary
USE("C:\wpoffice40\macros\macrolib.Wcm")
REPEAT
   MENU(vResult; Digit; ; ; {"AbortRetryIgnore!"; "OK!"; "OKCancel!"; "RetryCancel!";
   "YesNo!"; "YesNoCancel"; "Quit"})
   SWITCH(GetMsgBoxResult(vResult))
      CASEOF 1: vStatus = "OK"
      CASEOF 2: vStatus = "Cancel"
      CASEOF 3: vStatus = "Abort"
      CASEOF 4: vStatus = "Retry"
      CASEOF 5: vStatus = "Ignore"
      CASEOF 6: vStatus = "Yes"
```

```
      CASEOF 7: vStatus = "No"
   ENDSWITCH
   IF(NOT(vResult = 7))
      DisplayMsgBoxResult(vStatus)
   ENDIF
UNTIL(vResult = 7)
```

For Example...

```
APPLICATION(A1; "WPOffice"; Default; "US")
// Demonstrate DIALOGADDCHECKBOX, DIALOGADDCOLORWHEEL, DIALOGADDCOMBOBOX,
and DIALOGADDCOUNTER

CALL(DlgCreate)
REPEAT
MENU(vResult; Digit; ; ; {"Check box"; "Color wheel"; "Combination box"; "Counter"; "Quit"})
   DisplayControls
UNTIL(vResult = 5)
CALL(QuitMacro)

LABEL(DisplayControls)
   x := 0
   SWITCH(vResult)
      CASEOF 1: DIALOGDISPLAY(1000; 100; Msg)
                WHILE(x = 0)
                ENDWHILE
      CASEOF 2: DIALOGDISPLAY(1001; 1)
                vRed := vCW % 256
                vGreen := vCW / 256 % 256
                vBlue := vCW / 256 / 256
                DisplayColorValues(vRed; vGreen; vBlue)
      CASEOF 3: DIALOGDISPLAY(1002; 100)
      CASEOF 4: DIALOGDISPLAY(1003; 100)
   ENDSWITCH
   RETURN

LABEL(Msg)
   IF(Msg[5] = 274)
      DIALOGUNDISPLAY(1000; "CancelBttn")
      x := 1
   ENDIF
   SWITCH(Msg[3])
      CASEOF 100:
          DIALOGUNDISPLAY(1000; "CancelBttn")
          MESSAGEBOX(x; "SEE WHAT YOU CAN DO"; "This check box is used with a callback
          function.   See DIALOGDISPLAY for details."; IconInformation!)
   ENDSWITCH
   RETURN

PROCEDURE DisplayColorValues(r; g; b)
   MESSAGEBOX(x; "COLOR VALUES"; "The red component equals " + r + ". The green
   component equals " + g + ". The blue component equals " + b + "."; IconInformation!)
ENDPROC

LABEL(DlgCreate)
   DIALOGDEFINE (1000; 50; 50; 120; 50; Percent! | NoFrame!; "DIALOGADDCHECKBOX")
      DIALOGADDCHECKBOX(1000; 100; 10; 10; 100; 15; "Click check box"; vCB)
```

```
   DIALOGDEFINE (1001; 50; 50; 125; 130; Percent! | NoFrame! | OK!;
   "DIALOGADDCOLORWHEEL")
      DIALOGADDCOLORWHEEL(1001; 101; 10; 10; 105; 80; vCW)
   DIALOGDEFINE (1002; 50; 50; 125; 115; Percent! | NoFrame!; "DIALOGADDCOMBOBOX")
      DIALOGADDCOMBOBOX(1002; 100; 10; 10; 100; 50; DropDown!; var)
      DIALOGADDLISTITEM(1002; 100; "DropDown! combo box")
      DIALOGADDCOMBOBOX(1002; 101; 10; 40; 100; 50; Simple!; var)
      DIALOGADDLISTITEM(1002; 101; "Simple! combo box")
   DIALOGDEFINE (1003; 50; 50; 120; 50; Percent! | NoFrame!; "DIALOGADDCOUNTER")
   var := 5
      DIALOGADDCOUNTER(1003; 100; 10; 10; 50; 15; 0; var; 1; 10; 1)
   RETURN

LABEL(QuitMacro)
   FORNEXT(x; 1000; 4; 1)
      DIALOGDESTROY(x)
   ENDFOR
   QUIT
```

For Example...

```
APPLICATION(A1; "WPOffice"; Default; "US")
// Demonstrate DIALOGADDCONTROL and SENDKEYS commands

InhibitInput(Off!)
WM_SYSCOMMAND := 274
z := 0
DIALOGDEFINE (1000; 50; 50; 122; 150; Percent! | NoFrame! | OK!; "DIALOGADDCONTROL")
   DIALOGADDCHECKBOX(1000; 100; 10; 10; 75; 15; "Check for demo"; vCB)
   DIALOGADDCONTROL(1000; 101; 10; 40; 100; 75; "wpnslb20"; 2; ""; vListItem; 0)
   A := "Apple"
   B := "Apple pie"
   C := "Applesauce"
   D := "Banana"
   E := "Banana oil"
   F := "Banana seat"
   G := "Banana split"
   FORNEXT(x; 65; 71; 1)
      DIALOGADDLISTITEM(1000; 101; INDIRECT(NTOC(x)))
   ENDFOR
DIALOGDISPLAY(1000; 101; Msg)
WHILE(z = 0)
ENDWHILE
DIALOGDESTROY(1000)

LABEL(Msg)
   IF(Msg[5] = WM_SYSCOMMAND)
      z := 1
   ENDIF
   SWITCH(Msg[3])
      CASEOF 100: Demo
      CASEOF "OKBttn": DIALOGUNDISPLAY(1000; 101)
          IF(STRLEN(vListItem) > 0)
              ASSIGN(vSelect; "You selected """ + vListItem + """ from the list box")
          ELSE
              ASSIGN(vSelect; "You didn't select an item from the list box")
          ENDIF
          MESSAGEBOX(x; "LIST ITEM"; vSelect; IconInformation!)
          z := 1
   ENDSWITCH
   RETURN

LABEL(Demo)
   SENDKEYS WAIT("{Tab} {Up}")
   FOREACH(vLetter; {"A"; "p"; "p"; "l"; "e"; "s"; "a"; "u"; "c"; "e"})
      SENDKEYS WAIT(vLetter)
      WAIT(3)
   ENDFOR
   WAIT(12)
```

```
FORNEXT(w; 1; 10; 1)
   SENDKEYS WAIT("{BackSpace}")
   WAIT(1)
ENDFOR
WAIT(10)
FOREACH(vLetter; {"B"; "a"; "n"; "a"; "n"; "a"; " "; "s"; "p"; "l"; "i"; "t"})
   SENDKEYS WAIT(vLetter)
   WAIT(3)
ENDFOR
WAIT(12)
SENDKEYS WAIT("{Tab}")
MESSAGEBOX(x; "NAME SEARCH LIST BOX"; "Enter a word from the list to experiment with
WordPerfect's custom list box")
SENDKEYS WAIT("{Up}")
RETURN
```

For Example...

```
APPLICATION(A1; "WPOffice"; Default; "US")
// Demonstrate DIALOGADDHOTSPOT and DIALOGADDICON

VARERRCHK(Off!)
Yes := 6
WM_SYSCOMMAND := 274
DialogControl := 3

DIALOGDEFINE (1000; 50; 50; 115; 85; Percent! | NoFrame!; "DIALOGADDICON" )
DIALOGADDTEXT(1000; -1; 10; 10; 50; 13; Left!; "Click an icon:")
FORNEXT(z; 1; 3; 1)
   DIALOGADDICON (1000; 0; 25; z + 10 + (z * 13); 10; 8; "#" + (z+159); 0)
   DIALOGADDHOTSPOT(1000; z; 25; z + 10 + (z * 13); 10; 8; Click!)
ENDFOR

DIALOGDISPLAY(1000; 0; Msg)
WHILE(NOT(vStatus = Yes))
ENDWHILE
DIALOGDESTROY (1000)
QUIT

LABEL(Msg)
   IF(Msg[5] = WM_SYSCOMMAND)
      vStatus = Yes
   ENDIF
   SWITCH(Msg[DialogControl])
      CASEOF 1:    MESSAGEBOX(vStatus; "DIALOGADDHOTSPOT"; "Overlap hot spots and
                   icons to create custom controls."; IconExclamation!)
      CASEOF 2:    MESSAGEBOX(vStatus; "DIALOGADDHOTSPOT"; "See WordPerfect 6.0 for
                   Windows Software Developer's Kit for Shared Code information.";
                   IconInformation!)
      CASEOF 3:    MESSAGEBOX(vStatus; "DIALOGADDHOTSPOT"; "Are you finished with this
                   example?"; IconQuestion! | YesNo!)
   ENDSWITCH
   RETURN
```

For Example...

