

No additional information is available for this predefined constant.

_MAX_LENGTH

_MAX_LENGTH is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)

_MAX_STRING

_MAX_STRING is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)

AFTER

AFTER is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

 [Click here for a list of functions that use this constant.](#)

{button ,PI("", `comment`)} [Feedback](#)

ALLCONTENTS

ALLCONTENTS is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

 [Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)

ALLCONTROLS

ALLCONTROLS is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

 [Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)

APPEND

APPEND is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

 [Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)

ASKDESTPATH

ASKDESTPATH is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

 [Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)

ASKOPTIONS

ASKOPTIONS is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

 [Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)

ASKPATH

ASKPATH is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

 [Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)

ASKTEXT

ASKTEXT is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

 [Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)

BACK

BACK is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

 [Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)

BACKBUTTON

BACKBUTTON is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

 [Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)

BACKGROUND

BACKGROUND is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.

 [Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



BACKGROUNDCAPTION

BACKGROUNDCAPTION is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



BADPATH

BADPATH is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



BADTAGFILE

BADTAGFILE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



BASEMEMORY

BASEMEMORY is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



BEFORE

BEFORE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



BILLBOARD

BILLBOARD is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



BINARY

BINARY is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



BITMAP256COLORS

BITMAP256COLORS is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



BITMAPFADE

BITMAPFADE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



BITMAPICON

BITMAPICON is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



BK_BLUE

BK_BLUE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



BK_GREEN

BK_GREEN is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment`)} [Feedback](#)



BK_MAGENTA

BK_MAGENTA is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



BK_MAGENTA1

BK_MAGENTA1 is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment`)} [Feedback](#)



BK_ORANGE

BK_ORANGE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



BK_PINK

BK_PINK is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



BK_RED

BK_RED is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



BK_SMOOTH

BK_SMOOTH is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



BK_SOLIDBLACK

BK_SOLIDBLACK is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



BK_SOLIDBLUE

BK_SOLIDBLUE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



BK_SOLIDGREEN

BK_SOLIDGREEN is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



BK_SOLIDMAGENTA

BK_SOLIDMAGENTA is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



BK_SOLIDORANGE

BK_SOLIDORANGE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



BK_SOLIDPINK

BK_SOLIDPINK is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



BK_SOLIDRED

BK_SOLIDRED is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



BK_SOLIDWHITE

BK_SOLIDWHITE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



BK_SOLIDYELLOW

BK_SOLIDYELLOW is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



BK_YELLOW

BK_YELLOW is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



BLACK

BLACK is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



BLUE

BLUE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



BOOTUPDRIVE

BOOTUPDRIVE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment`)} [Feedback](#)



BUTTON_CHECKED

BUTTON_CHECKED is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



BUTTON_ENTER

BUTTON_ENTER is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



BUTTON_UNCHECKED

BUTTON_UNCHECKED is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



BUTTON_UNKNOWN

BUTTON_UNKNOWN is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



CANCEL

CANCEL is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



CANCELBUTTON

CANCELBUTTON is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



CC_ERR_FILEFORMATERROR

CC_ERR_FILEFORMATERROR is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



CC_ERR_FILEREADERROR

CC_ERR_FILEREADERROR is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



CC_ERR_NOCOMPONENTLIST

CC_ERR_NOCOMPONENTLIST is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



CC_ERR_OUTOFMEMORY

CC_ERR_OUTOFMEMORY is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



CDROM

CDROM is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



CDROM_DRIVE

CDROM_DRIVE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



CENTERED

CENTERED is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



CHANGEDIR

CHANGEDIR is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



CHECKBOX

CHECKBOX is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



CHECKBOX95

CHECKBOX95 is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



CHECKLINE

CHECKLINE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment`)} [Feedback](#)



CHECKMARK

CHECKMARK is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment`)} [Feedback](#)



CMD_CLOSE

CMD_CLOSE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



CMD_MAXIMIZE

CMD_MAXIMIZE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



CMD_MINIMIZE

CMD_MINIMIZE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



CMD_PUSHDOWN

CMD_PUSHDOWN is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



CMD_RESTORE

CMD_RESTORE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



COLORMODE256

COLORMODE256 is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



COLORS

COLORS is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



COMBOBOX_ENTER

COMBOBOX_ENTER is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



COMBOBOX_SELECT

COMBOBOX_SELECT is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment`)} [Feedback](#)



COMMAND

COMMAND is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



COMMANDEX

COMMANDEX is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



COMMON

COMMON is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



COMP_DONE

COMP_DONE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



COMP_ERR_CREATEDIR

COMP_ERR_CREATEDIR is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



COMP_ERR_DESTCONFLICT

COMP_ERR_DESTCONFLICT is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



COMP_ERR_FILENOTINLIB

COMP_ERR_FILENOTINLIB is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



COMP_ERR_FILESIZE

COMP_ERR_FILESIZE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



COMP_ERR_FILETOOLARGE

COMP_ERR_FILETOOLARGE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



COMP_ERR_HEADER

COMP_ERR_HEADER is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



COMP_ERR_INCOMPATIBLE

COMP_ERR_INCOMPATIBLE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



COMP_ERR_INTPUTNOTCOMPRESSED

COMP_ERR_INTPUTNOTCOMPRESSED is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



COMP_ERR_INVALIDLIST

COMP_ERR_INVALIDLIST is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



COMP_ERR_LAUNCHSERVER

COMP_ERR_LAUNCHSERVER is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment`)} [Feedback](#)



COMP_ERR_MEMORY

COMP_ERR_MEMORY is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



COMP_ERR_NODISKSPACE

COMP_ERR_NODISKSPACE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



COMP_ERR_OPENINPUT

COMP_ERR_OPENINPUT is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



COMP_ERR_OPENOUTPUT

COMP_ERR_OPENOUTPUT is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



COMP_ERR_OPTIONS

COMP_ERR_OPTIONS is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



COMP_ERR_OUTPUTNOTCOMPRESSED

COMP_ERR_OUTPUTNOTCOMPRESSED is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



COMP_ERR_SPLIT

COMP_ERR_SPLIT is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



COMP_ERR_TARGET

COMP_ERR_TARGET is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



COMP_ERR_TARGETREADONLY

COMP_ERR_TARGETREADONLY is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



COMP_ERR_WRITE

COMP_ERR_WRITE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



COMP_INFO_ATTRIBUTE

COMP_INFO_ATTRIBUTE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



COMP_INFO_COMPsize

COMP_INFO_COMPsize is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



COMP_INFO_DATE

COMP_INFO_DATE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment`)} [Feedback](#)



COMP_INFO_INVALIDATEPASSWORD

COMP_INFO_INVALIDATEPASSWORD is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



COMP_INFO_ORIGSIZE

COMP_INFO_ORIGSIZE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



COMP_INFO_SETPASSWORD

COMP_INFO_SETPASSWORD is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment`)} [Feedback](#)



COMP_INFO_TIME

COMP_INFO_TIME is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



COMP_INFO_VERSIONLS

COMP_INFO_VERSIONLS is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



COMP_INFO_VERSIONMS

COMP_INFO_VERSIONMS is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



COMP_NORMAL

COMP_NORMAL is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



COMP_UPDATE_DATE

COMP_UPDATE_DATE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



COMP_UPDATE_DATE_NEWER

COMP_UPDATE_DATE_NEWER is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



COMP_UPDATE_SAME

COMP_UPDATE_SAME is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



COMP_UPDATE_VERSION

COMP_UPDATE_VERSION is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



COMPACT

COMPACT is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



COMPARE_DATE

COMPARE_DATE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



COMPARE_SIZE

COMPARE_SIZE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment`)} [Feedback](#)



COMPARE_VERSION

COMPARE_VERSION is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



COMPONENT_FIELD_CDROM_FOLDER

COMPONENT_FIELD_CDROM_FOLDER is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



COMPONENT_FIELD_DESCRIPTION

COMPONENT_FIELD_DESCRIPTION is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



COMPONENT_FIELD_DESTINATION

COMPONENT_FIELD_DESTINATION is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



COMPONENT_FIELD_DISPLAYNAME

COMPONENT_FIELD_DISPLAYNAME is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



COMPONENT_FIELD_FILENEED

COMPONENT_FIELD_FILENEED is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



COMPONENT_FIELD_FTPLOCATION

COMPONENT_FIELD_FTPLOCATION is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



COMPONENT_FIELD_HTTPLOCATION

COMPONENT_FIELD_HTTPLOCATION is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



COMPONENT_FIELD_OVERWRITE

COMPONENT_FIELD_OVERWRITE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



COMPONENT_FIELD_MISC

COMPONENT_FIELD_MISC is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



COMPONENT_FIELD_PASSWORD

COMPONENT_FIELD_PASSWORD is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



COMPONENT_FIELD_SELECTED

COMPONENT_FIELD_SELECTED is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment`)} [Feedback](#)



COMPONENT_FIELD_SIZE

COMPONENT_FIELD_SIZE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



COMPONENT_FIELD_STATUS

COMPONENT_FIELD_STATUS is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



COMPONENT_FIELD_VISIBLE

COMPONENT_FIELD_VISIBLE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



COMPONENT_FILEINFO_COMPRESSED

COMPONENT_FILEINFO_COMPRESSED is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("", `comment`)} [Feedback](#)



COMPONENT_FILEINFO_COMPRESSENGINE

COMPONENT_FILEINFO_COMPRESSENGINE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



COMPONENT_FILEINFO_LANGUAGE

COMPONENT_FILEINFO_LANGUAGE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



COMPONENT_FILEINFO_OS

COMPONENT_FILEINFO_OS is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



COMPONENT_FILEINFO_POTENTIALLYLOCKED

COMPONENT_FILEINFO_POTENTIALLYLOCKED is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



COMPONENT_FILEINFO_SELFREGISTERING

COMPONENT_FILEINFO_SELFREGISTERING is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



COMPONENT_FILEINFO_SHARED

COMPONENT_FILEINFO_SHARED is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



COMPONENT_INFO_ATTRIBUTE

COMPONENT_INFO_ATTRIBUTE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



COMPONENT_INFO_COMPSIZE

COMPONENT_INFO_COMPSIZE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



COMPONENT_INFO_DATE

COMPONENT_INFO_DATE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)

COMPONENT_INFO_DATE_EX

COMPONENT_INFO_DATE_EX is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("", `comment`)} [Feedback](#)



COMPONENT_INFO_LANGUAGE

COMPONENT_INFO_LANGUAGE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



COMPONENT_INFO_ORIGSIZE

COMPONENT_INFO_ORIGSIZE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



COMPONENT_INFO_OS

COMPONENT_INFO_OS is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



COMPONENT_INFO_TIME

COMPONENT_INFO_TIME is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



COMPONENT_INFO_VERSIONLS

COMPONENT_INFO_VERSIONLS is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment`)} [Feedback](#)



COMPONENT_INFO_VERSIONMS

COMPONENT_INFO_VERSIONMS is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment`)} [Feedback](#)



COMPONENT_INFO_VERSIONSTR

COMPONENT_INFO_VERSIONSTR is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



COMPONENT_VALUE_ALWAYSoverwrite

COMPONENT_VALUE_ALWAYSoverwrite is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



COMPONENT_VALUE_CRITICAL

COMPONENT_VALUE_CRITICAL is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment`)} [Feedback](#)



COMPONENT_VALUE_HIGHLYRECOMMENDED

COMPONENT_VALUE_HIGHLYRECOMMENDED is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



COMPONENT_VALUE_NEVEROVERWRITE

COMPONENT_VALUE_NEVEROVERWRITE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



COMPONENT_VALUE_NEWERDATE

COMPONENT_VALUE_NEWERDATE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



COMPONENT_VALUE_NEWERVERSION

COMPONENT_VALUE_NEWERVERSION is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



COMPONENT_VALUE_OLDERDATE

COMPONENT_VALUE_OLDERDATE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



COMPONENT_VALUE_OLDERVERSION

COMPONENT_VALUE_OLDERVERSION is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



COMPONENT_VALUE_SAMEORNEWDATE

COMPONENT_VALUE_SAMEORNEWDATE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



COMPONENT_VALUE_SAMEORNEWERVERSION

COMPONENT_VALUE_SAMEORNEWERVERSION is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



COMPONENT_VALUE_STANDARD

COMPONENT_VALUE_STANDARD is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



COMPONENT_VIEW_CHANGE

COMPONENT_VIEW_CHANGE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment`)} [Feedback](#)



COMPONENT_VIEW_CHILDVIEW

COMPONENT_VIEW_CHILDVIEW is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



COMPONENT_VIEW_COMPONENT

COMPONENT_VIEW_COMPONENT is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



COMPONENT_VIEW_DESCRIPTION

COMPONENT_VIEW_DESCRIPTION is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



COMPONENT_VIEW_MEDIA

COMPONENT_VIEW_MEDIA is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment`)} [Feedback](#)



COMPONENT_VIEW_PARENTVIEW

COMPONENT_VIEW_PARENTVIEW is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



COMPONENT_VIEW_SIZEAVAIL

COMPONENT_VIEW_SIZEAVAIL is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



COMPONENT_VIEW_SIZETOTAL

COMPONENT_VIEW_SIZETOTAL is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment`)} [Feedback](#)



COMPONENT_VIEW_TARGETLOCATION

COMPONENT_VIEW_TARGETLOCATION is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



COMPRESSHIGH

COMPRESSHIGH is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



COMPRESSLOW

COMPRESSLOW is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



COMPRESSMED

COMPRESSMED is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment`)} [Feedback](#)



COMPRESSNONE

COMPRESSNONE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



CONTIGUOUS

CONTIGUOUS is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment`)} [Feedback](#)



CONTINUE

CONTINUE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



COPY_ERR_CREATEDIR

COPY_ERR_CREATEDIR is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



COPY_ERR_NODISKSPACE

COPY_ERR_NODISKSPACE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



COPY_ERR_OPENINPUT

COPY_ERR_OPENINPUT is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



COPY_ERR_OPENOUTPUT

COPY_ERR_OPENOUTPUT is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



COPY_ERR_TARGETREADONLY

COPY_ERR_TARGETREADONLY is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



COPY_ERR_MEMORY

COPY_ERR_MEMORY is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



CORECOMPONENTHANDLING

CORECOMPONENTHANDLING is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



CPU

CPU is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment`)} [Feedback](#)



CUSTOM

CUSTOM is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



DATA_COMPONENT

DATA_COMPONENT is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



DATA_LIST

DATA_LIST is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



DATA_NUMBER

DATA_NUMBER is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



DATA_STRING

DATA_STRING is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment`)} [Feedback](#)