```
APPLICATION(A1; "WPOffice"; Default; "US")
// Demonstrate DIALOGADDFILENAMEBOX, DIALOGADDEDITBOX, and DIALOGADDVIEWER

VARERRCHK(Off!)
ASSIGN(vText; "Note that DIALOGDEFINE uses Enter2HRtn! style with this edit box control.")
DIALOGDEFINE (1000; 50; 50; 195; 225; OK! | Percent! | Enter2HRtn!;
"DIALOGADDCONTROL")
   DIALOGADDTEXT(1000; -1; 10; 10; 100; 15; Left!; "DIALOGADDFILENAMEBOX")
   DIALOGADDFILENAMEBOX(1000; 100; 10; 20; 170; 15; FilesAndDirs!; vDir; "c:\")
   DIALOGADDTEXT(1000; -2; 10; 40; 100; 15; Left!; "DIALOGADDEDITBOX")
   DIALOGADDEDITBOX(1000; 101; 10; 50; 170; 50; Left! | Multiline! | WordWrap! | VScroll!;
   vText; 1000)
   DIALOGADDTEXT(1000; -3; 10; 105; 100; 15; Left!; "DIALOGADDVIEWER")
   DIALOGADDVIEWER(1000; 102; 10; 115; 170; 60; "C:\AUTOEXEC.BAT")
DIALOGDISPLAY(1000; 100)
DIALOGDESTROY(1000)

IF(STRLEN(vDir) = 0)
   Msg("You did not enter a path or filename")
ELSE
   MESSAGEBOX(x; "DIALOGADDFILENAMEBOX"; "You entered the following path or path and
   filename: " + vDir)
ENDIF
IF(STRLEN(vText) = 0)
   Msg("You did not enter text in the edit control")
ELSE
   MESSAGEBOX(x; "DIALOGADDEDITBOX"; "You entered the following text in the edit control:
   " + NTOC(0F90Ah) + NTOC(0F90Ah) + vText)
ENDIF

PROCEDURE Msg(x)
   BEEP
   MESSAGEBOX(x; "Oops!"; x; IconExclamation!)
ENDPROC
```

For Example...

```
APPLICATION(A1; "WPOffice"; Default; "US")
// Use array SortStr to sort up to eight character strings by length
// Calculate width of dialog box from longest string and height from the number of strings
// Demonstrate DIALOGADDFRAME command

CALL(GetNumberAndStrings)
CALL(GetDimensions)
CALL(DisplaySortedStrings)

LABEL(GetNumberAndStrings)
   GETNUMBER(vNmbr; "Number of strings to sort (up to eight):"; "FRAME EXAMPLE")
   IF(vNmbr > 8)
      vNmbr := 8
   ENDIF
   DECLARE StrSort[vNmbr]
   FORNEXT(x; 1; vNmbr; 1)
      GETSTRING(StrSort[x]; "String " + x + ":"; "SORT " + vNmbr + " STRINGS"; 50)
   ENDFOR
   RETURN

LABEL(GetDimensions)
   vSorted := 1
   WHILE(vSorted = 1)
      vSorted := 0
      FOR(x; 1; x < vNmbr; x + 1)
          IF(STRLEN(StrSort[x]) < STRLEN(StrSort[x + 1]))
              vTemp := StrSort[x]
              StrSort[x] := StrSort[x + 1]
              StrSort[x + 1] := vTemp
              vSorted := 1
          ENDIF
      ENDFOR
   ENDWHILE
   x := 20
   y := 20
   w := STRLEN(StrSort[1]) + (STRLEN(StrSort[1]) * 4)
   h := 13
   RETURN

LABEL(DisplaySortedStrings)
   DIALOGDEFINE(1000; 50; 50; w + 45; 40 + 30 * vNmbr; Percent!; "FRAME CONTROL")
   FORNEXT(r; 1; vNmbr; 1)
      DIALOGADDFRAME(1000; r; x - 5;   y - 5 + (r * 30) - 30; w + 10; h + 7; Frame! | Gray!)
      DIALOGADDTEXT(1000; 0; x; y + (r * 30) - 30; w; h; Center!; TOUPPER(StrSort[r]))
   ENDFOR
   DIALOGDISPLAY(1000; 1)
   DIALOGDESTROY(1000)
   QUIT
```

For Example...

```
APPLICATION(A1; "WPOffice"; Default; "US")
// Demonstrate group controls and buttons

VARERRCHK(Off!)
DIALOGDEFINE(1000; 50; 50; 200; 140; Percent!; "GROUP CONTROLS AND BUTTONS")
   DIALOGADDGROUPBOX(1000; -1; 10; 10; 75; 55; "Radio Buttons")
   DIALOGADDRADIOBUTTON(1000; 100; 25; 25; 50; 15; "Example 1"; vRadio1)
   DIALOGADDRADIOBUTTON(1000; 101; 25; 45; 50; 15; "Example 2"; vRadio2)
   DIALOGADDVLINE(1000; -2; 125; 15; 55)
   DIALOGADDPOPUPBUTTON(1000; 102; 135; 35; 50; 13; vPopup)
   DIALOGADDLISTITEM(1000; 102; "Popup 1")
   DIALOGADDLISTITEM(1000; 102; "Popup 2")
   DIALOGADDLISTITEM(1000; 102; "Popup 3")
   DIALOGADDHLINE(1000; -3; 10; 85; 175)
   DIALOGADDPUSHBUTTON(1000; 103; 15; 95; 165; 13; DefaultBttn!; "Cancel")
DIALOGDISPLAY(1000; 103)

IF(vRadio1 <> 0)
   vMessage := "You selected radio button 1. "
ELSE
   IF(vRadio2 <> 0)
      vMessage := "You selected radio button 2. "
   ELSE
      vMessage := "You didn't select a radio button. "
   ENDIF
ENDIF
vMessage := vMessage + "You selected " + vPopup + ". "
IF(MacroDialogResult = 103)
   vMessage := vMessage + "You chose Cancel to close the dialog box."
   MESSAGEBOX(x; "MESSAGE BOX"; vMessage; IconInformation!)
ELSE
   BEEP
   MESSAGEBOX(x; "MESSAGE BOX"; "You chose a system command to close the dialog box";
   IconExclamation!)
ENDIF
DIALOGDESTROY(1000)
```

For Example...

```
APPLICATION(A1; "WPOffice"; Default; "US")
// Demonstrate DIALOGADDLISTBOX and DIALOGADDLISTITEM

DECLARE Fruit[5]
y := 10
DIALOGDEFINE(1000; 50; 50; 122; 155; OK! | Percent! | NoFrame!; "SORTED LIST BOX")
   FOREACH(vList; {"Apples"; "Oranges"; "Bananas"; "Grapes"; "Pears"})
      DIALOGADDTEXT(1000; 0; 10; y; 30; 15; Left!; vList)
      DIALOGADDHOTSPOT(1000; y; 10; y; 30; 9; Click!)
      FRUIT[y/10] := vList
      y := y + 10
   ENDFOR
   DIALOGADDLISTBOX(1000; 103; 50; 10; 60; 50; Sorted!; vChoice)
   DIALOGADDTEXT(1000; 0; 10; 75; 102; 35; Left! | RecessBox!; "Click a ""fruit"" to display
   in the list box.   Click a list box item and choose OK when finished.")
DIALOGDISPLAY(1000; 102; Msg)

Stop   := 0
WHILE(Stop = 0)
ENDWHILE
DIALOGDESTROY(1000)

IF(vChoice = "")
   vMessage := "You didn't select a list box item " + vChoice
ELSE
   vMessage := "You selected " + vChoice
ENDIF
MESSAGEBOX(x; "LIST BOX CHOICE"; vMessage; IconInformation!)
QUIT

LABEL(Msg)
   IF(Msg[5] = 274)
      Stop := 1
   ENDIF
   SWITCH(Msg[3])
      CASEOF 10: DIALOGADDLISTITEM(1000; 103; Fruit[1])
      CASEOF 20: DIALOGADDLISTITEM(1000; 103; Fruit[2])
      CASEOF 30: DIALOGADDLISTITEM(1000; 103; Fruit[3])
      CASEOF 40: DIALOGADDLISTITEM(1000; 103; Fruit[4])
      CASEOF 50: DIALOGADDLISTITEM(1000; 103; Fruit[5])
      CASEOF "OKBttn":
               DIALOGUNDISPLAY(1000; "OKBttn")
               Stop := 1
   ENDSWITCH
   RETURN
```

For Example...

```
APPLICATION(A1; "WPOffice"; Default; "US")
// Demonstrate DIALOGADDSCROLLBAR command with callback function

SB_LINEUP := 0
SB_LINEDOWN := 1
WM_SYSCOMMAND := 274
WM_HSCROLL := 276

DIALOGDEFINE(1000; 50; 50; 125; 70; Percent!; "SCROLL BAR")
   DIALOGADDTEXT(1000; -1; 10; 10; 100; 15; Center! | RecessBox!; "Click the left or right
   arrow")
   DIALOGADDSCROLLBAR(1000; "Scroll"; 10; 30; 100; 0; Left! | HScroll!; BarPos; 1; 3)

DIALOGDISPLAY(1000; "Scroll"; Msg)
Stop := 1
WHILE(Stop = 1)
ENDWHILE
DIALOGDESTROY(1000)
QUIT

LABEL(Msg)
   SWITCH(Msg[5])
      CASEOF WM_SYSCOMMAND: Stop := 0
      CASEOF WM_HSCROLL: CALL(Scroll)
   ENDSWITCH
   RETURN

LABEL(Scroll)
   SWITCH(Msg[6])
      CASEOF SB_LINEUP:
               MESSAGEBOX(x; "CALLBACK FUNCTION"; "You clicked the left arrow";
               IconInformation!)
      CASEOF SB_LINEDOWN:
               MESSAGEBOX(x; "CALLBACK FUNCTION"; "You clicked the right arrow";
               IconInformation!)
   ENDSWITCH
   RETURN
```

For Example...