DATE

DATE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



DEFAULT

DEFAULT is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



DEFWINDOWMODE

DEFWINDOWMODE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



DELETE_EOF

DELETE_EOF is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



DIALOG

DIALOG is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



DIALOGCACHE

DIALOGCACHE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



DIALOGTHINFONT

DIALOGTHINFONT is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



DIR_WRITEABLE

DIR_WRITEABLE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



DIRECTORY

DIRECTORY is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



DISABLE

DISABLE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



DISK

DISK is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment`)} [Feedback](#)



DISK_FREESPACE

DISK_FREESPACE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



DISK_TOTALSPACE

DISK_TOTALSPACE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



DISKID

DISKID is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment`)} [Feedback](#)



DLG_ASK_OPTIONS

DLG_ASK_OPTIONS is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



DLG_ASK_PATH

DLG_ASK_PATH is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



DLG_ASK_TEXT

DLG_ASK_TEXT is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



DLG_ASK_YESNO

DLG_ASK_YESNO is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



DLG_CANCEL

DLG_CANCEL is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



DLG_CDIRE

DLG_CDIRE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



DLG_CDIR_MSG

DLG_CDIR_MSG is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



DLG_CENTERED

DLG_CENTERED is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



DLG_CLOSE

DLG_CLOSE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



DLG_DIR_DIRECTORY

DLG_DIR_DIRECTORY is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



DLG_DIR_DRIVE

DLG_DIR_DRIVE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



DLG_DIR_FILE

DLG_DIR_FILE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



DLG_ENTER_DISK

DLG_ENTER_DISK is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



DLG_ERR

DLG_ERR is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



DLG_ERR_ALREADY_EXISTS

DLG_ERR_ALREADY_EXISTS is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



DLG_ERR_ENDDLG

DLG_ERR_ENDDLG is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



DLG_INFO_ALTIMAGE

DLG_INFO_ALTIMAGE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



DLG_INFO_CHECKMETHOD

DLG_INFO_CHECKMETHOD is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment`)} [Feedback](#)



DLG_INFO_CHECKSELECTION

DLG_INFO_CHECKSELECTION is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



DLG_INFO_ENABLEIMAGE

DLG_INFO_ENABLEIMAGE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



DLG_INFO_KUNITS

DLG_INFO_KUNITS is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



DLG_INFO_USEDECIMAL

DLG_INFO_USEDECIMAL is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



DLG_INIT

DLG_INIT is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



DLG_MSG_ALL

DLG_MSG_ALL is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



DLG_MSG_INFORMATION

DLG_MSG_INFORMATION is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



DLG_MSG_NOT_HAND

DLG_MSG_NOT_HAND is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



DLG_MSG_SEVERE

DLG_MSG_SEVERE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



DLG_MSG_STANDARD

DLG_MSG_STANDARD is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



DLG_MSG_WARNING

DLG_MSG_WARNING is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



DLG_OK

DLG_OK is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



DLG_STATUS

DLG_STATUS is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



DLG_USER_CAPTION

DLG_USER_CAPTION is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



DRIVE

DRIVE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



DRIVEOPEN

DRIVEOPEN is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



EDITBOX_CHANGE

EDITBOX_CHANGE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



EFF_BOXSTRIPE

EFF_BOXSTRIPE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment`)} [Feedback](#)



EFF_FADE

EFF_FADE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



EFF_HORIZREVEAL

EFF_HORIZREVEAL is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



EFF_HORIZSTRIPE

EFF_HORIZSTRIPE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



EFF_NONE

EFF_NONE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



EFF_REVEAL

EFF_REVEAL is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



EFF_VERTSTRIPE

EFF_VERTSTRIPE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ENABLE

ENABLE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



END_OF_FILE

END_OF_FILE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



END_OF_LIST

END_OF_LIST is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ENHANCED

ENHANCED is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ENTERDISK

ENTERDISK is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ENTERDISK_ERRMSG

ENTERDISK_ERRMSG is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ENTERDISKBEEP

ENTERDISKBEEP is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ENVSPACE

ENVSPACE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



EQUALS

EQUALS is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment`)} [Feedback](#)



ERR_BADPATH

ERR_BADPATH is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ERR_BADTAGFILE

ERR_BADTAGFILE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ERR_BOX_BADPATH

ERR_BOX_BADPATH is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ERR_BOX_BADTAGFILE

ERR_BOX_BADTAGFILE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ERR_BOX_DISKID

ERR_BOX_DISKID is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ERR_BOX_DRIVEOPEN

ERR_BOX_DRIVEOPEN is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ERR_BOX_EXIT

ERR_BOX_EXIT is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ERR_BOX_HELP

ERR_BOX_HELP is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ERR_BOX_NOSPACE

ERR_BOX_NOSPACE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ERR_BOX_PAUSE

ERR_BOX_PAUSE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment`)} [Feedback](#)



ERR_BOX_READONLY

ERR_BOX_READONLY is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ERR_DISKID

ERR_DISKID is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ERR_DRIVEOPEN

ERR_DRIVEOPEN is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



EXCLUDE_SUBDIR

EXCLUDE_SUBDIR is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



EXCLUSIVE

EXCLUSIVE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



EXISTS

EXISTS is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment`)} [Feedback](#)



EXIT

EXIT is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



EXTENDEDMEMORY

EXTENDEDMEMORY is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



EXTENSION_ONLY

EXTENSION_ONLY is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



FADE_IN

FADE_IN is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



FADE_OUT

FADE_OUT is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



FAILIFEXISTS

FAILIFEXISTS is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



FALSE

FALSE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



FDRIVE_NUM

FDRIVE_NUM is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



FEEDBACK

FEEDBACK is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



FEEDBACK_FULL

FEEDBACK_FULL is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



FEEDBACK_OPERATION

FEEDBACK_OPERATION is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



FEEDBACK_SPACE

FEEDBACK_SPACE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



FILE_ATTR_ARCHIVED

FILE_ATTR_ARCHIVED is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



FILE_ATTR_DIRECTORY

FILE_ATTR_DIRECTORY is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



FILE_ATTR_HIDDEN

FILE_ATTR_HIDDEN is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



FILE_ATTR_NORMAL

FILE_ATTR_NORMAL is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



FILE_ATTR_READONLY

FILE_ATTR_READONLY is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



FILE_ATTR_SYSTEM

FILE_ATTR_SYSTEM is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



FILE_ATTRIBUTE

FILE_ATTRIBUTE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



FILE_BIN_CUR

FILE_BIN_CUR is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment`)} [Feedback](#)



FILE_BIN_END

FILE_BIN_END is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



FILE_BIN_START

FILE_BIN_START is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment`)} [Feedback](#)



FILE_DATE

FILE_DATE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



FILE_EXISTS

FILE_EXISTS is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



FILE_INSTALLED

FILE_INSTALLED is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



FILE_INVALID

FILE_INVALID is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment`)} [Feedback](#)



FILE_IS_LOCKED

FILE_IS_LOCKED is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



FILE_LINE_LENGTH

FILE_LINE_LENGTH is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



FILE_LOCKED

FILE_LOCKED is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



FILE_MODE_APPEND

FILE_MODE_APPEND is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



FILE_MODE_BINARY

FILE_MODE_BINARY is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



FILE_MODE_BINARYREADONLY

FILE_MODE_BINARYREADONLY is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



FILE_MODE_NORMAL

FILE_MODE_NORMAL is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



FILE_NO_VERSION

FILE_NO_VERSION is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



FILE_NOT_FOUND

FILE_NOT_FOUND is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



FILE_RD_ONLY

FILE_RD_ONLY is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



FILE_SIZE

FILE_SIZE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



FILE_SRC_EQUAL

FILE_SRC_EQUAL is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



FILE_SRC_OLD

FILE_SRC_OLD is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



FILE_TIME

FILE_TIME is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



FILE_WRITEABLE

FILE_WRITEABLE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



FILENAME

FILENAME is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



FILENAME_ONLY

FILENAME_ONLY is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment`)} [Feedback](#)



FINISHBUTTON

FINISHBUTTON is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



FIXED_DRIVE

FIXED_DRIVE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



FONT_TITLE

FONT_TITLE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



FREEENVSPACE

FREEENVSPACE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



FS_CREATEDIR

FS_CREATEDIR is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



FS_DISKONEREQUIRED

FS_DISKONEREQUIRED is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



FS_DONE

FS_DONE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



FS_FILENOTINLIB

FS_FILENOTINLIB is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



FS_GENEROR

FS_GENEROR is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



FS_INCORRECTDISK

FS_INCORRECTDISK is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



FS_LAUNCHPROCESS

FS_LAUNCHPROCESS is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



FS_OPERROR

FS_OPERROR is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



FS_OUTOFSPACE

FS_OUTOFSPACE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



FS_PACKAGING

FS_PACKAGING is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



FS_RESETREQUIRED

FS_RESETREQUIRED is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



FS_TARGETREADONLY

FS_TARGETREADONLY is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



FS_TONEXTDISK

FS_TONEXTDISK is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



FULL

FULL is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



FULLSCREEN

FULLSCREEN is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



FULLSCREENSIZE

FULLSCREENSIZE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



FULLWINDOWMODE

FULLWINDOWMODE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



GREATER_THAN

GREATER_THAN is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



GREEN

GREEN is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



HELP

HELP is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



HKEY_CLASSES_ROOT

HKEY_CLASSES_ROOT is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



HKEY_CURRENT_CONFIG

HKEY_CURRENT_CONFIG is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



HKEY_CURRENT_USER

HKEY_CURRENT_USER is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



HKEY_DYN_DATA

HKEY_DYN_DATA is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



HKEY_LOCAL_MACHINE

HKEY_LOCAL_MACHINE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



HKEY_PERFORMANCE_DATA

HKEY_PERFORMANCE_DATA is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



HKEY_USERS

HKEY_USERS is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



HOURGLASS

HOURGLASS is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



HWND_DESKTOP

HWND_DESKTOP is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



HWND_INSTALL

HWND_INSTALL is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



IGNORE_READONLY

IGNORE_READONLY is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



INCLUDE_SUBDIR

INCLUDE_SUBDIR is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



INDVFILESTATUS

INDVFILESTATUS is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



INFO

INFO is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



INFO_DESCRIPTION

INFO_DESCRIPTION is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



INFO_IMAGE

INFO_IMAGE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



INFO_MISC

INFO_MISC is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



INFO_SIZE

INFO_SIZE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



INFO_SUBCOMPONENT

INFO_SUBCOMPONENT is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



INFO_VISIBLE

INFO_VISIBLE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



INFORMATION

INFORMATION is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



INVALID_LIST

INVALID_LIST is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



IS_186

IS_186 is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment`)} [Feedback](#)



IS_286

IS_286 is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



IS_386

IS_386 is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment`)} [Feedback](#)



IS_486

IS_486 is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



IS_8514A

IS_8514A is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



IS_86

IS_86 is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



IS_ALPHA

IS_ALPHA is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



IS_CDROM

IS_CDROM is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



IS_CGA

IS_CGA is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



IS_DOS

IS_DOS is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



IS_EGA

IS_EGA is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



IS_FIXED

IS_FIXED is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



IS_FOLDER

IS_FOLDER is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



IS_ITEM

IS_ITEM is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment`)} [Feedback](#)



ISLANG_ALL

ISLANG_ALL is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_ARABIC

ISLANG_ARABIC is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_ARABIC_SAUDIARABIA

ISLANG_ARABIC_SAUDIARABIA is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_ARABIC_IRAQ

ISLANG_ARABIC_IRAQ is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



SLANG_ARABIC_EGYPT

ISLANG_ARABIC_EGYPT is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment`)} [Feedback](#)



ISLANG_ARABIC_LIBYA

ISLANG_ARABIC_LIBYA is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_ARABIC_ALGERIA

ISLANG_ARABIC_ALGERIA is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_ARABIC_MOROCCO

ISLANG_ARABIC_MOROCCO is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_ARABIC_TUNISIA

ISLANG_ARABIC_TUNISIA is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_ARABIC_OMAN

ISLANG_ARABIC_OMAN is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_ARABIC_YEMEN

ISLANG_ARABIC_YEMEN is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_ARABIC_SYRIA

ISLANG_ARABIC_SYRIA is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_ARABIC_JORDAN

ISLANG_ARABIC_JORDAN is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_ARABIC_LEBANON

ISLANG_ARABIC_LEBANON is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_ARABIC_KUWAIT

ISLANG_ARABIC_KUWAIT is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_ARABIC_UAE

ISLANG_ARABIC_UAE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_ARABIC_BAHRAIN

ISLANG_ARABIC_BAHRAIN is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment`)} [Feedback](#)



ISLANG_ARABIC_QATAR

ISLANG_ARABIC_QATAR is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_AFRIKAANS

ISLANG_AFRIKAANS is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_AFRIKAANS_STANDARD

ISLANG_AFRIKAANS_STANDARD is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment`)} [Feedback](#)



ISLANG_ALBANIAN

ISLANG_ALBANIAN is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_ALBANIAN_STANDARD

ISLANG_ALBANIAN_STANDARD is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_BASQUE

ISLANG_BASQUE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_BASQUE_STANDARD

ISLANG_BASQUE_STANDARD is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_BULGARIAN

ISLANG_BULGARIAN is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_BULGARIAN_STANDARD

ISLANG_BULGARIAN_STANDARD is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment`)} [Feedback](#)



ISLANG_BELARUSIAN

ISLANG_BELARUSIAN is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_BELARUSIAN_STANDARD

ISLANG_BELARUSIAN_STANDARD is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_CATALAN

ISLANG_CATALAN is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_CATALAN_STANDARD

ISLANG_CATALAN_STANDARD is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_CHINESE

ISLANG_CHINESE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_CHINESE_TAIWAN

ISLANG_CHINESE_TAIWAN is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_CHINESE_PRC

ISLANG_CHINESE_PRC is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_CHINESE_HONGKONG

ISLANG_CHINESE_HONGKONG is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_CHINESE_SINGAPORE

ISLANG_CHINESE_SINGAPORE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_CROATIAN

ISLANG_CROATIAN is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_CROATIAN_STANDARD

ISLANG_CROATIAN_STANDARD is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_CZECH

ISLANG_CZECH is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_CZECH_STANDARD

ISLANG_CZECH_STANDARD is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_DANISH

ISLANG_DANISH is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_DANISH_STANDARD