```
APPLICATION(A1; "WPOffice"; Default; "US")
// Returns the handle of five push button controls
// Demonstrate DIALOGHANDLE command

VARERRCHK(Off!)
DIALOGDEFINE(1000; 50; 50; 100; 190; Percent!; "DIALOGHANDLE")
   FORNEXT(x; 1; 5; 1)
      DIALOGADDPUSHBUTTON(1000; x; 10; x + (x * 20) - 10; 75; 15; NonDefaultBttn!;
      "Button " + x)
   ENDFOR
   DIALOGADDTEXT(1000; -1; 10; 120; 75; 40; RecessBox! | Left!; "Press Enter, or choose
   Close from the system menu box.")
DIALOGDISPLAY(1000; 1)
vResult := ""
FOREACH(vControl; {"1"; "2"; "3"; "4"; "5"})
   vResult := vResult + "Button " + vControl + ": "
   DIALOGHANDLE(vHandles; 1000; vControl)
   vResult := vResult + vHandles + ".   "
ENDFOR
MESSAGEBOX(x; "CONTROL HANDLES"; vResult)
DIALOGDESTROY(1000)
```

For Example...

```
APPLICATION(A1; "WPOffice"; Default; "US")
// Purpose: Use BIFWrite to set the default macros directory

ONCANCEL(CheckDirectoryName)

LABEL(Start)
   BIFREAD(vStatus; "WPOffice"; "Files Locations"; "BBarMacros"; vMacroDir; AnsiString!)
   CALL(CheckStatus)
   CALL(WriteToBIF)
   CALL(CheckDirectoryName)

LABEL(CheckStatus)
   SWITCH(vStatus)
      CASEOF(-1): vMessage := "Item types do not match"
      CASEOF(-2): vMessage := "Length of Goup, Section, or Item exceeds buffer size"
      CASEOF(-3): vMessage := "Requested Group, Section, or Item not found"
      CASEOF(-4): vMessage := "Unknown error occurred"
      DEFAULT: RETURN
   ENDSWITCH
   BEEP
   MESSAGEBOX(x; "ERROR"; vMessage; IconExclamation!)
   QUIT

LABEL(WriteToBIF)
   GETSTRING(vStr; "Current directory: " + TOUPPER(vMacroDir) + NTOC(0F90Ah) + "Length:
   " + vStatus + " bytes" + NTOC(0F90Ah)   + NTOC(0F90Ah) + "Type the path and name of
   a new macros directory:"; "Default Macros Directory")
   IF(vStr = "")
      ASSERT(CancelCondition!)
   ENDIF
   BIFWRITE(vStatus; "WPOffice"; "Files Locations"; "BBarMacros"; vStr; AnsiString!)
   RETURN

LABEL(CheckDirectoryName)
   BEEP
   IF(NOT(vStr = ""))
      vDir := vStr
   ELSE
      vDir := vMacroDir
   ENDIF
   MESSAGEBOX(x; "Default Macros Directory"; "The default macros directory is " +
   TOUPPER(vDir); IconStop! | RetryCancel!)
   IF(x = 4)
      GO(Start)
   ELSE
      QUIT
   ENDIF
```

For Example...

```
APPLICATION(A1; "WPOffice"; Default; "US")
// Create a BIF file with five groups, sections, items, and item values
// Demonstrate BIFWRITE command

VARERRCHK(Off!)
HdRtrn := NTOC(0F90Ah)
vMacroPath := EnvPrefMacroPath
vFile := vMacroPath + "test.bif"

FORNEXT(x; 1; 5; 1)
   BIFWRITE(vStatus; "Group " + x; "Section " + x; "Item " + x; "Value " + x; WPWordString!;
   0; vFile)
ENDFOR

ASSIGN(hBIF; APPLOCATE("BIF Edit*"))
IF(hBIF)
   APPACTIVATE(hBIF)
ELSE
   ASSIGN(vResult; APPEXECUTEEXT("c:\wpc20\bifed20.exe"; 1))
ENDIF
IF(vResult > 31)
   // replace default macros directory if necessary
   SENDKEYS WAIT("{Alt + F}Oc:\wpcorp\houston\macros\test.bif {Enter}{Enter}{Down}
   {Enter}{Down}")
   MESSAGEBOX(x; "BIFWRITE COMMAND"; "After you close the message box, double-click
   another" + HdRtrn + "Group, Section, or Item." + HdRtrn + HdRtrn + "Click Help for more
   information about BIF Edit."; IconInformation!)
ELSE
   BEEP
   MESSAGEBOX(x; "ERROR"; "Cannot execute BIFED20.EXE.   Check the path and modify if
   necessary."; IconExclamation!)
   QUIT
ENDIF
```

For Example...

```
APPLICATION(A1; "WPOffice"; Default; "US")
// This example uses the BIF created with the BIFWRITE example
// Demonstrate BIFREAD command

ONCANCEL(QuitMacro)
VARERRCHK(Off!)
REPEAT
   GETNUMBER(vNmbr; "Type the number (1-5) of a group to read:" "BIFREAD COMMAND")
   IF((vNmbr > 5) OR (vNmbr < 1))
      BEEP
      NEXT
   ENDIF
   // replace default macros directory if necessary
   BIFREAD(vResult; "Group " + vNmbr; "Section " + vNmbr; "Item " + vNmbr; vItemValue;
   WPWordString!; ; ; "c:\wpoffice40\macros\test.bif")
   IF(vResult < 0)
      BEEP
      MESSAGEBOX(x; "ERROR"; "Unknown error occurred"; IconExclamation!)
      QUIT
   ELSE
      MESSAGEBOX(x; "Item Value"; "The value of Group, Section, and Item " + vNmbr + " is:
      " + vItemValue; IconInformation! | RetryCancel!)
   ENDIF
UNTIL(x = 2)

LABEL(QuitMacro)
   QUIT
```

For Example...

```
APPLICATION(A1; "WPOffice"; Default; "US")
// Demonstrate BIFFILEPATH command

HdRtrn := NTOC(0F90Ah)

BIFFILEPATH(vPrivate; Private!)
BIFFILEPATH(vPublic; Public!)
MESSAGEBOX(x; "BIFFILEPATH COMMAND"; "Private BIF path and filename: " + vPrivate +
HdRtrn + HdRtrn + "Public BIF path and filename: " + vPublic; IconInformation!)
```

For Example...

```
APPLICATION(A1; "WPOffice"; Default; "US")
// This example uses the BIF created with the BIFWRITE example
// Demonstrate BIFINFO command

VARERRCHK(Off!)
HdRtrn := NTOC(0F90Ah)

BIFINFO(vStatus; "Group 1"; "Section 1"; "Item 1"; vDate; vFlags; vLength; vType; vCount; ;
"c:\wpoffice40\macros\test.bif")

MESSAGEBOX(x; "BIFINFO COMMAND - Test.bif file"; "Date/time created: " + vDate + HdRtrn
+ HdRtrn + "Flags: " + vFlags + HdRtrn + HdRtrn + "Length of item: " + vLength + HdRtrn
+ HdRtrn + "Item type: " + vType + HdRtrn + HdRtrn + "Count: " + vCount + HdRtrn)
```

For Example...

```
// Create a DOS or Windows dialog box
// Demonstrate IFPLATFORM command

// compile on DOS platform
IFPLATFORM(DOS)
   DLGCREATE(x; "TITLE"; DLGNoCancel!;;;30;14)
      DLGCONTROL(CtrlRadioButton!;a;"Letter";StyInitial!;4;4;;2)
      DLGCONTROL(CtrlRadioButton!;b;"Itinerary";;4;6;;2)
      DLGCONTROL(CtrlRadioButton!;c;"Memo";;4;8;;2)
      DLGCONTROL(CtrlRadioButton!;d;"Fax";;4;10;;2)
   DLGEND

   SWITCH(1)
      CASEOF a: BEEP
      CASEOF b: BEEP
      CASEOF c: BEEP
      CASEOF d: BEEP
   ENDSWITCH
ENDIFPLATFORM

// compile on Windows platform
IFPLATFORM(WIN)
   APPLICATION(A1; "WPOffice"; Default; "US")
   DIALOGDEFINE(1000; 50; 50; 100; 125; Style:OK!; Caption:"TITLE")
      DIALOGADDRADIOBUTTON(1000; 1; 10; 10; 75; 15; ButtonText:"Letter"; a)
      DIALOGADDRADIOBUTTON(1000; 2; 10; 25; 75; 15; ButtonText:"Itinerary"; b)
      DIALOGADDRADIOBUTTON(1000; 3; 10; 40; 75; 15; ButtonText:"Memo"; c)
      DIALOGADDRADIOBUTTON(1000; 4; 10; 55; 75; 15; ButtonText:"Fax"; d)
   DIALOGDISPLAY(1000; 1)

   SWITCH(1)
      CASEOF a: vMsg := "Letter"
      CASEOF b: vMsg := "Itinerary"
      CASEOF c: vMsg := "Memo"
      CASEOF d: vMsg := "Fax"
   ENDSWITCH
   MESSAGEBOX(x; "Selection"; vMsg; IconInformation!)
   DIALOGDESTROY(1000)
ENDIFPLATFORM
```

For Example...

```
APPLICATION(A1; "WPOffice"; Default; "US")
// Compiling this macro produces a compile-time syntax error
// To compile, make the second AboutDlg a comment (//)
// Demonstrate ENDAPP command

AboutDlg

ENDAPP(A1)

AboutDlg
```

MENULIST

This command is included for DOS compatibility. The windows equivalent is MENU.

NEST

This command is included for DOS compatibility. The windows equivalent is RUN.

For Example...

```
APPLICATION(WP; "WPOffice"; Default; "US")
// Demonstrate DIALOGDISPLAY command

HRt = NTOC(0F90Ah)
HSp = "       "
vCombo = "Combo item 1"
vList = "List item 1"
WM_COMMAND = 273
WM_SYSCOMMAND = 274

DIALOGDEFINE(1000; 50; 50; 160; 125; Percent! | NoFrame! | Cancel!; "WM_COMMAND")
   DIALOGADDLISTBOX(1000; 100; 10; 10; 60; 30; Sorted!; vList)
      DIALOGADDLISTITEM(1000; 100; "List item 1")
      DIALOGADDLISTITEM(1000; 100; "List item 2")
      DIALOGADDLISTITEM(1000; 100; "List item 3")
   DIALOGADDCOMBOBOX(1000; 101; 10; 50; 60; 40; Droplist!; vCombo; 100)
DIALOGADDLISTITEM(1000; 101; "Combo item 1")
      DIALOGADDLISTITEM(1000; 101; "Combo item 2")
      DIALOGADDLISTITEM(1000; 101; "Combo item 3")
   DIALOGADDPOPUPBUTTON(1000; 102; 10; 80; 60; 15; vPopup)
      DIALOGADDLISTITEM(1000; 102; "Popup item 1")
      DIALOGADDLISTITEM(1000; 102; "Popup item 2")
      DIALOGADDLISTITEM(1000; 102; "Popup item 3")
   DIALOGADDPUSHBUTTON(1000; 103; 100; 10; 50; 15; NonDefaultBttn!; "Push Button")
   DIALOGADDRADIOBUTTON(1000; 104; 100; 35; 50; 15; "Radio"; vRadio)
   DIALOGADDCHECKBOX(1000; 105; 100; 60; 50; 15; "Check box"; vCheck)
   DIALOGHANDLE(hButton; 1000; 103)
   DIALOGHANDLE(hRadio; 1000; 104)
   DIALOGHANDLE(hCheck; 1000; 105)
DIALOGDISPLAY(1000; 1; Msg)
x := 0
While(x = 0)
Endwhile
DIALOGDESTROY(1000)

LABEL(MSG)
   IF((Msg[5] = WM_SYSCOMMAND) OR (Msg[3] = "CancelBttn"))
      BEEP
      x = 1
      RETURN
   ENDIF
   IF(Msg[5] = WM_COMMAND)
         lParam = Msg[7]
      DIALOGUNDISPLAY(1000; 1)
         MsgBox
      DIALOGDISPLAY(1000; 1; Msg)
   ENDIF
   RETURN
```

```
LABEL(MsgBox)
   SWITCH(LOWRD(lParam))
      CASEOF hButton: vHandle = hButton
      CASEOF hRadio: vHandle = hRadio
      CASEOF hCheck: vHandle = hCheck
      DEFAULT: vHandle = "N/A"
   ENDSWITCH
   Lo = "Low-word: " + LOWRD(lParam) + HRt + HRt
   Hi = "High-word: " + HIWRD(lParam) + HRt + HRt
   Handle = "Control handle: " + vHandle + HRt + HRt
   ListItems = "Selected list items:" + HRt + HRt + HSp + vPopup + HRt + HSp + vList +
   HRt + HSp + vCombo + HRt
   MESSAGEBOX(vResult; "WM_COMMAND"; Hi + Lo + Handle + ListItems)
   RETURN

FUNCTION LOWRD(lo)
   lo = lo & 65535
   Return(lo)
ENDFUNC

FUNCTION HIWRD(hi)
   hi = hi >> 16
   RETURN(hi)
ENDFUNC
```

For Example...

Example not available at this time.

For Example...

```
APPLICATION(WP; "WordPerfect"; Default; "US")
// MACWRITE.WCM
// Start up process for writing a GroupWise macro in WordPerfect 6.0a

ONERROR(Stop)

CALL(Font)
CALL(Margins)
CALL(Setup)
QUIT

LABEL(Font)
   Display(On!)
   FileNew
   Font(Name: "Arial Regular"; Family: FamilyHelvetica!; Attributes: FontMatchNormal!;
   Weight: 90; Width: WidthUnknown!; Source: DRSFile!; Type: TrueType!; CharacterSet:
   FontMatchASCII!)
   FontSize(11p)
   RETURN

LABEL(Margins)
   MarginTop(0.5")
   MarginBottom(0.5")
   MarginLeft(0.5")
   MarginRight(0.5")
   RETURN

LABEL(Setup)
   MacroEditControlBar
   ViewDraft
   ZoomToMarginWidth
   Type("APPLICATION(WP; ""WPOffice""; Default; ""US"")")
   HardReturn
   Type("// Purpose: ")
   FileSaveAsDlg
   RETURN

LABEL(Stop)
   BEEP
   MESSAGEBOX(x;"WARNING";"Did you save the macro?"; IconExclamation!)
   QUIT
```

For Example...

```
APPLICATION(WP; "WPOffice"; Default; "US")
// Demonstrate DIMENSIONS command

HRt = NTOC(0F90Ah)
DECLARE array[4; 2; 5]
v1 = DIMENSIONS(array[])
v2 = DIMENSIONS(array[]; 0)
v3 = DIMENSIONS(array[]; 1)
v4 = DIMENSIONS(array[]; 2)
v5 = DIMENSIONS(array[]; 3)
v6 = DIMENSIONS(array[]; 4)
v7 = DIMENSIONS(c)
v8 = DIMENSIONS(b[])
vMsg = v1 + HRt + v2 + HRt + v3 + HRt + v4 + HRt + v5 + HRt + v6 + HRt + v7 + HRt +
v8

MESSAGEBOX(x; "DIMENSIONS"; "Variables v1 - v8 values:" + Hrt + Hrt + vMsg)
```

For Example...

```
APPLICATION(WP; "WPOffice"; Default; "US")
// Demonstrate REGIONADDLISTITEM command

Loop = 1
ListItem = 1
WM_SYSCOMMAND = 274

DIALOGDEFINE(1000; 50; 50; 107; 125; Percent! | NoFrame!; "RegionAddListItem")
DIALOGADDFRAME(1000; 100; 10; 10; 10; 10; Frame! | Filled! | Gray!)
DIALOGADDHOTSPOT(1000; 101; 10; 10; 10; 10; Click!)
DIALOGADDTEXT(1000; 0; 28; 10; 75; 15; Left!; "Click to add list item")
DIALOGADDLISTBOX(1000; 102; 10; 35; 85; 75; Sorted!; var)
DIALOGDISPLAY(1000; 101; Msg)

WHILE(Loop = 1)
ENDWHILE
DIALOGDESTROY(1000)
QUIT

LABEL(Msg)
   IF(Msg[5] = WM_SYSCOMMAND)
      Loop = 0
   ENDIF
   IF(Msg[3] = 101)
      AddListBoxItem
   ENDIF
   RETURN

LABEL(AddListBoxItem)
   vItem = "Item " + ListItem
   REGIONADDLISTITEM("1000.102"; vItem)
   ListItem = ListItem + 1
   RETURN
```

For Example...

```
APPLICATION(WP; "WPOffice"; Default; "US")
// Command: REGIONENABLEWINDOW

Loop = 1
WM_SYSCOMMAND = 274
vDisable = 1

DIALOGDEFINE(1000; 50; 50; 155; 75; Percent! | NoFrame!; "RegionEnableWindow")
DIALOGADDFRAME(1000; 100; 10; 10; 10; 10; Frame! | Filled! | Gray!)
DIALOGADDHOTSPOT(1000; 101; 10; 10; 10; 10; Click!)
DIALOGADDTEXT(1000; 0; 28; 10; 125; 15; Left!; "Click to disable/enable check box")
DIALOGADDCHECKBOX(1000; 102; 55; 30; 50; 15; "Check box"; var)
DIALOGDISPLAY(1000; 101; Msg)

WHILE(Loop = 1)
ENDWHILE
DIALOGDESTROY(1000)
QUIT

LABEL(Msg)
   IF(Msg[5] = WM_SYSCOMMAND)
      Loop = 0
   ENDIF
   IF(Msg[3] = 101)
      IF(vDisable = 1)
         REGIONENABLEWINDOW("1000.102"; Disable!)
         vDisable = 0
      ELSE
         REGIONENABLEWINDOW("1000.102"; Enable!)
         vDisable = 1
      ENDIF
   ENDIF
   RETURN
```

For Example...