ISLANG_DANISH_STANDARD is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_DUTCH

ISLANG_DUTCH is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_DUTCH_STANDARD

ISLANG_DUTCH_STANDARD is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_DUTCH_BELGIAN

ISLANG_DUTCH_BELGIAN is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("", `comment`)} [Feedback](#)



ISLANG_ENGLISH

ISLANG_ENGLISH is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_ENGLISH_UNITEDSTATES

ISLANG_ENGLISH_UNITEDSTATES is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_ENGLISH_UNITEDKINGDOM

ISLANG_ENGLISH_UNITEDKINGDOM is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_ENGLISH_AUSTRALIAN

ISLANG_ENGLISH_AUSTRALIAN is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_ENGLISH_CANADIAN

ISLANG_ENGLISH_CANADIAN is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_ENGLISH_NEWZEALAND

ISLANG_ENGLISH_NEWZEALAND is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_ENGLISH_IRELAND

ISLANG_ENGLISH_IRELAND is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_ENGLISH_SOUTHAFRICA

ISLANG_ENGLISH_SOUTHAFRICA is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_ENGLISH_JAMAICA

ISLANG_ENGLISH_JAMAICA is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_ENGLISH_CARIBBEAN

ISLANG_ENGLISH_CARIBBEAN is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_ENGLISH_BELIZE

ISLANG_ENGLISH_BELIZE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment`)} [Feedback](#)



ISLANG_ENGLISH_TRINIDAD

ISLANG_ENGLISH_TRINIDAD is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment`)} [Feedback](#)



ISLANG_ESTONIAN

ISLANG_ESTONIAN is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_ESTONIAN_STANDARD

ISLANG_ESTONIAN_STANDARD is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_FAEROESE

ISLANG_FAEROESE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_FAEROESE_STANDARD

ISLANG_FAEROESE_STANDARD is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_FARSI

ISLANG_FARSI is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_FARSI_STANDARD

ISLANG_FARSI_STANDARD is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_FINNISH

ISLANG_FINNISH is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_FINNISH_STANDARD

ISLANG_FINNISH_STANDARD is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_FRENCH

ISLANG_FRENCH is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_FRENCH_STANDARD

ISLANG_FRENCH_STANDARD is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_FRENCH_BELGIAN

ISLANG_FRENCH_BELGIAN is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_FRENCH_CANADIAN

ISLANG_FRENCH_CANADIAN is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_FRENCH_SWISS

ISLANG_FRENCH_SWISS is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_FRENCH_LUXEMBOURG

ISLANG_FRENCH_LUXEMBOURG is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_GERMAN

ISLANG_GERMAN is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_GERMAN_STANDARD

ISLANG_GERMAN_STANDARD is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_GERMAN_SWISS

ISLANG_GERMAN_SWISS is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_GERMAN_AUSTRIAN

ISLANG_GERMAN_AUSTRIAN is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_GERMAN_LUXEMBOURG

ISLANG_GERMAN_LUXEMBOURG is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_GERMAN_LIECHTENSTEIN

ISLANG_GERMAN_LIECHTENSTEIN is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_GREEK

ISLANG_GREEK is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment`)} [Feedback](#)



ISLANG_GREEK_STANDARD

ISLANG_GREEK_STANDARD is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment`)} [Feedback](#)



ISLANG_HEBREW

ISLANG_HEBREW is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_HEBREW_STANDARD

ISLANG_HEBREW_STANDARD is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_HUNGARIAN

ISLANG_HUNGARIAN is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_HUNGARIAN_STANDARD

ISLANG_HUNGARIAN_STANDARD is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_ICELANDIC

ISLANG_ICELANDIC is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("", `comment')} [Feedback](#)



ISLANG_ICELANDIC_STANDARD

ISLANG_ICELANDIC_STANDARD is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_INDONESIAN

ISLANG_INDONESIAN is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_INDONESIAN_STANDARD

ISLANG_INDONESIAN_STANDARD is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_ITALIAN

ISLANG_ITALIAN is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment`)} [Feedback](#)



ISLANG_ITALIAN_STANDARD

ISLANG_ITALIAN_STANDARD is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_ITALIAN_SWISS

ISLANG_ITALIAN_SWISS is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_JAPANESE

ISLANG_JAPANESE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_JAPANESE_STANDARD

ISLANG_JAPANESE_STANDARD is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_KOREAN

ISLANG_KOREAN is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_KOREAN_STANDARD

ISLANG_KOREAN_STANDARD is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_KOREAN_JOHAB

ISLANG_KOREAN_JOHAB is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_LATVIAN

ISLANG_LATVIAN is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment`)} [Feedback](#)



ISLANG_LATVIAN_STANDARD

ISLANG_LATVIAN_STANDARD is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_LITHUANIAN

ISLANG_LITHUANIAN is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_LITHUANIAN_STANDARD

ISLANG_LITHUANIAN_STANDARD is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_NORWEGIAN

ISLANG_NORWEGIAN is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_NORWEGIAN_BOKMAL

ISLANG_NORWEGIAN_BOKMAL is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_NORWEGIAN_NYNORSK

ISLANG_NORWEGIAN_NYNORSK is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_POLISH

ISLANG_POLISH is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment`)} [Feedback](#)



ISLANG_POLISH_STANDARD

ISLANG_POLISH_STANDARD is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment`)} [Feedback](#)



ISLANG_PORTUGUESE

ISLANG_PORTUGUESE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_PORTUGUESE_STANDARD

ISLANG_PORTUGUESE_STANDARD is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_PORTUGUESE_BRAZILIAN

ISLANG_PORTUGUESE_BRAZILIAN is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_ROMANIAN

ISLANG_ROMANIAN is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_ROMANIAN_STANDARD

ISLANG_ROMANIAN_STANDARD is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_RUSSIAN

ISLANG_RUSSIAN is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment`)} [Feedback](#)



ISLANG_RUSSIAN_STANDARD

ISLANG_RUSSIAN_STANDARD is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_SLOVAK

ISLANG_SLOVAK is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



IISLANG_SLOVAK_STANDARD

IISLANG_SLOVAK_STANDARD is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_SLOVENIAN

ISLANG_SLOVENIAN is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_SLOVENIAN_STANDARD

ISLANG_SLOVENIAN_STANDARD is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment`)} [Feedback](#)



ISLANG_SERBIAN

ISLANG_SERBIAN is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment`)} [Feedback](#)



ISLANG_SERBIAN_LATIN

ISLANG_SERBIAN_LATIN is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_SERBIAN_CYRILLIC

ISLANG_SERBIAN_CYRILLIC is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment`)} [Feedback](#)



ISLANG_SPANISH

ISLANG_SPANISH is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_SPANISH_TRADITIONALSORT

ISLANG_SPANISH_TRADITIONALSORT is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_SPANISH_MEXICAN

ISLANG_SPANISH_MEXICAN is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_SPANISH_MODERNSORT

ISLANG_SPANISH_MODERNSORT is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_SPANISH_GUATEMALA

ISLANG_SPANISH_GUATEMALA is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_SPANISH_COSTARICA

ISLANG_SPANISH_COSTARICA is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment`)} [Feedback](#)



ISLANG_SPANISH_PANAMA

ISLANG_SPANISH_PANAMA is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_SPANISH_DOMINICANREPUBLIC

ISLANG_SPANISH_DOMINICANREPUBLIC is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_SPANISH_VENEZUELA

ISLANG_SPANISH_VENEZUELA is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_SPANISH_COLOMBIA

ISLANG_SPANISH_COLOMBIA is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment`)} [Feedback](#)



ISLANG_SPANISH_PERU

ISLANG_SPANISH_PERU is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_SPANISH_ARGENTINA

ISLANG_SPANISH_ARGENTINA is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_SPANISH_ECUADOR

ISLANG_SPANISH_ECUADOR is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_SPANISH_CHILE

ISLANG_SPANISH_CHILE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_SPANISH_URUGUAY

ISLANG_SPANISH_URUGUAY is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_SPANISH_PARAGUAY

ISLANG_SPANISH_PARAGUAY is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_SPANISH_BOLIVIA

ISLANG_SPANISH_BOLIVIA is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment`)} [Feedback](#)



SLANG_SPANISH_ELSALVADOR

ISLANG_SPANISH_ELSALVADOR is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment`)} [Feedback](#)



ISLANG_SPANISH_HONDURAS

ISLANG_SPANISH_HONDURAS is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_SPANISH_NICARAGUA

ISLANG_SPANISH_NICARAGUA is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_SPANISH_PUERTORICO

ISLANG_SPANISH_PUERTORICO is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_SWEDISH

ISLANG_SWEDISH is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_SWEDISH_STANDARD

ISLANG_SWEDISH_STANDARD is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_SWEDISH_FINLAND

ISLANG_SWEDISH_FINLAND is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_THAI

ISLANG_THAI is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_THAI_STANDARD

ISLANG_THAI_STANDARD is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_TURKISH

ISLANG_TURKISH is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_TURKISH_STANDARD

ISLANG_TURKISH_STANDARD is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment`)} [Feedback](#)



ISLANG_UKRAINIAN

ISLANG_UKRAINIAN is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



IISLANG_UKRAINIAN_STANDARD

IISLANG_UKRAINIAN_STANDARD is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_VIETNAMESE

ISLANG_VIETNAMESE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISLANG_VIETNAMESE_STANDARD

ISLANG_VIETNAMESE_STANDARD is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



IS_MIPS

IS_MIPS is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



IS_MONO

IS_MONO is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



IS_OS2

IS_OS2 is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISOSL_ALL

ISOSL_ALL is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISOSL_WIN31

ISOSL_WIN31 is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISOSL_WIN95

ISOSL_WIN95 is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISOSL_NT351

ISOSL_NT351 is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISOSL_NT351_ALPHA

ISOSL_NT351_ALPHA is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISOSL_NT351_MIPS

ISOSL_NT351_MIPS is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISOSL_NT40

ISOSL_NT40 is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISOSL_NT40_ALPHA

ISOSL_NT40_ALPHA is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISOSL_NT40_MIPS

ISOSL_NT40_MIPS is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



IS_PENTIUM

IS_PENTIUM is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



IS_POWERPC

IS_POWERPC is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



IS_RAMDRIVE

IS_RAMDRIVE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



IS_REMOTE

IS_REMOTE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



IS_REMOVABLE

IS_REMOVABLE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment`)} [Feedback](#)



IS_SVGA

IS_SVGA is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



IS_UNKNOWN

IS_UNKNOWN is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment`)} [Feedback](#)



IS_UVGA

IS_UVGA is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



IS_VALID_PATH

IS_VALID_PATH is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



IS_VGA

IS_VGA is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



IS_WIN32S

IS_WIN32S is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



IS_WINDOWS

IS_WINDOWS is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



IS_WINDOWS95

IS_WINDOWS95 is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



IS_WINDOWSNT

IS_WINDOWSNT is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



IS_WINOS2

IS_WINOS2 is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



IS_XVGA

IS_XVGA is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ISTYPE

ISTYPE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



LANGUAGE

LANGUAGE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



LANGUAGE_DRV

LANGUAGE_DRV is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



LESS_THAN

LESS_THAN is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



LINE_NUMBER

LINE_NUMBER is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment`)} [Feedback](#)



LISTBOX_ENTER

LISTBOX_ENTER is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



LISTBOX_SELECT

LISTBOX_SELECT is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



LISTFIRST

LISTFIRST is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



LISTLAST

LISTLAST is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



LISTNEXT

LISTNEXT is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



LISTPREV

LISTPREV is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



LOCKEDFILE

LOCKEDFILE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



LOGGING

LOGGING is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



LOWER_LEFT

LOWER_LEFT is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



LOWER_RIGHT

LOWER_RIGHT is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment`)} [Feedback](#)



MAGENTA

MAGENTA is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



MAINCAPTION

MAINCAPTION is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



MATH_COPROCESSOR

MATH_COPROCESSOR is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



MAX_STRING

MAX_STRING is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



MENU

MENU is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



METAFILE

METAFILE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



MMEDIA_AVI

MMEDIA_AVI is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



MMEDIA_MIDI

MMEDIA_MIDI is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



MMEDIA_PLAYASYNCH

MMEDIA_PLAYASYNCH is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



MMEDIA_PLAYCONTINUOUS

MMEDIA_PLAYCONTINUOUS is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



MMEDIA_PLAYSYNCH

MMEDIA_PLAYSYNCH is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



MMEDIA_STOP

MMEDIA_STOP is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment`)} [Feedback](#)



MMEDIA_WAVE

MMEDIA_WAVE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



MOUSE

MOUSE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



MOUSE_DRV

MOUSE_DRV is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment`)} [Feedback](#)



NETWORK

NETWORK is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



NETWORK_DRV

NETWORK_DRV is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment`)} [Feedback](#)



NEXT

NEXT is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



NEXTBUTTON

NEXTBUTTON is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



NO

NO is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



NO_SUBDIR

NO_SUBDIR is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



NO_WRITE_ACCESS

NO_WRITE_ACCESS is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



NONCONTIGUOUS

NONCONTIGUOUS is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



NONEXCLUSIVE

NONEXCLUSIVE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



NORMAL

NORMAL is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



NORMALMODE

NORMALMODE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



NOSET

NOSET is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



NOTEXISTS

NOTEXISTS is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



NOTRESET

NOTRESET is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



NOWAIT

NOWAIT is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



NULL

NULL is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



NUMBERLIST

NUMBERLIST is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment`)} [Feedback](#)



OFF

OFF is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



OK

OK is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ON

ON is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ONLYDIR

ONLYDIR is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



OS

OS is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment`)} [Feedback](#)



OSMAJOR

OSMAJOR is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



OSMINOR

OSMINOR is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



OTHER_FAILURE

OTHER_FAILURE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



OUT_OF_DISK_SPACE

OUT_OF_DISK_SPACE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



PARALLEL

PARALLEL is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



PARTIAL

PARTIAL is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment`)} [Feedback](#)



PATH

PATH is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



PATH_EXISTS

PATH_EXISTS is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



PAUSE

PAUSE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



PERSONAL

PERSONAL is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



PROFSTRING

PROFSTRING is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



PROGMAN

PROGMAN is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



RAM_DRIVE

RAM_DRIVE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



REAL

REAL is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



RECORDMODE

RECORDMODE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



RED

RED is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



REGDB_APPPATH

REGDB_APPPATH is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



REGDB_APPPATH_DEFAULT

REGDB_APPPATH_DEFAULT is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



REGDB_BINARY

REGDB_BINARY is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



REGDB_ERR_CONNECTIONEXISTS

REGDB_ERR_CONNECTIONEXISTS is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



REGDB_ERR_CORRUPTEDREGISTRY

REGDB_ERR_CORRUPTEDREGISTRY is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



REGDB_ERR_FILECLOSE

REGDB_ERR_FILECLOSE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



REGDB_ERR_FILENOTFOUND

REGDB_ERR_FILENOTFOUND is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



REGDB_ERR_FILEOPEN

REGDB_ERR_FILEOPEN is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



REGDB_ERR_FILEREAD

REGDB_ERR_FILEREAD is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



REGDB_ERR_INITIALIZATION

REGDB_ERR_INITIALIZATION is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



REGDB_ERR_INVALIDFORMAT

REGDB_ERR_INVALIDFORMAT is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



REGDB_ERR_INVALIDHANDLE

REGDB_ERR_INVALIDHANDLE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



REGDB_ERR_INVALIDNAME

REGDB_ERR_INVALIDNAME is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



REGDB_ERR_INVALIDPLATFORM

REGDB_ERR_INVALIDPLATFORM is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



REGDB_ERR_OUTOFMEMORY

REGDB_ERR_OUTOFMEMORY is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



REGDB_ERR_REGISTRY

REGDB_ERR_REGISTRY is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



REGDB_KEYS

REGDB_KEYS is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



REGDB_NAMES

REGDB_NAMES is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



REGDB_NUMBER

REGDB_NUMBER is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



REGDB_STRING

REGDB_STRING is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



REGDB_STRING_EXPAND

REGDB_STRING_EXPAND is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



REGDB_STRING_MULTI

REGDB_STRING_MULTI is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



REGDB_UNINSTALL_NAME

REGDB_UNINSTALL_NAME is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



REGKEY_CLASSES_ROOT

REGKEY_CLASSES_ROOT is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



REGKEY_CURRENT_USER

REGKEY_CURRENT_USER is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



REGKEY_LOCAL_MACHINE

REGKEY_LOCAL_MACHINE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



REGKEY_USERS

REGKEY_USERS is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



REMOTE_DRIVE

REMOTE_DRIVE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



REMOVE

REMOVE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



REMOVEABLE_DRIVE

REMOVEABLE_DRIVE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



REPLACE

REPLACE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



REPLACE_ITEM

REPLACE_ITEM is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment`)} [Feedback](#)



RESET

RESET is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment`)} [Feedback](#)



RESTART

RESTART is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ROOT

ROOT is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



ROTATE

ROTATE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



RUN_MAXIMIZED

RUN_MAXIMIZED is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



RUN_MINIMIZED

RUN_MINIMIZED is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



RUN_SEPARATEMEMORY

RUN_SEPARATEMEMORY is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



SELECTFOLDER

SELECTFOLDER is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



SELFREGISTER

SELFREGISTER is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment`)} [Feedback](#)



SELFREGISTERBATCH

SELFREGISTERBATCH is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



SELFREGISTRATIONPROCESS

SELFREGISTRATIONPROCESS is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment`)} [Feedback](#)



SERIAL

SERIAL is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment`)} [Feedback](#)



SET

SET is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



SETUPTYPE

SETUPTYPE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



SETUPTYPE_INFO_DESCRIPTION

SETUPTYPE_INFO_DESCRIPTION is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



SETUPTYPE_INFO_DISPLAYNAME

SETUPTYPE_INFO_DISPLAYNAME is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



SEVERE

SEVERE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



SHARE

SHARE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



SHAREDFILE

SHAREDFILE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



SILENTMODE

SILENTMODE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



SPLITCOMPRESS

SPLITCOMPRESS is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



SPLITCOPY

SPLITCOPY is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



SRCTARGETDIR

SRCTARGETDIR is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



STANDARD

STANDARD is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



STATUS

STATUS is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



STATUS95

STATUS95 is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment`)} [Feedback](#)



STATUSBAR

STATUSBAR is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



STATUSDLG

STATUSDLG is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



STATUSEX

STATUSEX is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



STATUSOLD

STATUSOLD is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



STRINGLIST

STRINGLIST is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



STYLE_BOLD

STYLE_BOLD is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



STYLE_ITALIC

STYLE_ITALIC is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment`)} [Feedback](#)



STYLE_NORMAL

STYLE_NORMAL is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



STYLE_SHADOW

STYLE_SHADOW is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



STYLE_UNDERLINE

STYLE_UNDERLINE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



SW_HIDE

SW_HIDE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



SW_MAXIMIZE

SW_MAXIMIZE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



SW_MINIMIZE

SW_MINIMIZE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



SW_NORMAL

SW_NORMAL is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



SW_RESTORE

SW_RESTORE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



SW_SHOW

SW_SHOW is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



SW_SHOWMAXIMIZED

SW_SHOWMAXIMIZED is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



SW_SHOWMINIMIZED

SW_SHOWMINIMIZED is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



SW_SHOWMINNOACTIVE

SW_SHOWMINNOACTIVE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



SW_SHOWNA

SW_SHOWNA is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



SW_SHOWNOACTIVATE

SW_SHOWNOACTIVATE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



SW_SHOWNORMAL

SW_SHOWNORMAL is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



SYS_BOOTMACHINE

SYS_BOOTMACHINE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



SYS_BOOTWIN

SYS_BOOTWIN is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment`)} [Feedback](#)



SYS_BOOTWIN_INSTALL

SYS_BOOTWIN_INSTALL is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



SYS_RESTART

SYS_RESTART is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



SYS_SHUTDOWN

SYS_SHUTDOWN is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



SYS_TODOS

SYS_TODOS is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



TEXT

TEXT is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



TILED

TILED is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



TIME

TIME is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



TRUE

TRUE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



TYPICAL

TYPICAL is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



UPPER_LEFT

UPPER_LEFT is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



UPPER_RIGHT

UPPER_RIGHT is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



USER_ADMINISTRATOR

USER_ADMINISTRATOR is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



VALID_PATH

VALID_PATH is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



VARIABLE_LEFT

VARIABLE_LEFT is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



VARIABLE_UNDEFINED

VARIABLE_UNDEFINED is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



VER_DLL_NOT_FOUND

VER_DLL_NOT_FOUND is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



VER_UPDATE_ALWAYS

VER_UPDATE_ALWAYS is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



VER_UPDATE_COND

VER_UPDATE_COND is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



VERSION

VERSION is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



VIDEO

VIDEO is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



VOLUMELABEL

VOLUMELABEL is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment`)} [Feedback](#)



WAIT

WAIT is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment`)} [Feedback](#)



WARNING

WARNING is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



WELCOME

WELCOME is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment`)} [Feedback](#)



WHITE

WHITE is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment`)} [Feedback](#)



WIN32SINSTALLED

WIN32SINSTALLED is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



WIN32SMAJOR

WIN32SMAJOR is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



WIN32SMINOR

WIN32SMINOR is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



WINDOWS_SHARED

WINDOWS_SHARED is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



WINMAJOR

WINMAJOR is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment`)} [Feedback](#)



WINMINOR

WINMINOR is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



XCOPY_DATETIME

XCOPY_DATETIME is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



YELLOW

YELLOW is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment')} [Feedback](#)



YES

YES is a predefined constant used to represent a value that is passed to or returned by one or more built-in functions. You cannot change the value of a predefined constant.



[Click here for a list of functions that use this constant.](#)

{button ,PI("",`comment`)} [Feedback](#)



Warning C7501

Message

'name' : macro redefinition

Description

The identifier specified by *name* has already been used in a #define statement.

Notes

If the second definition is simply a duplicate of the first, delete it. If you intended the second definition to be different from the first, rename the identifier in the second definition and update all statements in the script that reference it.

{button ,AL(`#define',0,`,`')} [See also](#)

{button ,PI("",`comment')} [Feedback](#)



Warning C7502

Message

'size' : string/array size exceeds recommended limit

Description

The numeric value specified by *size* indicates the size of a string or array that is larger than InstallShield's recommended maximum.

Notes

To avoid unexpected effects, reduce the size of the array.

```
{button ,AL(`String indexing;String size',0,`,`')}
```

[See also](#)

```
{button ,PI("",`comment')} Feedback
```



Warning C7503

Message

'name' : function defined but never called

Description

The function specified by *name* is not called in your setup.

Notes

To reduce the size of your setup, remove the unused function from your script.

{button ,PI("",`comment`)} [Feedback](#)



Error C8001

Message

'program' : multiple main programs defined

Description

The keyword program was encountered after the main program block.

Notes

- n A script can contain only one main program block (beginning with the keyword program and ending with the keyword endprogram. Restructure your script to use only one main program block.
- n If your script consists of multiple source files, this error message will occur if more than one of those source files contains a main program block.

{button ,AL(`Structure of a setup script;Declarations;Program block',0,`,`)}}

[See also](#)

{button ,PI("",`comment')} [Feedback](#)



Error C8002

Message

'text' : function name expected

Description

The compiler expected to encounter a function name at the location indicated by *text*.

Notes

- n Check for a missing or mistyped function name before a parameter list.
- n Check for missing semicolons in preceding lines.

{button ,AL(`Building a function',0,`,`')}

[See also](#)

{button ,PI("",`comment')} [Feedback](#)



Error C8003

Message

'name' : function has no prototype declaration

Description

The function specified by *name* has not been declared in a prototype statement.

Notes

- n If the function has not been declared, insert a prototype statement before the current block (main program or function definition).
- n If the function has been declared ahead of the current block, compare the function declaration in the prototype statement with the function header in the function definition. Be sure the spelling of the function name is identical in both places and that the number and types of parameters matches.
- n If the function has been declared later in the script, move the declaration before the current block.

{button ,AL('Declaring functions',0,'')} }

[See also](#)

{button ,PI("",`comment`)} [Feedback](#)



Error C8004

Message

'name' : identifier already declared

Description

The identifier specified by *name* has already been declared in the current block. Identifiers can be declared only once in the same block.

Notes

- n Examine the main program block or function block where the error occurred to find the first declaration of the identifier. If the second declaration is simply a duplicate of the first, delete it. If you intended the second declaration to be a different variable than the first, rename the identifier in the second declaration and update all statements in the script that reference it.
- n If you cannot find a previous declaration of the variable, you may be declaring an identifier whose name is reserved by InstallShield. Reserved words are syntax colored in the Script Editor.

{button ,AL('Reserved words',0,'')}

[See also](#)

{button ,PI('','comment')} [Feedback](#)



Error C8005

Message

'name' : function was declared as DLL function

Description

The function definition for the function specified by *name* is invalid because that function has been declared in a prototype statement as a DLL function.

Notes

- n If the function is not a DLL function, remove the DLL file name from the function prototype.
- n If the name of the user-defined function conflicts with the DLL function name, rename the user-defined function and update all statements in the script that reference it. Be sure the user-defined function has been declared in a prototype statement.

```
{button ,AL(`DLL Function;Call a custom DLL function from unInstallShield',0,`,`')}
```

[See also](#)

```
{button ,PI("",`comment')} Feedback
```



Error C8006

Message

'text' : missing '(' after function name

Description

The compiler expected to find the opening parenthesis that follows a function name in a function definition; the character or characters indicated by *text* were encountered instead.

Notes

- n Check that the function's parameter list is enclosed within parentheses. If the function takes no parameters, the parentheses will be empty.
- n Verify that only space characters come between the function name and its parameter list.

{button ,AL(`Building a function;Calling a function;Declaring functions',0,'')} [See also](#)

{button ,PI("",`comment')} [Feedback](#)



Error C8007

Message

'text' : comma expected

Description

The character or characters specified by *text* were encountered at a location where a comma was expected.

Notes

- n If the error occurred in a function declaration, then a comma required to separate parameter declarations is probably missing.
- n If the error occurred in a function call, then a comma required to separate parameters is probably missing.

{button ,AL(`Building a function;Calling a function;Declaring functions',0,`,`)}}

[See also](#)

{button ,PI("",`comment')} [Feedback](#)



Error C8008

Message

'text' : identifier expected

Description

The character or characters specified by *text* were encountered at a location where an identifier was expected.

Notes

- n If this error occurs in a variable declaration, check that the data type is followed by an identifier, with no intervening punctuation; check that the identifier contains only allowable characters.
- n If this error occurs in a function call, check that each pair of argument is delimited by one and only one comma.

{button ,AL(`Variable data;Building a function;Calling a function;Declaring functions',0,`,`')} [See also](#)

{button ,PI("",`comment')} [Feedback](#)



Error C8009

Message

'text' : too many parameters

Description

The character or characters specified by *text* were encountered in a function call parameter list that contains more parameters than have been declared for the function.

Notes

To identify the argument or arguments that do not belong in the function call, compare the number and types of arguments in the function call to the number and types of parameters declared for that function.

{button ,AL(`Building a function;Calling a function;Declaring functions',0,`,`)}}

[See also](#)

{button ,PI("",`comment')} [Feedback](#)



Error C8010

Message

'text' : missing right parenthesis

Description

The character or characters specified by *text* were encountered at a location where a right parenthesis was expected.

Notes

This error is usually triggered when the right parentheses is missing from the parameter list of a function header in a function definition.

```
{button ,AL(`Building a function;Calling a function;Declaring functions',0,`,`)}}
```

[See also](#)

```
{button ,PI("",`comment')} Feedback
```



Error C8011

Message

'end' : missing 'begin' at start of function

Description

The keyword begin is missing in a function declaration.

Notes

Insert the keyword begin before the first statement in the function. Local variable declarations should appear between the function header and the keyword begin. The function header itself should not be terminated with a semi-colon.

{button ,AL(`Building a function',0,`,`')}

[See also](#)

{button ,PI("",`comment')} [Feedback](#)



Error C8012

Message

'text' :semicolon expected

Description

The character or characters specified by *text* were encountered at a location where a semicolon was expected.

Notes

Insert a semicolon to terminate the statement that appears just before the error location.

{button ,PI("",`comment`)} [Feedback](#)



Error C8013

Message

unexpected end of source file

Description

The compiler reached the end of the source file without encountering the keyword required to close the current program or function block

Notes

- n If your script contains only a main program, check that the keyword `endprogram` appears as the last statement of the main program (before the function block begins). The `endprogram` statement should not be terminated with a semicolon.
- n If your script contains function definitions, check that each function definition is terminated with the keyword `end` and a semicolon.
- n Check that typedef blocks are closed with an end statement.
- n Check that all flow control blocks are correctly closed.
- n Check that all `#if`, `#ifdef`, and `#ifndef` statements have a matching `#endif` statement.