```
APPLICATION(WP; "WPOffice"; Default; "US")
// Demonstrate REGIONGETCHECK command

Loop = 1
WM_SYSCOMMAND = 274

DIALOGDEFINE(1000; 20; 30; 135; 75; NoFrame!; "RegionGetCheck")
DIALOGADDFRAME(1000; 100; 10; 10; 10; 10; Frame! | Filled! | Gray!)
DIALOGADDHOTSPOT(1000; 101; 10; 10; 10; 10; Click!)
DIALOGADDTEXT(1000; 0; 28; 10; 125; 15; Left!; "Click to get check box state")
DIALOGADDCHECKBOX(1000; 102; 30; 35; 20; 15; "1"; var)
DIALOGADDCHECKBOX(1000; 103; 60; 35; 20; 15; "2"; var)
DIALOGADDCHECKBOX(1000; 104; 90; 35; 20; 15; "3"; var)
DIALOGDISPLAY(1000; 101; Msg)

WHILE(Loop = 1)
ENDWHILE
DIALOGDESTROY(1000)
QUIT

LABEL(Msg)
   IF(Msg[5] = WM_SYSCOMMAND)
      Loop = 0
   ENDIF
   IF(Msg[3] = 101)
      REGIONGETCHECK(vChk1; "1000.102")
      REGIONGETCHECK(vChk2; "1000.103")
      REGIONGETCHECK(vChk3; "1000.104")
      FORNEXT(x; 1; 3; 1)
          IF(INDIRECT("vChk" + x) = 1)
              ASSIGN(INDIRECT("vChk" + x); "selected")
          ELSE
              ASSIGN(INDIRECT("vChk" + x); "not selected")
          ENDIF
      ENDFOR
      MESSAGEBOX(vYesNo; "Check Box State"; "1: " + vChk1 + NTOC(0F90Ah) + "2: " +
      vChk2 + NTOC(0F90Ah) + "3: " + vChk3)
   ENDIF
   RETURN
```

For Example...

```
APPLICATION(WP; "WPOffice"; Default; "US")
// Demonstrate REGIONGETSELECTEDTEXT command

Loop = 1
lParam = 7
WM_SYSCOMMAND = 274

DIALOGDEFINE(1000; 20; 30; 107; 125; NoFrame!; "RegionGetSelectedText")
DIALOGADDTEXT(1000; 0; 0; 15; 107; 15; Center!; "Select a fruit")
DIALOGADDLISTBOX(1000; 102; 10; 35; 85; 75; Sorted!; var)
FOREACH(vList; {"Apples"; "Bananas"; "Oranges"; "Grapes"; "Pears"; "Corn"})
   DIALOGADDLISTITEM(1000; 102; vList)
ENDFOR
DIALOGHANDLE(hItem; 1000; 102)
DIALOGDISPLAY(1000; 102; Msg)

WHILE(Loop = 1)
ENDWHILE
DIALOGDESTROY(1000)
QUIT

LABEL(Msg)
   IF(Msg[5] = WM_SYSCOMMAND)
      Loop = 0
   ENDIF

   IF(LWORD(Msg[lParam]) = hItem)
      REGIONGETSELECTEDTEXT(vSelectItem; "1000.102")
      IF(vSelectItem = "Corn")
          BEEP
          MESSAGEBOX(z; "Error"; "You didn't select a fruit!"; IconExclamation!)
      ELSE
          MESSAGEBOX(z; "List Box Item"; "You selected " + vSelectItem)
      ENDIF

   ENDIF
   RETURN

FUNCTION LWORD(vNum)
   vNum = vNum & 65535
   RETURN(vNum)
ENDFUNC
```

For Example...

```
APPLICATION(WP; "WPOffice"; Default; "US")
// Demonstrate REGIONAGETWINDOWTEXT command

Loop = 1
WM_SYSCOMMAND = 274

DIALOGDEFINE(1000; 50; 50; 158; 50; Percent! | NoFrame!; "RegionGetWindowText")
DIALOGADDFRAME(1000; 100; 10; 10; 10; 10; Frame! | Filled! | Gray!)
DIALOGADDHOTSPOT(1000; 101; 10; 10; 10; 10; Click!)
DIALOGADDTEXT(1000; 0; 28; 10; 125; 15; Left!; "Click to retrieve the caption bar title")
DIALOGDISPLAY(1000; 101; Msg)

WHILE(Loop = 1)
ENDWHILE
DIALOGDESTROY(1000)
QUIT

LABEL(Msg)
   IF(Msg[5] = WM_SYSCOMMAND)
      Loop = 0
   ENDIF
   IF(Msg[3] = 101)
      GetTitle
   ENDIF
   RETURN

LABEL(GetTitle)
   REGIONGETWINDOWTEXT(vTitle; "1000")
   MESSAGEBOX(x; "Region Commands"; "The caption bar title is " + vTitle; IconInformation!)
   Loop = 0
   RETURN
```

For Example...

```
APPLICATION(WP; "WPOffice"; Default; "US")
// Demonstrate REGIONMOVEWINDOW command

Loop = 1
WM_SYSCOMMAND = 274

DIALOGDEFINE(1000; 20; 40; 135; 75; NoFrame!; "RegionMoveWindow")
DIALOGADDCHECKBOX(1000; 101; 10; 10; 75; 15; "Click to start demo"; var)
DIALOGDISPLAY(1000; 101; Msg)

WHILE(Loop = 1)
ENDWHILE
DIALOGDESTROY(1000)
QUIT

LABEL(Msg)
   IF(Msg[5] = WM_SYSCOMMAND)
      Loop = 0
   ENDIF
   IF(Msg[3] = 101)
                  REGIONSHOWWINDOW("1000.101"; Hide!)
      FORNEXT(x; 50; 190; 10)
          REGIONMOVEWINDOW("1000"; 20; x; 135; 75)
      WAIT(.5)
      ENDFOR
      FORNEXT(x; 50; 190; 10)
          REGIONMOVEWINDOW("1000"; x; 190; 135; 75)
      WAIT(.5)
      ENDFOR
      FORNEXT(x; 190; 40; -10)
          REGIONMOVEWINDOW("1000"; 200; x; 135; 75)
      WAIT(.5)
      ENDFOR
      FORNEXT(x; 190; 40; -10)
          REGIONMOVEWINDOW("1000"; x; 40; 135; 75)
      WAIT(.5)
      ENDFOR
      WAIT(.5)
      REGIONMOVEWINDOW("1000"; 20; 40; 300; 225)
      DIALOGADDTEXT(1000; 102; 0; 15; 300; 15; Center!; "Finished")
      FORNEXT(x; 20; 100; 5)
          REGIONMOVEWINDOW("1000.102"; 0; x; 300; 15)
      WAIT(.9)
      ENDFOR
   ENDIF
   RETURN
```

For Example...

```
APPLICATION(WP; "WPOffice"; Default; "US")
// Demonstrate REGIONREMOVELISTITEM command

Loop = 1
lParam = 7
WM_SYSCOMMAND = 274

DIALOGDEFINE(1000; 50; 50; 107; 125; Percent! | NoFrame!; "RegionRemoveListItem")
DIALOGADDTEXT(1000; 0; 0; 15; 107; 15; Center!; "Select a fruit to delete")
DIALOGADDLISTBOX(1000; 102; 10; 35; 85; 75; Sorted!; var)
FOREACH(vList; {"Apples"; "Bananas"; "Oranges"; "Grapes"; "Pears"; "Watermelon";
"Grapefruit"; "Peaches"})
   DIALOGADDLISTITEM(1000; 102; vList)
ENDFOR
DIALOGHANDLE(hItem; 1000; 102)
DIALOGDISPLAY(1000; 102; Msg)

WHILE(Loop = 1)
ENDWHILE
DIALOGDESTROY(1000)
QUIT

LABEL(Msg)
   IF(Msg[5] = WM_SYSCOMMAND)
      Loop = 0
   ENDIF
   IF(LWORD(Msg[lParam]) = hItem)
      REGIONGETSELECTEDTEXT(vSelectItem; "1000.102")
      REGIONREMOVELISTITEM("1000.102"; vSelectItem)
   ENDIF
   RETURN

FUNCTION LWORD(vNum)
   vNum = vNum & 65535

   RETURN(vNum)
ENDFUNC
```

For Example...

```
APPLICATION(WP; "WPOffice"; Default; "US")
// Demonstrate REGIONRESETLIST command

Loop = 1
Yes = 6
WM_SYSCOMMAND = 274

DIALOGDEFINE(1000; 20; 20; 107; 125; NoFrame!; "RegionResetList")
DIALOGADDFRAME(1000; 100; 10; 10; 10; 10; Frame! | Filled! | Gray!)
DIALOGADDHOTSPOT(1000; 101; 10; 10; 10; 10; Click!)
DIALOGADDTEXT(1000; 0; 28; 10; 75; 15; Left!; "Click to clear list")
DIALOGADDLISTBOX(1000; 102; 10; 35; 85; 75; Sorted!; var)
   CALL(AddItems)
DIALOGDISPLAY(1000; 101; Msg)

WHILE(Loop = 1)
ENDWHILE
DIALOGDESTROY(1000)
QUIT

LABEL(Msg)
   IF(Msg[5] = WM_SYSCOMMAND)
      Loop = 0
   ENDIF
   IF(Msg[3] = 101)
      REGIONRESETLIST("1000.102")
      MESSAGEBOX(vResult; "Macro Example"; "Do you want to start over?"; IconQuestion! |
      YesNo!)
      IF(vResult = Yes)
          CALL(AddItems)
      ELSE
          Loop = 0
      ENDIF
   ENDIF

   RETURN

LABEL(AddItems)
   FOREACH(vList; {"Apples"; "Bananas"; "Oranges"; "Grapes"; "Pears"})
      DIALOGADDLISTITEM(1000; 102; vList)
   ENDFOR
   RETURN
```

For Example...