{button ,AL(`Structure of a setup script;Program block;Function block;Flow control',0,`,`')} [See also](#)

{button ,PI(`",`comment')} [Feedback](#)



Error C8014

Message

'name' : identifier already defined

Description

The identifier specified by *name* has already been declared in the structure.

Notes

Examine the script and any scripts included before the error location to find the first declaration of the identifier. If the second declaration is simply a duplicate of the first, delete it. If you intended the second declaration to create a different identifier than the first, rename the identifier in the second declaration and update all statements in the script that reference it.

{button ,AL(`Data structures',0,`,`)} [See also](#)

{button ,PI("",`comment')} [Feedback](#)



Error C8015

Message

'name' : member name already defined

Description

The member identified by *name* has already been declared in the structure.

Notes

Examine the structure in which the error occurred to find the first declaration of the member. If the second declaration is simply a duplicate of the first, delete it. If you intended the second declaration to be a different member than the first, rename the member in the second declaration and update all statements in the script that reference it.

{button ,AL(`Data structures',0,`,`)} [See also](#)

{button ,PI("",`comment')} [Feedback](#)



Error C8016

Message

'name ' : undefined label:

Description

The identifier specified by *name* is used in a goto statement to reference a label that has not been defined.

Notes

The error is located in the block directly above the error location. Find the goto statement that references name. If the goto statement is in a program block, it must reference a label that has been defined in the program block. If the goto statement is in a function definition, it must reference a label that has been defined in that function definition. Determine where you want control to flow and define the label at that location.

{button ,AL(`goto',0,`,`')} [See also](#)

{button ,PI("",`comment')} [Feedback](#)



Error C8017

Message

'text' : expected typedef (struct) name

Description

The character or characters specified by *text* were encountered at a location where a typedef declaration was expected.

Notes

- n Check that a data type has been specified for the variable being declared. Verify that the reserved word for that data type is spelled correctly and is in all upper case.
- n If more than one variable is declared for that data type, check that the individual identifier names are separated by commas and not some other punctuation mark.
- n Verify that the data declaration is terminated with a semicolon.

{button ,AL(`Declarations',0,`,`')}` [See also](#)

{button ,PI("",`comment')}` [Feedback](#)



Error C8019

Message

'text' : expected type declaration

Description

The character or characters specified by *text* were encountered at a location where a data declaration was expected.

Notes

This error generally occurs when a semicolon is missing at the end of a data declaration in a line ahead of the error location. It also occurs in a function definition when the function header is terminated with a semicolon.

{button ,AL(`Declarations',0,`,`')}} [See also](#)

{button ,PI("",`comment')}} [Feedback](#)



Error C8020

Message

'text' : parameter list expected

Description

The character or characters specified by *text* were encountered immediately after a prototype declaration for which there was no parameter list.

Notes

Specify the parameter list for the function above the error location. If the function takes no parameters, specify an empty list by placing opening and closing parenthesis after the function name.

```
{button ,AL(`Declaring functions',0,`,`')}
```

[See also](#)

```
{button ,PI("",`comment')} Feedback
```



Error C8021

Message

'text' : missing 'begin' after 'typedef'

Description

The character or characters specified by *text* were encountered in a typedef statement where the keyword `begin` was expected. In a typedef statement, the keyword `begin` must follow the structure name.

Notes

Insert the keyword `begin` after the structure name, with no intervening punctuation.

{button ,AL(`Data structures',0,`,`')} [See also](#)

{button ,PI("",`comment')} [Feedback](#)



Error C8022

Message

'text' : comma or semicolon expected

Description

The character or characters specified by *text* were encountered at a location where a comma or semicolon was expected.

Notes

If this error occurs near a data declaration, check that the identifier is valid and that the data declaration is terminated with a semicolon. If two or more identifiers are being declared in the same statement, verify that they are delimited by commas.

{button ,AL('Declarations',0,','),''} [See also](#)

{button ,PI('','comment')} [Feedback](#)



Error C8023

Message

'text' : right bracket expected

Description

The character or characters specified by *text* were encountered at a location where a right bracket was expected.

Notes

This error occurs when the closing right bracket is missing from a reference to an array.

{button ,PI("",`comment`)} [Feedback](#)



Error C8024

Message

'text' : invalid array/string size

Description

The character or characters specified by *text* may not be used to declare the size of a string or array.

Notes

- n If the size indicator is a constant, verify that it is spelled correctly.
- n Verify that the size of the array is greater than 0.

{button ,AL(`String indexing',0,'')} [See also](#)

{button ,PI("",`comment')} [Feedback](#)



Error C8025

Message

'text' : undefined identifier

Description

The identifier specified by *text* has not been declared.

Notes

All identifiers in a script must be declared before they can be referenced in the main program block or in a function block. Check that the identifier has been declared. If it has, check that spelling used in the declaration matches the spelling of the identifier at the error location.

{button ,AL(`Declarations',0,`,`')} [See also](#)

{button ,PI("",`comment')} [Feedback](#)



Error C8026

Message

'text' : invalid use of identifier

Description

The identifier specified by *text* is being used incorrectly in a statement.

Notes

This error message can be triggered by a number of errors, such as the use of a structure type name, rather than the structure name, in an operation. Examine the statement at the error location and check that all identifiers in the statement are being used correctly.

{button ,PI("",`comment')} [Feedback](#)



Error C8027

Message

missing operand

Description

The compiler was unable to find the operand that was expected after an operator.

Notes

This error message may be displayed when the compiler encounters an incomplete statement ending with an operator at the end of the source file.

{button ,AL(`Operators',0,`,`')} [See also](#)

{button ,PI("",`comment')} [Feedback](#)



Error C8028

Message

'text' : unresolved operator

Description

The character or characters specified by *text* were encountered after an expression that could not be resolved.

Notes

This error occurs when an operand is missing from an expression. Examine each operator in the expression and make sure that each one has the operands it requires.

{button ,AL(`Operators',0,`,`')} [See also](#)

{button ,PI("",`comment')} [Feedback](#)



Error C8031

Message

'text : uncalled function

Description

The character or characters specified by *text* were encountered after a function name, where the opening parenthesis of the argument list was expected.

Notes

This error occurs in statements in which the result of a function call is assigned to a variable. The message is triggered by an error in the argument list that follows the function name.

- n Check that the argument list is delimited from the function name by an opening parenthesis.
- n If the argument list was not specified, insert it after the function name; when calling a function that takes no parameters, place empty set of opening and closing parentheses after the function name.

{button ,AL(`Building a function;Calling a function;Declaring functions',0,','')}

[See also](#)

{button ,PI("",`comment')} [Feedback](#)



Error C8032

Message

'text' : missing member reference

Description

The character or characters specified by *text* were encountered at a location where a member name was expected.

Notes

The line of code at the error location references a structure rather than a member of a structure. Determine which member of the structure you intended to reference and insert that member name. Remember to use the member operator to delimit the structure name and the member name.

{button ,AL(`Data structures;Member operator',0,`,`)} [See also](#)

{button ,PI("",`comment')} [Feedback](#)



Error C8033

Message

'text' : unsubscripted array

Description

The identifier that precedes the character or characters specified by *text* is an unsubscripted array variable. InstallShield does not allow references to a complete array structure. You must reference elements in the array individually.

Notes

Determine which element of the array you want to reference and indicate its position in the array with a subscript. The subscript must be placed within square brackets and positioned immediately after the array name.

{button ,PI("",`comment`)} [Feedback](#)



Error C8034

Message

'text' : missing expression

Description

The character or characters specified by *text* were encountered at a location where an expression was expected.

Notes

- n This error generally occurs in an assignment statement when the value to be assigned is missing. Insert an expression to the right of the assignment operator.
- n If the error occurs in a statement that references a constant, verify that the definition of that constant includes both the constant name and a value.

{button ,AL(`#define;Assignment operator',0,`,`')} [See also](#)

{button ,PI(`",`comment')} [Feedback](#)



Error C8035

Message

'text' : statement label required

Description

The character or characters specified by *text* were encountered after the keyword `goto`, where a label is required.

Notes

Add a label after the keyword `goto` or delete the `goto` statement.

{button ,AL(`goto',0,`,`')} [See also](#)

{button ,PI("",`comment')} [Feedback](#)



Error C8036

Message

'name' : too many arguments

Description

The function specified by *name* is being called with too many arguments.

Notes

Compare the arguments in the function call with the parameter list in the declaration for that function. Delete the argument or arguments in the function call that are not defined in the declaration.

{button ,AL(`Building a function;Calling a function;Declaring functions',0,`,`)}}

[See also](#)

{button ,PI("",`comment')} [Feedback](#)



Error C8037

Message

'text' : variable required

Description

The constant, literal value or structure member specified by *text* is being passed as an argument in a parameter position that requires a variable.

Notes

- n Declare a variable of the type required for the parameter; assign the constant or literal value to that variable; then pass the variable instead of the constant or literal value.
- n InstallShield does not allow a structure member to be passed by reference. Declare a variable of the same type as the structure member; assign the value of the structure member to the new variable; then pass the variable instead. When the function returns, assign the value of variable to the structure member.

{button ,AL(^ Building a function;Calling a function;Declaring functions;BYREF operator',0,'')}

[See also](#)

{button ,PI("",`comment')} [Feedback](#)



Error C8038

Message

'text' : numeric value required

Description

The non-numeric argument specified by *text* was passed in a parameter position requiring a numeric argument.

Notes

Make sure all arguments passed are of the correct data types for the parameter positions they occupy. Also make sure no arguments are missing.

{button ,AL(`Building a function;Calling a function;Declaring functions',0,`,`)}`} [See also](#)

{button ,PI("",`comment')} } [Feedback](#)



Error C8039

Message

'value' : string value required

Description

The number specified by *value* was encountered in an expression, statement, or function argument that requires string data.

Notes

- n In a string operation, check that all operands and operators are compatible with string data.
- n In an assignment statement, verify that the value you are assigning to a string variable is itself a string.
- n In a function call, be sure that you are not passing non-string data in a parameter position that requires a string.

{button ,AL(`Data Types;String comparisons;String operators;Assignment operator;Calling a function',0,`,`)} See also

{button ,PI(`",`comment')} Feedback



Error C8040

Message

'name' : incomplete argument list

Description

One or more arguments are missing in the call to the function specified by *name*.

Notes

- n To identify the missing argument, compare the number and types of arguments in the function call to the number and types of parameters for that function.
- n If all of the arguments are present, check that they are delimited by commas.

{button ,AL(`Building a function;Calling a function;Declaring functions',0,`,`')}} [See also](#)

{button ,PI("",`comment')} [Feedback](#)



Error C8042

Message

'[' : string or array type required

Description

The identifier to the left of the bracket must be a string or array type.

Notes

This error occurs when you specify an index value for a variable that is not a string or array. Examine the identifier to the left of the bracket. If it is not a string or array, delete the index value or declare the variable as a string or array.

{button ,AL(`String indexing',0,'')} [See also](#)

{button ,PI("",`comment')} [Feedback](#)



Error C8043

Message

'text' : typedef pointer type required

Description

The character or characters specified by *text* can be used only with a pointer to a structure.

Notes

A common reason for this error is the use of a structure pointer operator between a structure name and a member name. The structure pointer operator can be used only to specify the member of a record that is referenced with a pointer to a structure. Replace the structure pointer operator with a member operator.

{button ,AL(`Data structures;Member operator;Structure pointer operator',0,`,`')} [See also](#)

{button ,PI("",`comment')} [Feedback](#)



Error C8044

Message

'text' : typedef type required

Description

The character or characters specified by *text* can be used only with a structure that has been declared in a typedef statement.

Notes

A common reason for this error is the use of a member operator between a pointer to a structure and a member name. The member operator can be used only to specify the member of a structure that is referenced with a structure name. Replace the member operator with a structure pointer operator.

{button ,AL(`Data structures;Member operator;Structure pointer operator',0,`,``)} [See also](#)

{button ,PI("",`comment')} [Feedback](#)



Error C8045

Message

'name' : member name not found

Description

The member specified by *name* does not exist in the structure you are referencing.

Notes

- n Check the structure definition for the specified member.
- n Check the spelling of the member name.

{button ,AL(`Data structures;Member operator',0,`,`)} [See also](#)

{button ,PI("",`comment')} [Feedback](#)



Error C8046

Message

numeric variable required

Description

A non-numeric argument was passed in a parameter position requiring a numeric variable.

Notes

Replace the argument with a numeric variable.

```
{button ,AL(^Building a function;Calling a function;Declaring functions',0,'')}
```

[See also](#)

```
{button ,PI("",`comment')} Feedback
```



Error C8047

Message

'&' : can only take address of variable

Description

The compiler encountered an address operation on a constant or literal.

Notes

The address operator (&) can be used only with a variable.

{button ,AL(`Address operator',0,`,`')}

[See also](#)

{button ,PI("",`comment')} [Feedback](#)



Error C8048

Message

macro name missing

Description

The #define statement at the error location has no macro name.

Notes

Insert a macro name after the #define preprocessor directive and before the macro value.

{button ,AL(`#define',0,`,`')} [See also](#)

{button ,PI("",`comment')} [Feedback](#)



Error C8049

Message

missing expression for #if/#elif

Description

The #if or #elif preprocessor directive is not followed by a valid expression to determine the flow of compilation.

Notes

This error message is triggered if an expression has not been specified after an #if or #elif keyword. It can also be triggered if an invalid expression follows one of those directives.

{button ,AL(`#if. . #else. . #endif;#elif,0,`,``)} [See also](#)

{button ,PI(`",`comment`)} [Feedback](#)



Error C8050

Message

'text' : invalid expression for #if/#elif

Description

The character or characters specified by *text* are invalid in the expression following the #if or #elif preprocessor command.

Notes

This error message is triggered if an invalid expression after an #if or #elif keyword.

{button ,AL(`#if. . #else. . #endif;#elif,0,`,``)} [See also](#)

{button ,PI("",`comment`)} [Feedback](#)



Error C8051

Message

missing end of string literal

Description

The string literal in a #define statement has no closing quotation mark.

Notes

Look for a #define statement above the line on which the error was detected. For a string constant, enclose the literal in opening and closing quotation marks. Numeric constants should not be enclosed within quotation marks.

{button ,AL(`#define',0,`,`')} [See also](#)

{button ,PI("",`comment')} [Feedback](#)



Error C8052

Message

macro expansion text too large

Description

The text specified for the identifier in a #define statement exceeds 256 characters.

Notes

Either shorten the string specified for that identifier or divide the text into two or more shorter strings and specify each in a #define statement. Then concatenate the strings in your program.

{button ,AL(`#define',0,`,`')} [See also](#)

{button ,PI("",`comment')} [Feedback](#)



Error C8053

Message

'name' : identifier not a #define macro

Description

The identifier specified by *name*, which is used in an #undef statement, has not been defined.

Notes

Check the spelling of the identifier specified by *name*.

{button ,AL(`#define;#undef,0,`,`)} [See also](#)

{button ,PI(`",`comment')} [Feedback](#)



Error C8054

Message

include file name missing

Description

The preprocessor directive `#include` does not specify the name of the file to include.

Notes

Specify the name of a source file you want to include or delete the preprocessor directive.

`{button ,AL(`#include',0,`,`')}` [See also](#)

`{button ,PI("",`comment')}` [Feedback](#)



Error C8055

Message

#elif not preceded by #if

Description

The compiler encountered an #elif directive that was not preceded by an #if directive.

Notes

Examine the statements that precede the error location and restructure the conditional compile directives to start with an #if directive.

{button ,AL(`#if. . #else. . #endif;#elif,0,`,``)} [See also](#)

{button ,PI(`",`comment`)} [Feedback](#)



Error C8057

Message

#else not preceded by #if

Description

The compiler encountered an #else directive that was not preceded by an #if directive.

Notes

Examine the statements that precede the error location and restructure the conditional compile directives to start with an #if directive.

{button ,AL(`#if. . #else. . #endif',0,'')} [See also](#)

{button ,PI("",`comment')} [Feedback](#)



Error C8058

Message

#endif not preceded by #if

Description

The compiler encountered an #endif directive that was not preceded by an #if directive.

Notes

Examine the statements that precede the error location and restructure the conditional compile directives to start with an #if directive.

{button ,AL(`#if. . #else. . #endif',0,'')} [See also](#)

{button ,PI("",`comment')} [Feedback](#)



Error C8059

Message

'text' : unrecognized preprocessor command

Description

The character or characters specified by *text* and preceded by the symbol # is not recognized as a preprocessor command.

Notes

Verify that the preprocessor command is valid in an InstallShield script and that it is spelled correctly.

{button ,AL('Preprocessor statements',0,'')}[See also](#)

{button ,PI('','comment')}[Feedback](#)



Error C8062

Message

'text' : constant operand required

Description

The character or characters specified by *text* were encountered in a statement where a constant was expected.

Notes

This error message is triggered in a `switch..endswitch` block when the keyword `case` is not followed by a constant or literal.

`{button ,AL(`switch',0,`,`)}` [See also](#)

`{button ,PI("",`comment')}` [Feedback](#)



Error C8063

Message

input line too long

Description

A source code line exceeds 1,024 characters.

Notes

Divide the line into two or more lines.

{button ,PI("",`comment`)} [Feedback](#)



Error C8064

Message

unterminated comment

Description

The comment that begins at the error location is not terminated.

Notes

Find the end of the comment. Insert comment termination characters or correct the comment termination characters if they were entered incorrectly.

```
{button ,AL(`Writing comments',0,`,`'')}
```

[See also](#)

```
{button ,PI("",`comment')} Feedback
```



Error C8065

Message

string literal exceeds 255 characters

Description

The string literal at the error location has more than 255 characters.

Notes

Divide the string literal into two or more string literals and concatenate them.

{button ,AL(`Concatenate',0,`,`')} [See also](#)

{button ,PI("",`comment')} [Feedback](#)



Error C8066

Message

missing #endif statement at end of file

Description

An #if directive that appears in the script does not have a corresponding #endif directive.

Notes

Examine the preprocessor directives in your script. Make sure that all #if directives have a corresponding #endif directive.

{button ,AL(`#if. . #else. . #endif',0,'')} [See also](#)

{button ,PI("",`comment')} [Feedback](#)



Error C8067

Message

'number' : integer constant too large

Description

The value specified by *number* is too large for the number variable.

Notes

The valid range of NUMBER data is -2147483648 and +2147483647. If you specify a value in the range +2147483647 to +4294967295, that value will overflow and the result will be a negative number. If the value is greater than 4294967299, this error message will be displayed.

{button ,AL(`Data types',0,`,`)} [See also](#)

{button ,PI(`",`comment')} [Feedback](#)



Error C8068

Message

'char' : unrecognized character encountered

Description

The character specified by *char* was not recognized by the compiler.

Notes

Certain characters, such as "\$" and "{", are not recognized by the compiler.

{button ,PI("",`comment`)} [Feedback](#)



Error C8069

Message

'#' : preprocessor command must be first on line

Description

The preprocessor directive identified by the symbol # was encountered at a location where it is not allowed. A preprocessor directive must be the first keyword in any line where it appears.

Notes

The symbol # is reserved to identify preprocessor directives. It has no other valid use in InstallScript. Examine the line that produced the error. If the symbol # appears in an identifier, rename that identifier there and everywhere else in the script where it is referenced.

{button ,AL('Preprocessor statements',0,'')}[See also](#)

{button ,PI('','comment')} [Feedback](#)



Error C8070

Message

'text' : string constant expected

Description

The character or characters specified by *text* were encountered in a statement where a string constant was expected.

Notes

This error message is triggered in a `switch..endswitch` block when the keyword `case` is not followed by a string constant. In a `switch..endswitch` block, the constants specified in the case statements must be the same data type as the variable or expression result specified in the switch statement.

{button ,AL(`switch',0,`,`')} [See also](#)

{button ,PI("",`comment')} [Feedback](#)



Error C8071

Message

colon expected

Description

The compiler expected a colon in the statement.

Notes

This error occurs in a switch statement when the constant(s) after the keyword case is not followed by a colon. Insert a colon between the constant(s) and the statement(s) for that case.

{button ,AL(`switch',0,`,`')} [See also](#)

{button ,PI("",`comment')} [Feedback](#)



Error C8072

Message

'elseif' : 'elseif' cannot follow 'else'

Description

The keyword elseif was encountered after an else statement in an if..endif block.

Notes

The keyword elseif cannot be used in an if..endif block after an else statement. Restructure the if..endif block to avoid this error.

{button ,AL(`if,0`,``)} [See also](#)

{button ,PI(`",`comment`)} [Feedback](#)



Error C8073

Message

'text' : missing 'then' keyword

Description

The character or characters specified by text were encountered in an if block where the keyword then was expected.

Notes

Locate the if statement that immediately precedes the error location. Insert the keyword then after the conditional expression that follows the keyword if.

{button ,AL(`if,0`,``)} [See also](#)

{button ,PI(`",`comment`)} [Feedback](#)



Error C8074

Message

'else' : 'else' clause already encountered

Description

Two else statements were encountered in an if..endif block.

Notes

An if..endif block can include only one else branch. If you require multiple conditional branching, use an if..endif block that includes elseif statements or a switch..endswitch block instead.

{button ,AL(`if,0`,``)} [See also](#)

{button ,PI(`",`comment`)} [Feedback](#)



Error C8075

Message

'text' : 'default' label already encountered

Description

A second default case was encountered in a switch..endswitch statement.

Notes

- n A switch..endswitch statement can include only one instance of the keyword default to specify a default case. Restructure the switch....endswitch statement to eliminate one of the default cases.
- n In some versions of InstallShield, this error message is displayed when multiple case labels are encountered in a switch block.

{button ,AL(`switch',0,`,`')} [See also](#)

{button ,PI("",`comment')} [Feedback](#)



Error C8076

Message

'case' : multiple case labels for statement

Description

The compiler encountered the keyword `case` where it expected to find the statement or statements to be executed for the last defined case.

Notes

- n The case statement that immediately precedes the character or characters specified by *text* is incomplete. It may be missing a terminating semicolon.
- n In some versions of InstallShield, this error message is displayed when more than one default statement is encountered in a switch block.

{button ,AL(`switch',0,`,`')} [See also](#)

{button ,PI("",`comment')} [Feedback](#)



Error C8077

Message

'name' : label already defined

Description

The label indicated by *name* has already been used in the main program block or function.

Notes

Each label in the program block or in a function block must be unique.

{button ,AL(`goto',0,`,`')} [See also](#)

{button ,PI("",`comment')} [Feedback](#)



Error C8079

Message

'text' : invalid statement

Description

The statement at the location indicated by *text* specifies a process that is not allowed or is invalid in the current context.

Notes

- n Examine the statement at the error location. Verify that the process is valid. Examples of invalid statements are assignment statements with a constant or an arithmetic operation on the left side of the assignment operator and an end or endprogram statement directly below a label.
- n If the statement at the error location is valid, examine statement blocks (such as record declarations, control statements and function definitions) in the lines above the error location. Verify that each statement block is terminated with the correct keyword.

{button ,PI("", `comment')} [Feedback](#)



Error C8080

Message

'text' : missing arguments for function

Description

The character or characters specified by *text* were encountered after a function call where an argument list was expected.

Notes

Determine which arguments are required by the function and revise the call to include a complete argument list within parentheses.

```
{button ,AL(`Building a function;Calling a function;Declaring functions',0,`,`;`,`)}
```

[See also](#)

```
{button ,PI("",`comment')} Feedback
```



Error C8081

Message

'text' : invalid result for assignment

Description

The character or characters specified by *text* cannot be assigned the value specified by the assignment statement.

Notes

This error occurs when you attempt to assign a value to a structure. InstallShield does not allow direct assignment to a structure. You must assign a value to each member of the structure individually.

{button ,AL(`Data structures',0,`,`)} [See also](#)

{button ,PI("",`comment')} [Feedback](#)



Error C8082

Message

'text' : missing '(' after switch

Description

The character or characters specified by *text* were encountered in a switch..endswitch block at a location where a left parenthesis was expected.

Notes

Examine the switch statement in which this error occurred. The expression that follows the keyword switch must be enclosed within parentheses.

{button ,AL(`switch',0,`,`)} [See also](#)

{button ,PI("",`comment')} [Feedback](#)



Error C8083

Message

'text' : missing ')' after switch

Description

The character or characters specified by *text* were encountered in a switch..endswitch block at a location where a right parenthesis was expected.

Notes

Examine the switch statement in which this error occurred. The expression that follows the keyword switch must be enclosed within parentheses.

{button ,AL(`switch',0,`,`)} [See also](#)

{button ,PI("",`comment')} [Feedback](#)



Error C8084

Message

'text' : missing 'case' after switch

Description

The character or characters specified by *text* were encountered in a switch..endswitch block at a location where a case statement was expected.

Notes

This error will occur if the switch statement is terminated with a semicolon or is not followed immediately by a case statement.

{button ,AL(`switch',0,`,`)} [See also](#)

{button ,PI("",`comment')} [Feedback](#)



Error C8085

Message

'text' : missing '='

Description

The character or characters specified by *text* were encountered in a for..endfor block at a location where an equal sign was expected.

Notes

This error occurs when the equal sign is missing in the expression that follows the keyword for. Examine the for statement at which the error occurred and verify that the expression that follows the keyword for is structured correctly.

{button ,AL(`for,0,`,``)} [See also](#)

{button ,PI(`",`comment`)} [Feedback](#)



Error C8086

Message

'text' : missing 'to' or 'downto'

Description

The character or characters specified by *text* were encountered in a for..endfor block at a location where an equal sign was expected.

Notes

This error occurs when the keyword to or downto is missing in a for statement.

{button ,AL(`for',0,`,`')} [See also](#)
{button ,PI("",`comment')} [Feedback](#)



Error C8087

Message

cannot return value from program

Description

The main program block contains a return statement. You cannot return a value from a program.

Notes

Remove the return statement from the program block.

{button ,AL(`Program block',0,`,`')} [See also](#)

{button ,PI("",`comment')} [Feedback](#)



Error C8088

Message

'endif' : not inside if statement

Description

The keyword endif could not be matched to a previous if statement.

Notes

Examine the if..endif block at the error location. Verify that the keyword endif has a corresponding if statement. This error is often triggered after an earlier error in an if..endif block.

{button ,AL(`if,0`,``)} [See also](#)

{button ,PI(`",`comment`)} [Feedback](#)



Error C8089

Message

'endfor' : not inside for statement

Description

The keyword endfor could not be matched to a previous for statement.

Notes

Examine the for..endfor block at the error location. Verify that the keyword endfor has a corresponding for statement. This error is often triggered after an earlier error in a for..endfor block.

{button ,AL(`for',0,`,`')} [See also](#)

{button ,PI("",`comment')} [Feedback](#)



Error C8090

Message

'until' : not inside repeat statement

Description

The keyword until could not be matched to a previous repeat statement.

Notes

Examine the repeat..until block at the error location. Verify that the keyword until has a corresponding repeat statement. This error is often triggered after an earlier error in a repeat..until block.

{button ,AL(`repeat',0,`,`')} [See also](#)

{button ,PI("",`comment')} [Feedback](#)



Error C8091

Message

'endwhile' : not inside while statement

Description

The keyword endwhile could not be matched to a previous while statement.

Notes

Examine the while..endwhile block at the error location. Verify that the keyword endwhile has a corresponding while statement. This error is often triggered after an earlier error in a while..endwhile block.

{button ,AL(`while',0,`,`')} [See also](#)

{button ,PI("",`comment')} [Feedback](#)



Error C8092

Message

'name' : function called but not defined

Description

The function specified by *name* has been prototyped and called in the body of the script, but it has not been defined.

Notes

This error can occur if the function name is not spelled identically in the prototype, the function call and the function definition. It can also occur if the function definition appears in a file that should have been included (with and `#include` statement), but was not.

{button ,AL(`Building a function;Calling a function;Declaring functions',0,`,`)}}

[See also](#)

{button ,PI("`",`comment')}; [Feedback](#)



Error C8093

Message

'name' : size required for string in typedef

Description

The identifier specified by *name* is a string that has been defined as a member in a typedef block, but it does not include a size specification.

Notes

Specify the size of all STRING declarations in a structure—InstallShield's autosizing feature does not work in typedef statements.

```
{button ,AL(`typedef;String size',0,`,`)}}
```

[See also](#)

```
{button ,PI("",`comment')} Feedback
```



Error C8097

Message

'text' :syntax error

Description

The character or characters specified by *text* triggered an unclassified syntax error.

Notes

This error message is triggered for syntax errors for which there is not a specific error message.

{button ,PI("",`comment`)} [Feedback](#)



Error C8098

Message

'text' : missing '.name' after DLL name

Description

The character or characters specified by *text* were encountered after a DLL filename, where a function in the DLL file was expected.