```
APPLICATION(WP; "WPOffice"; Default; "US")
// Demonstrate REGIONSELECTLISTITEM command

Loop = 1
lParam = 7
WM_SYSCOMMAND = 274

DIALOGDEFINE(1000; 50; 50; 205; 125; Percent! | NoFrame!; "RegionSelectListItem")
DIALOGADDTEXT(1000; 0; 0; 15; 107; 15; Center!; "Select a fruit")
DIALOGADDLISTBOX(1000; 100; 11; 35; 85; 75; Sorted!; var)
DIALOGADDLISTBOX(1000; 101; 108; 35; 85; 75; UnSorted!; var)
FOREACH(vList; {"Oranges"; "Grapes"; "Apples"; "Pears"; "Watermelon"; "Grapefruit";
"Peaches"; "Bananas"})
   DIALOGADDLISTITEM(1000; 100; vList)
   DIALOGADDLISTITEM(1000; 101; vList)
ENDFOR
DIALOGHANDLE(hItem; 1000; 100)
DIALOGDISPLAY(1000; 100; Msg)

WHILE(Loop = 1)
ENDWHILE
DIALOGDESTROY(1000)
QUIT

LABEL(Msg)
   IF(Msg[5] = WM_SYSCOMMAND)
      Loop = 0
   ENDIF
   IF(LWORD(Msg[lParam]) = hItem)
      REGIONGETSELECTEDTEXT(vSelectItem; "1000.100")
      REGIONSELECTLISTITEM("1000.101"; vSelectItem)
   ENDIF
   RETURN


FUNCTION LWORD(vNum)
   vNum = vNum & 65535
   RETURN(vNum)
ENDFUNC
```

For Example...

```
APPLICATION(WP; "WPOffice"; Default; "US")
// Demonstrate REGIONSETCHECK command

Loop = 1
WM_SYSCOMMAND = 274
vCheck = 1

DIALOGDEFINE(1000; 50; 50; 160; 75; Percent! | NoFrame!; "RegionSetCheck")
DIALOGADDFRAME(1000; 100; 10; 10; 10; 10; Frame! | Filled! | Gray!)
DIALOGADDHOTSPOT(1000; 101; 10; 10; 10; 10; Click!)
DIALOGADDTEXT(1000; 0; 28; 10; 125; 15; Left!; "Click to select/deselect check box")
DIALOGADDCHECKBOX(1000; 102; 60; 35; 50; 15; "Check Box"; var)
DIALOGDISPLAY(1000; 101; Msg)

WHILE(Loop = 1)
ENDWHILE
DIALOGDESTROY(1000)
QUIT

LABEL(Msg)
   IF(Msg[5] = WM_SYSCOMMAND)
      Loop = 0
   ENDIF
   IF(Msg[3] = 101)
      IF(vCheck = 1)
          REGIONSETCHECK("1000.102"; Check!)
          vCheck = 0
      ELSE
          REGIONSETCHECK("1000.102"; UnCheck!)
          vCheck = 1
      ENDIF
   ENDIF
   RETURN
```

For Example...

```
APPLICATION(WP; "WPOffice"; Default; "US")
// Demonstrate REGIONSETFOCUS command

Loop = 1
lParam = 7
WM_SYSCOMMAND = 274

DIALOGDEFINE(1000; 20; 30; 107; 110; NoFrame!; "RegionSetFocus")
DIALOGADDTEXT(1000; 0; 0; 15; 107; 15; Center!; "Select a button")
DIALOGADDLISTBOX(1000; 100; 10; 35; 85; 35; Sorted!; var)
FOREACH(vList; {"Radio Button 1"; "Radio Button 2"; "Radio Button 3"})
   DIALOGADDLISTITEM(1000; 100; vList)
ENDFOR
DIALOGHANDLE(hItem; 1000; 100)
DIALOGADDRADIOBUTTON(1000; 101; 10; 75; 15; 15; "1"; var)
DIALOGADDRADIOBUTTON(1000; 102; 40; 75; 15; 15; "2"; var)
DIALOGADDRADIOBUTTON(1000; 103; 70; 75; 15; 15; "3"; var)
DIALOGDISPLAY(1000; 100; Msg)

WHILE(Loop = 1)
ENDWHILE
DIALOGDESTROY(1000)
QUIT

LABEL(Msg)
   IF(Msg[5] = WM_SYSCOMMAND)
      Loop = 0
   ENDIF
   IF(LWORD(Msg[lParam]) = hItem)
      REGIONGETSELECTEDTEXT(vRB; "1000.100")
      SWITCH(vRB)
          CASEOF "Radio Button 1": vFocus = "101"
          CASEOF "Radio Button 2": vFocus = "102"
          CASEOF "Radio Button 3": vFocus = "103"
      ENDSWITCH
      REGIONSETFOCUS("1000." + vFocus)
   ENDIF
   RETURN

FUNCTION LWORD(vNum)
   vNum = vNum & 65535
   RETURN(vNum)
ENDFUNC
```

For Example...

```
APPLICATION(WP; "WPOffice"; Default; "US")
// Demonstrate REGIONASETWINDOWTEXT command

Loop = 1
vNmbr = 2
WM_SYSCOMMAND = 274

DIALOGDEFINE(1000; 50; 50; 150; 50; Percent! | NoFrame!; "RegionSetWindowText")
DIALOGADDFRAME(1000; 100; 10; 10; 10; 10; Frame! | Filled! | Gray!)
DIALOGADDHOTSPOT(1000; 101; 10; 10; 10; 10; Click!)
DIALOGADDTEXT(1000; 102; 28; 10; 125; 15; Left!; "Click to change text")
DIALOGDISPLAY(1000; 101; Msg)

WHILE(Loop = 1)
ENDWHILE
DIALOGDESTROY(1000)
QUIT

LABEL(Msg)
   IF(Msg[5] = WM_SYSCOMMAND)
      Loop = 0
   ENDIF
   IF(Msg[3] = 101)
      ChangeTitle
   ENDIF
   RETURN

LABEL(ChangeTitle)
   vChange = "Click " + vNmbr
   REGIONSETWINDOWTEXT("1000"; vChange)
   REGIONSETWINDOWTEXT("1000.102"; vChange)
   ASSIGN(vNmbr; vNmbr + 1)
   RETURN
```

For Example...

```
APPLICATION(WP; "WPOffice"; Default; "US")
// Demonstrate REGIONSHOWWINDOW command

Loop = 1
WM_SYSCOMMAND = 274
vDisable = 1

DIALOGDEFINE(1000; 50; 50; 175; 75; Percent! | NoFrame!; "RegionShowWindow")
DIALOGADDFRAME(1000; 100; 10; 10; 10; 10; Frame! | Filled! | Gray!)
DIALOGADDHOTSPOT(1000; 101; 10; 10; 10; 10; Click!)
DIALOGADDTEXT(1000; 0; 28; 10; 125; 15; Left!; "Click to hide/show the following text")
DIALOGADDTEXT(1000; 102; 0; 35; 175; 15; Center!; "Novell GroupWise 4.1")
DIALOGDISPLAY(1000; 101; Msg)

WHILE(Loop = 1)
ENDWHILE
DIALOGDESTROY(1000)
QUIT

LABEL(Msg)
   IF(Msg[5] = WM_SYSCOMMAND)
      Loop = 0
   ENDIF
   IF(Msg[3] = 101)
      IF(vDisable = 1)
          REGIONSHOWWINDOW("1000.102"; Hide!)
          vDisable = 0
      ELSE
          REGIONSHOWWINDOW("1000.102"; Show!)
          vDisable = 1
      ENDIF
   ENDIF
   RETURN
```

For Example...

```
APPLICATION(WP; "WPOffice"; Default; "US")
// Demonstrate COPYFILE, DOESFILEEXIST, GETCURRENTDIRECTORY, and
DOESDIRECTORYEXIST commands

VARERRCHK(Off!)
HRt = NTOC(0F90Ah)
CALL(GetFileNames)
CALL(CheckNames)
CALL(CopyFiles)
QUIT

LABEL(GetFileNames)
   GETSTRING(vSource; "Current directory: " + GETCURRENTDIRECTORY
      + HRt + HRt + "File to copy:"; "Copy File"; 100)
   GETSTRING(vDestination; "File to copy: "
      + vSource + HRt + HRt + "To:"; "Copy File"; 100)
   RETURN

LABEL(CheckNames)
   IF(NOT(DOESFILEEXIST(vSource) = True))
      ErrorMsg(vSource + " does not exist")
   ENDIF
   vLen = STRLEN(vDestination)
   vCnt = vLen
   WHILE(NOT(vResult = "\") OR (vStrLen = 0))
      vResult = SUBSTR(vDestination; vCnt; 1)
                vCnt = vCnt - 1
   ENDWHILE
   // parsed filename is not used
   vFilename = SUBSTR(vDestination; vCnt + 2; vLen - vCnt)
   vDirectoryName = SUBSTR(vDestination; 1; vCnt)
   IF(DOESDIRECTORYEXIST(vDirectoryName) = True)
      RETURN
   ELSE
      ErrorMsg(vDirectoryName + " does not exist")
   ENDIF

LABEL(CopyFiles)
   IF(COPYFILE(vSource; vDestination; NoPrompts!) = True)
                MESSAGEBOX(x; "COPYFILE"; "File copied successfully";
         IconInformation!)
   ELSE
      ErrorMsg("File not copied")
   ENDIF
   RETURN

PROCEDURE ErrorMsg(Msg)
   BEEP
```

```
    MESSAGEBOX(x; "Error"; Msg; IconExclamation!)
    QUIT
ENDPROC
```

For Example...