Notes

When calling a function from a DLL, specify the function name, not the DLL filename.

{button ,AL('DLL function',0,'')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



Error C8099

Message

'text' : DLL function name expected

Description

The character or characters specified by *text* were encountered after a DLL filename, where a function in the DLL file was expected.

Notes

This error occurs when the function name specified with a DLL filename has not been declared in a prototype statement. Verify that this function has been prototyped; if it has, check the spelling of the function name.

{button ,AL(`DLL function',0,`,`')} [See also](#)

{button ,PI("",`comment')} [Feedback](#)



Error C8100

Message

'name' : function not defined for this DLL

Description

The function specified by *name* has not been prototyped for the DLL whose name precedes it.

Notes

This error occurs when you have prototyped functions from two or more DLLs and then called one of those functions with the wrong DLL name

{button ,AL('DLL function',0,'')} [See also](#)

{button ,PI("",`comment')} [Feedback](#)



Error C8101

Message

'name' : must specify DLL for this function

Description

The function specified by *name* was prototyped twice from two different DLLs.

Notes

When calling functions with identical names from two or more DLLs, you must qualify the function name by prefixing it with the DLL name followed by a period, as in the example below:

```
NUMBER nValue;
POINTER psvString;

prototype MYDLL.UsefulFunction(INT, POINTER);
prototype YOURDLL.UsefulFunction(INT, POINTER);

program
    MYDLL.UsefulFunction(nValue, psvString);
endprogram
```

{button ,AL(`DLL function',0,`,`)} [See also](#)

{button ,PI(`",`comment')} [Feedback](#)



Error C8112

Message

'name' : typedef includes instance of self

Description

The data type of the member specified by *name* is invalid. A typedef cannot contain a member that is an instance of itself.

Notes

Restructure your typedefs to avoid this error.

{button ,AL(`Data structures',0,`,`)} [See also](#)

{button ,PI("",`comment')} [Feedback](#)



Error C8113

Message

'external' : local variables cannot be external

Description

The keyword external was encountered in data declaration within a function block.

Notes

Data that is local to a function cannot be declared external.

{button ,PI("",`comment`)} [Feedback](#)



Error C8114

Message

'text' : preprocessor user define error

Description

This message is triggered by an `#error` directive.

`{button ,AL(`#error',0,`,`')}` [See also](#)

`{button ,PI("",`comment')}` [Feedback](#)



Error C8501

Message

I/O error on action file

Description

An error occurred when InstallShield attempted to access an intermediate file created during compilation.

Notes

Each time you compile your setup, InstallShield creates intermediate files with the following file extensions: .dbg, .ino, .ins and .obs. After compilation, these files remain in the scripts folder. This message is displayed when an error occurs on an attempt to access the action file; that file has the extension .obs). Among the possible causes:

- n The disk is full. Create more available disk space by deleting unwanted files; then compile again.
- n The action file is damaged. Rename it or delete it from Scripts folder; then compile again.

{button ,PI("", `comment')} [Feedback](#)



Error C8502

Message

I/O error on debug file

Description

An error occurred when InstallShield attempted to access an intermediate file created during compilation.

Notes

Each time you compile your setup, InstallShield creates intermediate files with the following file extensions: .dbg, .ino, .ins and .obs. After compilation, these files remain in the scripts folder. This message is displayed when an error occurs on an attempt to access the debug file; that file has the extension .dbg). Among the possible causes:

- n The disk is full. Create more available disk space by deleting unwanted files; then compile again.
- n The debug file is damaged. Rename it or delete it from Scripts folder; then compile again.

{button ,PI("", `comment')} [Feedback](#)



Error C8503

Message

Can't open script input file

Description

The command line compiler was unable to open the script file specified for compile.

Notes

Verify the location and name of the file. You may have misspelled the filename or specified an incorrect path.

{button ,PI("",`comment')} [Feedback](#)



Error C8504

Message

Can't open .ins output file

Description

The command line compiler was unable to create the file for the compiled script.

Notes

The following problems will cause of this error:

- n The disk is full. Create more available disk space by deleting unwanted files; then compile again.
- n There are insufficient system resources (file handles and/or memory). Close any other applications that are open; then compile again.

{button ,PI("", `comment`)} [Feedback](#)



Error C8505

Message

Can't open .dbg debug file

Description

The command line compiler was unable to create the debug file. This file contains the information required to run the script with the debugger.

Notes

The following problems will cause of this error:

- n The disk is full. Create more available disk space by deleting unwanted files; then compile again.
- n There are insufficient system resources (file handles and/or memory). Close any other applications that are open; then compile again.

{button ,PI("",`comment')} [Feedback](#)



Error C8506

Message

Out of memory

Description

The compiler was unable to allocate sufficient memory to complete the compilation.

Notes

Close any other applications that are open; then compile again. If the problem persists, restart Windows; then load InstallShield and compile again.

{button ,PI("",`comment')} [Feedback](#)



Error C8508

Message

too many macro expansions in line

Description

The specified line contains too many macros; the maximum number of macros per line is 100.

Notes

Reduce the number of macros on the line by breaking the line into two or more lines.

{button ,PI("",`comment')} [Feedback](#)



Error C8509

Message

invalid delimiter for include file name

Description

The filename specified after the preprocessor directive `#include` is missing the initial quotation mark or angled bracket (`<`).

Notes

Filenames specified with the preprocessor directive `#include` must be enclosed by quotation marks (`"filename"`) or angled brackets (`<filename>`). Insert a quotation mark or opening angled bracket before the file name. Note that if you insert an initial quotation mark, the filename must be followed by a closing quotation mark; if you insert an opening angled bracket, the filename must be followed by a closing angled bracket (`>`).

`{button ,AL(`#include',0,`,`')}` [See also](#)

`{button ,PI(`",`comment')}` [Feedback](#)



Error C8510

Message

missing delimiter for include file name

Description

The filename specified after the preprocessor directive `#include` is missing the closing quotation mark or angled bracket (`<`).

Notes

Filenames specified with the preprocessor directive `#include` must be enclosed by quotation marks (`"filename"`) or angled brackets (`<filename>`). Insert a quotation mark or closing angled bracket after the filename. Note that if the filename has an opening quotation mark, you must insert a quotation mark after the filename; if the filename has an opening angled bracket, you must insert a closing angled bracket (`>`).

`{button ,AL('#include',0,';')}` [See also](#)

`{button ,PI("" , `comment')}` [Feedback](#)



Error C8511

Message

can't open include file

Description

The compiler was unable to find or open a file that has been included in the script.

Notes

The following problems will cause of this error:

- n The filename in an `#include` statement is misspelled.
- n A filename in an `#include` statement is specified without a full path, but the file can not be found in the source directory.
- n You don't have rights to open files on the network drive where the file resides.
- n The file is damaged.

`{button ,AL(`#include',0,`,`')}` [See also](#)

`{button ,PI("",`comment')}` [Feedback](#)



Error C8512

Message

include file plus path is too large

Description

The path specified in an #include statement exceeds the maximum length of 256 characters

Notes

Move the included files to a folder that can be referenced with a path whose length is less than 256 characters.

{button ,AL(`#include',0,`,`')} [See also](#)

{button ,PI("",`comment')} [Feedback](#)



Error C8513

Message

too many nested #include files

Description

InstallShield allows nesting of include files up to a maximum of 8 levels. This limit has been exceeded.

Notes

Reduce the number of nested levels.

{button ,AL(^#include',0,','')} [See also](#)

{button ,PI("",`comment')} [Feedback](#)



Error C8514

Message

#if statements too deeply nested

Description

InstallShield allows nesting of #if statements up to a maximum of 10 levels. This limit has been exceeded.

Notes

Reduce the number of nested levels.

{button ,AL(`#if. . #else. . #endif,0,`,`)} [See also](#)

{button ,PI("",`comment`)} [Feedback](#)



Error C8515

Message

macro expansion buffer overflow

Description

The expansion of macros in the error line exceeds the maximum of 1024 characters.

Notes

Restructure your code to eliminate this error.

`{button ,AL(`#include',0,`,`')}` [See also](#)

`{button ,PI("",`comment')}` [Feedback](#)



Error C8516

Message

maximum error count reached

Description

The maximum number of errors specified in the Compile tab of the Setup Setting dialog box has been reached.

Notes

For more information about the Compile tab of the Setup Setting dialog box, [click here](#).

{button ,PI("",`comment')} [Feedback](#)



Error C8517

Message

too many include files

Description

The maximum number of include was exceeded.

Notes

A setup script can include no more than 100 source files. Combine two or more source files to bring the total number down to 100.

{button ,AL(`#include',0,`,`')} [See also](#)

{button ,PI("",`comment')} [Feedback](#)



Error C8518

Message

I/O error on link file

Description

An error occurred when InstallShield attempted to write to the link file.

Notes

Each time you compile your setup, InstallShield creates intermediate files with the following file extensions: .dbg, .ino, .ins and .obs. After compilation, these files remain in the scripts folder. This message is displayed when an error occurs on an attempt to access the link file; that file has the extension .ino). Among the possible causes:

- n The disk is full. Create more available disk space by deleting unwanted files; then compile again.
- n The link file is damaged. Rename it or delete it from Scripts folder; then compile again.

{button ,PI("", `comment')} [Feedback](#)



Error C8519

Message

user define fatal error

Description

This message is triggered by an `#error` directive.

`{button ,AL(`#error',0,`,`)}` [See also](#)

`{button ,PI("",`comment')}` [Feedback](#)



Error C9001

Message

'text' : internal error

Description

The character or characters specified by *text* triggered an unclassified syntax error.

Notes

This error message is triggered by syntax errors for which there is not a specific error message.



Reserved words

Reserved words and characters have special meaning in InstallScript and cannot be used except for their intended purposes. InstallScript has four classes of reserved words:

[Built-in function names](#)

[Language keywords](#)

[Predefined constants](#)

[System variables](#)

{button ,PI("",`comment')} [Feedback](#)



Language keywords

Language keywords are words InstallShield uses as commands in the script. Language keywords are interpreted by the InstallShield Script Compiler to perform some action, or are considered part of a statement. You cannot use the following keywords for any reason other than their predefined purpose:

<u>:(semicolon)</u>	<u>elseif</u>	<u>HWND</u>	<u>return</u>
<u>abort</u>	<u>endfor</u>	<u>if</u>	<u>SHORT</u>
<u>begin</u>	<u>endif</u>	<u>INT</u>	<u>step</u>
<u>BOOL</u>	<u>endprogram</u>	<u>KERNEL</u>	<u>STRING</u>
<u>BYREF</u>	<u>endswitch</u>	<u>LIST</u>	<u>switch</u>
<u>case</u>	<u>endwhile</u>	<u>LONG</u>	<u>then</u>
<u>CHAR</u>	<u>exit</u>	<u>NUMBER</u>	<u>to</u>
<u>default</u>	<u>for</u>	<u>POINTER</u>	<u>typedef</u>
<u>downto</u>	<u>function</u>	<u>program</u>	<u>until</u>
<u>else</u>	<u>GDI</u>	<u>prototype</u>	<u>USER</u>
<u>end</u>	<u>goto</u>	<u>repeat</u>	<u>while</u>

{button ,PI("",`comment')} [Feedback](#)



Dialog styles

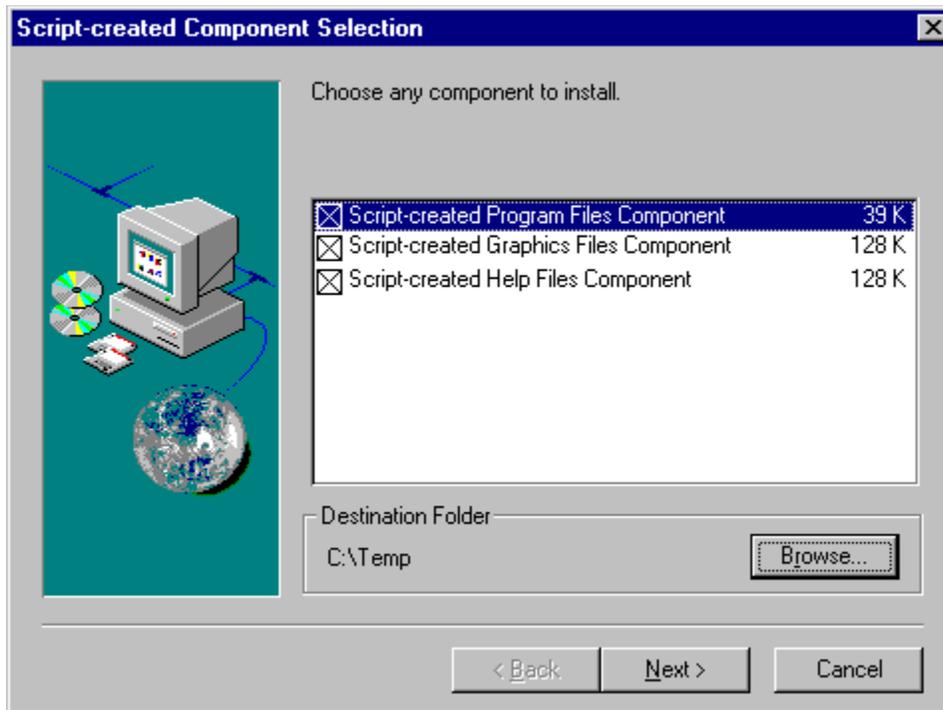
Four different dialog box styles are available:

<code>{button ,JI('LangRef.hlp>Dialog',`CHECKBOX_dialog_box_style')}</code>	<u>CHECKBOX</u>
<code>{button ,JI('LangRef.hlp>Dialog',`CHECKBOX95_dialog_box_style')}</code>	<u>CHECKBOX95</u>
<code>{button ,JI('LangRef.hlp>Dialog',`CHECKMARK_dialog_box_style')}</code>	<u>CHECKMARK</u>
<code>{button ,JI('LangRef.hlp>Dialog',`CHECKLINE_dialog_box_style')}</code>	<u>CHECKLINE</u>