```
APPLICATION(WP; "WPOffice"; Default; "US")
// Demonstrate GETCURRENTDIRECTORY and SETCURRENTDIRECTORY command

ONCANCEL(QuitMacro)
HRt = NTOC(0F90Ah)

LABEL(SetCurrentDir)
   GETSTRING(vNewDir; "Current directory: " + GETCURRENTDIRECTORY
      + HRt + HRt + "New directory:"; "Directory Commands"; 100)
   IF(SETCURRENTDIRECTORY(vNewDir) = False)
      BEEP
      MESSAGEBOX(x; "Error"; "Invalid directory"; IconExclamation!)
      GO(SetCurrentDir)
   ELSE
      MESSAGEBOX(x; "Current Directory"; vNewDir; IconInformation!)
   ENDIF
QUIT

LABEL(QuitMacro)
   MESSAGEBOX(x; "End Macro"; "You chose Cancel or pressed Esc"; IconInformation!)
   QUIT
```

For Example...

```
APPLICATION(WP; "WPOffice"; Default; "US")
// Demonstrate CREATEDIRECTORY and DELETEDIRECTORY commands

ONCANCEL(QuitMacro)
HRt = NTOC(0F90Ah)

LABEL(CreateDir)
   GETSTRING(vCreateDir; "Current directory: " + GETCURRENTDIRECTORY
      + HRt + HRt + "New directory:"; "Create New Directory"; 100)
   IF(CREATEDIRECTORY(vCreateDir) = False)
      BEEP
      MESSAGEBOX(x; "Error"; "Directory not created"; IconExclamation!)
      GO(CreateDir)
   ELSE
      MESSAGEBOX(vYesNo; "New Directory"; vCreateDir + HRt + Hrt + "Delete new directory
      now?"; IconQuestion! | YesNo!)
      IF(vYesNo = 6)
                           MESSAGEBOX(x; "Delete Directory";
          "Directory deleted: " + DELETEDIRECTORY(vCreateDir))
      ENDIF
   ENDIF
QUIT

LABEL(QuitMacro)
   MESSAGEBOX(x; "End Macro"; "You chose Cancel or pressed Esc";
IconInformation!)
   QUIT
```

For Example...

```
APPLICATION(WP; "WPOffice"; Default; "US")
// Demonstrate RENAMEDIRECTORY command

ONCANCEL(QuitMacro)
HRt = NTOC(0F90Ah)

LABEL(RenameDir)
   GETSTRING(vRenameDir; "Current directory: " +
      GETCURRENTDIRECTORY + HRt + HRt + "Rename Directory:";
      "Rename Current Directory"; 100)
   IF(RENAMEDIRECTORY("c:\test\test"; vRenameDir; Prompts!) = False)
      BEEP
      MESSAGEBOX(x; "Error"; "Directory not renamed"; IconExclamation!)
      GO(RenameDir)
   ELSE
      MESSAGEBOX(x; "New Directory Name"; vRenameDir; IconInformation!)
   ENDIF

LABEL(QuitMacro)
   MESSAGEBOX(x; "End Macro"; "You chose Cancel or pressed Esc"; IconInformation!)
   QUIT
```

F For Example...
F
F
F

```
APPLICATION(WP; "WPOffice"; Default; "US")
// Demonstrate GETFILEATTRIBUTS command

ONCANCEL(QuitMacro)
HRt = NTOC(0F90Ah)

GETSTRING(vFilename; "File:"; "Get File Attributes"; 100)
IF(NOT(vFilename = ""))
   MESSAGEBOX(x; ""; GETFILEATTRIBUTES(vFilename))
ENDIF
QUIT

LABEL(QuitMacro)
   MESSAGEBOX(x; "End Macro"; "You chose Cancel or pressed Esc"; IconInformation!)
   QUIT
```

For Example...

```
APPLICATION(WP; "WPOffice"; Default; "US")
// Demonstrate SETFILEATTRIBUTES command

ONCANCEL(QuitMacro)
HRt = NTOC(0F90Ah)

GETSTRING(vFilename; "Set read-only attribute." + HRt + Hrt + "File:";
"SETFILEATTRIBUTES"; 100)
IF(NOT(vFilename = ""))
   MESSAGEBOX(x; "SETFILEATTRIBUTES"; "Read-only status: " +
   SETFILEATTRIBUTES(vFilename; ReadOnly!; Prompts!))
ENDIF
QUIT

LABEL(QuitMacro)
   MESSAGEBOX(x; "End Macro"; "You chose Cancel or pressed Esc"; IconInformation!)
   QUIT
```

For Example...

```
APPLICATION(WP; "WPOffice"; Default; "US")
// Demonstrate GETFILEDATEANDTIME command

ONCANCEL(QuitMacro)
HRt = NTOC(0F90Ah)

GETSTRING(vFilename; "Date and time." + HRt + Hrt + "File:"; "GETFILEDATEANDTIME";
100)
IF(NOT(vFilename = ""))
   MESSAGEBOX(x; "Number of seconds since Jan 1, 1980";
   GETFILEDATEANDTIME(vFilename))
ENDIF
QUIT

LABEL(QuitMacro)
   MESSAGEBOX(x; "End Macro"; "You chose Cancel or pressed Esc"; IconInformation!)
   QUIT
```

For Example...

```
APPLICATION(WP; "WPOffice"; Default; "US")
// Demonstrate SETFILEDATEANDTIME command

ONCANCEL(QuitMacro)
HRt = NTOC(0F90Ah)

GETSTRING(vFilename; "Date and time." + HRt + Hrt + "File:"; "GETFILEDATEANDTIME";
100)
IF(NOT(vFilename = ""))
   MESSAGEBOX(x; "New File Date and Time Stamp"; "Result: " +
   SETFILEDATEANDTIME(vFilename; 485780000) + " (Jun 30, 1994 / 7:02a)")
ENDIF
QUIT

LABEL(QuitMacro)
   MESSAGEBOX(x; "End Macro"; "You chose Cancel or pressed Esc"; IconInformation!)
   QUIT
```

For Example...

```
APPLICATION(WP; "WPOffice"; Default; "US")
// Demonstrate DELETEFILE command

ONCANCEL(QuitMacro)

GETSTRING(vFilename; "File:"; "DELETEFILE"; 100)
IF(NOT(vFilename = ""))
   IF(NOT(DELETEFILE(vFilename; Prompts!) = True))
      Msg = vFilename + " not deleted"
   ELSE
      Msg = vFilename + " successfully deleted"
   ENDIF
   MESSAGEBOX(x; "DELETEFILE"; Msg; IconInformation!)
ENDIF
QUIT

LABEL(QuitMacro)
   MESSAGEBOX(x; "End Macro"; "You chose Cancel or pressed Esc"; IconInformation!)
   QUIT
```

For Example...

```
APPLICATION(WP; "WPOffice"; Default; "US")
// Demonstrate RENAMEFILE command

ONCANCEL(QuitMacro)
HRt = NTOC(0F90Ah)

LABEL(begin)
   GETSTRING(vOldName; "Old file name:"; "RENAMEFILE"; 100)
   IF(vOldName = "")
      GO(begin)
   ENDIF
   GETSTRING(vNewName; "Old file: " + vOldName + HRt + HRt + "New file name:";
   "RENAMEFILE"; 100)
   IF(NOT(vNewName = ""))
      IF(NOT(RENAMEFILE(vOldName; vNewName; Prompts!) = True))
          Msg = vOldName + " not renamed"
      ELSE
          Msg = "New file name: " + vNewName
      ENDIF
      MESSAGEBOX(x; "RENAMEFILE"; Msg; IconInformation!)
   ENDIF
QUIT

LABEL(QuitMacro)
   MESSAGEBOX(x; "End Macro"; "You chose Cancel or pressed Esc"; IconInformation!)
   QUIT
```

For Example...

```
APPLICATION(WP; "WPOffice"; Default; "US")
// Demonstrate OPENFILE, CLOSEFILE, FILEWRITE, FILEREAD, FILEPOSITION, FILEISEOF, and
FILESIZE commands

CALL(OpenInfo)
CALL(WriteToFile)
CALL(ReadFromFile)
CALL(QuitMacro)

LABEL(OpenInfo)
   HRt = NTOC(0F90Ah)
   ASSIGN(vPath; EnvPrefMacroPath)
   hFile = OPENFILE(vPath + "FILE_IO.DOC"; ReadWrite!;
      DenyNone!; AnsiText!)
   IF(hFile < 0)
      ErrorMsg("opening"; hFile)
   ELSE
      vIsEOF = "Postion marker (EOF): " + FILEISEOF(hFile)
      vSize = "File size: " + FILESIZE(vPath + "FILE_IO.DOC")
      MESSAGEBOX(X;"File Information"; vIsEOF + HRt + HRt + vSize)
   ENDIF
   RETURN

LABEL(WriteToFile)
   FORNEXT(z; 1; 10; 1)
      IF(FILEWRITE(hFile; "Novell GroupWise 4.1 is a great product! ";
          NoNewLine!) < 0)
          ErrorMsg("writing to"; hFile)
      ENDIF
   ENDFOR
   RETURN

LABEL(ReadFromFile)
   FILEPOSITION(hFile; 0; FromBeginning!)
   IF(FILEREAD(hFile; vText) < 0)
      ErrorMsg("reading from"; hFile)
   ELSE
      MESSAGEBOX(x; "File I/O"; vText)
   ENDIF
   RETURN

PROCEDURE ErrorMsg(Msg; hFile)
   BEEP
   MESSAGEBOX(x; "Error"; "Error " + Msg + " file"; IconExclamation!)
ENDPROC

LABEL(QuitMacro)
   IF(NOT(CLOSEFILE(hFile)) = True)
```

```
   BEEP
   MESSAGEBOX(x; "Error"; "Error closing file"; IconExclamation!)
ELSE
   MESSAGEBOX(x; "File I/O"; "File successfully closed")
ENDIF
QUIT
```

For Example...

```
APPLICATION(WP; "WPOffice"; Default; "US")
// Demonstrate FILETRUNCATE command

CALL(OpenWriteToReadFromFile)
CALL(QuitMacro)

LABEL(OpenWriteToReadFromFile)
   HRt = NTOC(0F90Ah)
   ASSIGN(vPath; EnvPrefMacroPath)
   hFile = OPENFILE(vPath + "FILE_IO.DOC"; ReadWrite!;
      DenyNone!; AnsiText!)
   IF(hFile < 0)
      MESSAGEBOX(x; "Error"; "Error opening file"; IconExclamation!)
      QUIT
   ENDIF
   Test(FILEWRITE(hFile;
      "Novell GroupWise 4.1 is a wonderful product!";
      NoNewLine!); "FILEWRITE")
   Test(FILEPOSITION(hFile; 0; FromBeginning!); "FILEPOSITION")
   Test(FILEREAD(hFile; vText); "FILEREAD")
   Test(FILEPOSITION(hFile; 20; FromBeginning!);
      "FILEPOSITION")
   Test(FILETRUNCATE(hFile); "FILETRUNCATE")
   Test(FILEPOSITION(hFile; 0; FromBeginning!);
      "FILEPOSITION")
   Test(FILEREAD(hFile; vTruncate); "FILEREAD")
   MESSAGEBOX(x; "File I/O - Truncate File"; vText+HRt+HRt+vTruncate)
   RETURN

PROCEDURE Test(vError; vStr)
   // vError = value returned from FILE command
   IF(vError < 0)
      BEEP
      MESSAGEBOX(x; "Error"; vStr; IconExclamation!)
      GO(QuitMacro)
   ENDIF
ENDPROC

LABEL(QuitMacro)
   IF(NOT(CLOSEFILE(hFile)) = True)
      BEEP
      MESSAGEBOX(x; "Error"; "Error closing file"; IconExclamation!)
   ELSE
      MESSAGEBOX(x; "File I/O"; "File successfully closed")
   ENDIF
   QUIT
```

For Example...

```
APPLICATION(WP; "WPOffice"; Default; "US")
// Demonstrate FILEERROR and FINDFILE commands

vPath = EnvPrefMacroPath
vFilename = FILEFIND(vPath + "*.wcm"; Normal!; 1)
WHILE(vFilename != "")
   CALL DisplayName(vFilename)
   vFilename = FILEFIND("";;1)
ENDWHILE

PROCEDURE DisplayName(vFilename)
   MESSAGEBOX(x; "FILEFIND"; vFilename; IconInformation!)
ENDPROC

MESSAGEBOX(x; "Message"; "Error code = " + FILEERROR)
```

For Example...

```
// There are two macros in this example
// The INCLUDE macro is compiled when you play the main macro
// Demonstrate INCLUDE command

// INCLUDE macro: INC.WCM
// Copy INC.WCM to directory specified in PARENT.WCM
APPLICATION (A1; "WPOffice"; Default; "US")
   Type("2")
   PROCEDURE TestProc(x)
      Type(x)
   ENDPROC

//Main macro: PARENT.WCM
APPLICATION (A1; "WPOffice"; Default; "US")
   Type("1")
   INCLUDE("C:\OFFICE40\MACROS\INC.WCM")
   Type("3")
   CALL TestProc(4)
   QUIT
```

For Example...

```
APPLICATION (A1; "WPOffice"; Default; "US")

// Place DLLCALL PROTOTYPE statements at start of macro

// Example 1
// Parameter names (buffer and size) are not required
DLLCALL PROTOTYPE GetWindowsDirectory("kernel"; ; word;
      {address(ansistring(buffer)); loword(size)})

// Example 2
DLLCALL PROTOTYPE GetSysDir("kernel"; "GetSystemDirectory";
      void; {address(ansistring(buffer)); loword(size)})

// Example 3
DLLCALL PROTOTYPE GetSystemDirectory("kernel"; word;
      {address(ansistring(buffer)); loword(size)})

// Load DLL library
DLLLOAD(h; "kernel")

// Call DLL function with DLLCALL
DLLCALL(h; "GetWindowsDirectory"; s; word;
   {address(ansistring(a)); loword(250)})
MESSAGEBOX(x; "DLLCALL"; " Directory: " + a + " -- Size: " + s)

// Call DLL function with DLLCALL PROTOTYPE
// Value returning function
a := ""
s := 0
s := GetWindowsDirectory(size:250; buffer:a)
MESSAGEBOX(x; "DLLCALL PROTOTYPE (function)";
   "Directory: " + a + " -- Size: " + s)

// DLL routine defined as a procedure
// DLL routine cannot be called as function
a := ""
GetWindowsDirectory(a; GetSystemDirectory(buffer:a; 250)+10)
MESSAGEBOX(x; "DLLCALL PROTOTYPE (procedure)"; a)

a := ""
GetSysDir(a; 250)
MESSAGEBOX(x; "DLLCALL PROTOTYPE (procedure)"; a)

// Free DLL library
DLLFree(h)

// The following statements produce errors
//s := GetWindowsDirectory(size:250; buffer:a; size:10; size:1024)
```

```
//s := GetWindowsDirectory(size:250; buffer:a; b; c);
//s := GetWindowsDirectory(size:250; size:20)
//s := GetWindowsDirectory(a; b; c)
```




For Example...

```
APPLICATION(A1; "WordPerfect"; Default; "US")
// Demonstrate DIALOGSHOW and DIALOGDISMISS commands

HRt = NTOC(0F90Ah)
WM_COMMAND = 273
WM_SYSCOMMAND = 274
vPopup = "Popup item 1"
vList = "List item 1"

DIALOGSHOW("Dialog1"; "WordPerfect"; Msg)

x := 0
While(x = 0)
Endwhile
QUIT

LABEL(MSG)
        IF((Msg[5] = WM_SYSCOMMAND) OR (Msg[3] = "CancelBttn"))
                BEEP
                x = 1
                RETURN
        ENDIF
        IF(Msg[5] = WM_COMMAND)
                DIALOGDISMISS("Dialog1"; "OKBttn")
                MsgBox
                DIALOGSHOW("Dialog1"; "WordPerfect"; Msg)
        ENDIF
        RETURN

LABEL(MsgBox)
        MESSAGEBOX(z; "Selections"; vList + HRt + HRt + vPopup + HRt)
        RETURN
```

For Example...

```
APPLICATION(A1; "WPOffice"; Default; "US")
// Demonstrate REGIONISVISIBLE, DIALOGSHOW, and DIALOGDISMISS commands

DialogDefine("Dialog"; 50; 50; 200; 200; Percent!; "REGIONISVISIBLE")
DialogAddPushButton("Dialog"; 1; 85; 10; 25; 13;
      NonDefaultBttn!; "OK")
DialogAddPushButton("Dialog"; 2; 85; 30; 25; 13;
      DefaultBttn!; "Cancel")
DialogShow("Dialog"; "WordPerfect"; Msg)

x = 0
WHILE(x != 1)
         REGIONISVISIBLE (vResult; "Dialog.1")
         MESSAGEBOX(z; "OK button"; "Visible: " + CheckResult(vResult))
         REGIONSHOWWINDOW ("Dialog.1"; Hide!)
         REGIONISVISIBLE (vResult; "Dialog.1")
         MESSAGEBOX(z; "OK button"; "Visible: " + CheckResult(vResult))
         x = 1
ENDWHILE
DialogDismiss("Dialog"; "CancelBttn")
QUIT

LABEL(Msg)
RETURN

FUNCTION CheckResult(vResult)
         IF(vResult = 0)
                  vResult = "No"
         ELSE
                  vResult = "Yes"
         ENDIF
         RETURN(vResult)
ENDFUNC
```