`{button ,PI("",`comment')}` Feedback



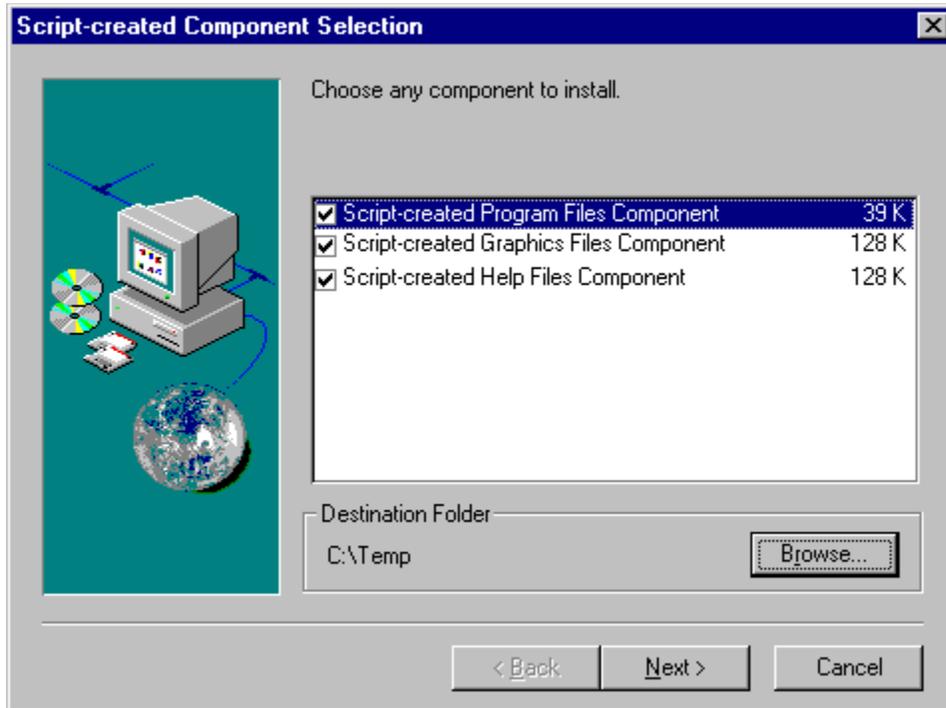
CHECKBOX dialog box style



{button ,PI("",`comment')} [Feedback](#)



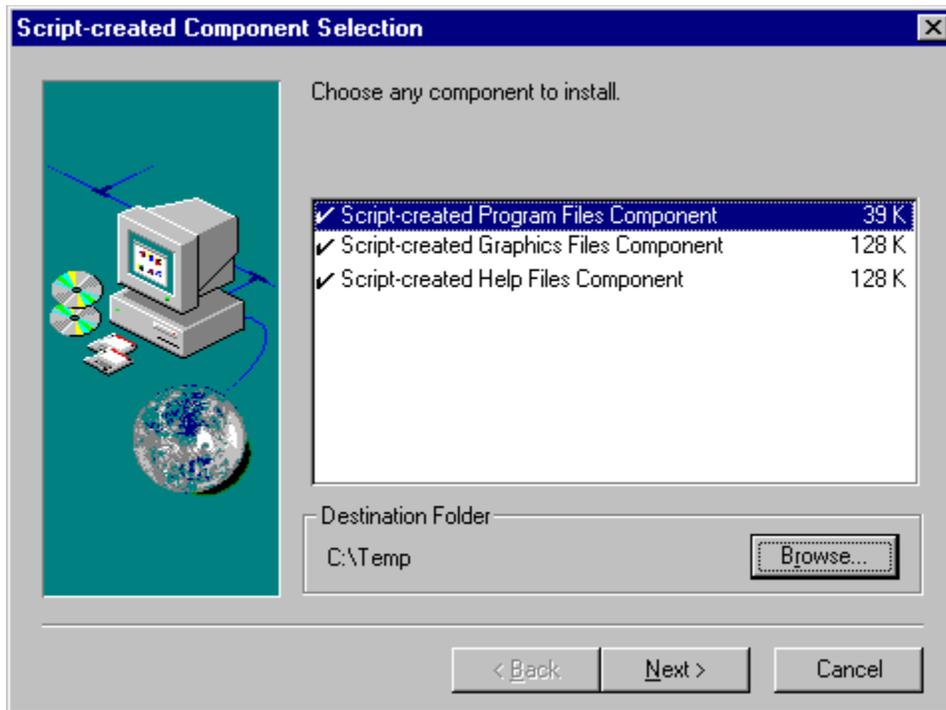
CHECKBOX95 dialog box style



{button ,PI("",`comment')} [Feedback](#)



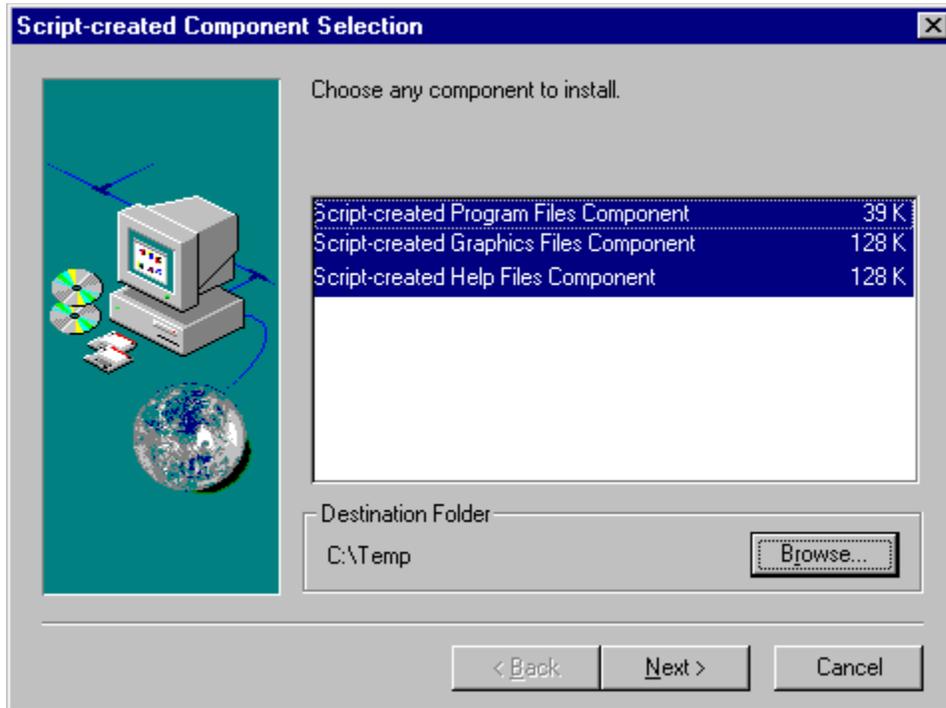
CHECKMARK dialog box style



{button ,PI("",`comment')} [Feedback](#)



CHECKLINE dialog box style



{button ,PI("",`comment')} [Feedback](#)

Error messages

The InstallScript compiler displays error messages in the Output window. This window, which is normally positioned near the bottom of the main window, has a tab with the label Compile. If the Output window is not present, select View from the InstallShield menu; then select Output from the view menu.

The following types of error messages are displayed to identify errors in your setup script:

<u>Warnings</u>	C7501 through C7503
<u>Syntax errors</u>	C8001 through C8114
<u>Fatal errors</u>	C8501 through C8519
<u>Internal errors</u>	C9001

{button ,PI("",`comment')} Feedback



Warnings

The compiler issues warning messages to alert you to irregularities in the script that may result in inefficiencies or run-time errors. Normally, warnings do not prevent script compilation; however, compilation will be interrupted if "Warnings as errors" is checked in the Setup Settings Compile tab or if the number of warnings exceeds the maximum specified in the Setup Settings Compile tab. The default maximum value is 50. InstallScript implements three warning levels. For more information, [click here](#).

C7501 macro redefinition

C7502 string/array size exceeds recommended limit

C7503 function defined but never called

{button ,PI("",`comment')} [Feedback](#)



Syntax errors

Syntax errors prevent the file Setup.ins from being created and cause the compiler to display one of the error messages in the list below. Note that a single syntax error can cause the compiler to generate more than one error messages. The first error message is triggered by the actual syntax error. The others because the compiler is not able to resolve the statement or statements that follow the error. If your script generates more than one syntax error, correct the first one in the list; then recompile. By correcting the first error, you may eliminate one or more of the others.

- C8001 multiple main programs defined
- C8002 function name expected
- C8003 function has no prototype declaration
- C8004 identifier already declared
- C8005 function was declared as DLL function
- C8006 missing '(' after function name
- C8007 comma expected
- C8008 identifier expected
- C8009 too many parameters
- C8010 missing right parenthesis
- C8011 missing 'begin' at start of function
- C8012 semicolon expected
- C8013 unexpected end of source file
- C8014 identifier already defined
- C8015 member name already defined
- C8016 undefined label:
- C8017 expected typedef (struct) name
- C8019 expected type declaration
- C8020 parameter list expected
- C8021 missing 'begin' after 'typedef'
- C8022 comma or semicolon expected
- C8023 right bracket expected
- C8024 invalid array/string size
- C8025 undefined identifier
- C8026 invalid use of identifier
- C8027 missing operand
- C8028 unresolved operator
- C8031 uncalled function
- C8032 missing member reference
- C8033 unsubscripted array
- C8034 missing expression
- C8035 statement label required
- C8036 too many arguments
- C8037 variable required
- C8038 numeric value required

C8039 string value required
C8040 incomplete argument list
C8042 string or array type required
C8043 typedef pointer type required
C8044 typedef type required
C8045 member name not found
C8046 numeric variable required
C8047 can only take address of variable
C8048 macro name missing
C8049 missing expression for #if/elif
C8050 invalid expression for #if/elif
C8051 missing end of string literal
C8052 macro expansion text too large
C8053 identifier not a #define macro
C8054 include file name missing
C8055 #elif not preceded by #if
C8057 #else not preceded by #if
C8058 #endif not preceded by #if
C8059 unrecognized preprocessor command
C8062 constant operand required
C8063 input line too long
C8064 unterminated comment
C8065 string literal exceeds 255 characters
C8066 missing #endif statement at end of file
C8067 integer constant too large
C8068 unrecognized character encountered
C8069 preprocessor command must be first on line
C8070 string constant expected
C8071 colon expected
C8072 'elseif' cannot follow 'else'
C8073 missing 'then' keyword
C8074 'else' clause already encountered
C8075 'default' label already encountered
C8076 multiple case labels for statement
C8077 label already defined
C8079 invalid statement
C8080 missing arguments for function
C8081 invalid result for assignment
C8082 missing '(' after switch
C8083 missing ')' after switch
C8084 missing 'case' after switch
C8085 missing '='
C8086 missing 'to' or 'downto'

C8087 cannot return value from program
C8088 not inside if statement
C8089 not inside for statement
C8090 not inside repeat statement
C8091 not inside while statement
C8092 function called but not defined
C8093 size required for string in typedef
C8097 syntax error
C8098 missing '.name' after DLL name
C8099 DLL function name expected
C8100 function not defined for this DLL
C8101 must specify DLL for this function
C8112 typedef includes instance of self
C8113 local variables cannot be external
C8114 preprocessor user define error

{button ,PI("",`comment`)} [Feedback](#)



Fatal errors

Fatal errors are those that prevent the compilation from continuing. They are caused by missing include files, incorrectly specified filenames, disk I/O errors, and insufficient system resources; the compiler also has certain built-in limitations, such as the limit on include files, that will trigger a fatal error if exceeded.

- C8501 I/O error on action file
- C8502 I/O error on debug file
- C8503 Can't open script input file
- C8504 Can't open .ins output file
- C8505 Can't open .dbg debug file
- C8506 out of memory
- C8508 too many macro expansions in line
- C8509 invalid delimiter for include file name
- C8510 missing delimiter for include file name
- C8511 can't open include file
- C8512 include file plus path is too large
- C8513 too many nested #include files
- C8514 #if statements too deeply nested
- C8515 macro expansion buffer overflow
- C8516 maximum error count reached
- C8517 too many include files
- C8518 I/O error on link file
- C8519 user define fatal error

{button ,PI("",`comment')} [Feedback](#)



Internal errors

C9001 internal error

{button ,PI("", `comment`)} Feedback



Constant data

A constant is a named data item with a defined value. InstallShield supports two types of constants:

- n [Predefined constants](#), such as TRUE and RESET, are part of InstallScript. These constants, which are used as function parameters and return values for built-in functions, cannot be redefined in the script. Attempting to redefine a predefined constant will result in a compiler error.
- n User-defined constants are declared by the programmer as needed for individual scripts. Although a user-defined constant can be redefined after the initial declaration, it is generally not considered good programming practice to do so.

User-defined constants are declared by with a #define preprocessor statement. (InstallScript does not support the const keyword for declaring variable constants as the C++ language does.) String constants must be enclosed within quotation marks; numeric constants are defined without quotation marks and contain only numeric characters. Once declared, a string constant can be used anywhere that a string literal can be used. Likewise, number constants can be used anywhere that a numeric literal can be used.

In the following example, a string constant and a numeric constant are declared:

```
#define COMPANY_NAME    "Example_Company"  
#define MAXCOUNT      1000
```

A constant name must follow the rules for InstallScript identifiers. By convention, constant identifiers are created with all upper case characters. InstallScript's predefined constants follow that convention.



When creating setups that require multi-language support, use the InstallShield IDE to specify user-defined string constants in string tables. Then, in your script, access those string constants with the @ operator. For more information about string tables, click [here](#).

{button ,AL('Predefined constants;#define',0,'')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



Data structures

A data structure is a named data item that consists of logically-related variables, called members. In many programming languages, data structures are called records, and the variables within records are called fields. In InstallScript, data structures are similar in form and function to structures in C. They can include members of different data types, and those members within the data structure can be referenced directly by using the member operator.

To define a data structure, use the keyword `typedef` and follow it with the name of the data structure. Fields in the structure must be defined within a `begin..end` block, as shown in the example below, which defines a structure called `EMPLOYEE`. The data structure `EMPLOYEE` includes three variables: a string variable for an employee's name, a string variable for an employee's department, and a number variable for an employee's phone extension.

```
typedef EMPLOYEE
begin
    STRING szName[50];
    STRING szDepartment[50];
    NUMBER nExtension;
end;
```

When you define a data structure, you are actually defining a new data type. To use a data structure within a program, you must first declare a variable of that type. To do that, use the name of the defined data structure as the data type and follow it with an identifier, as shown in the example below, which creates a variable of type `EMPLOYEE`.

```
EMPLOYEE structEmployee;
```

To reference members of the structure variable, use the member operator (`.`). In the example below, literal values are assigned to each member of `structEmployee`.

```
program
    structEmployee.szName = "I. S. Coder;
    structEmployee.szDepartment = "Development";
    structEmployee.nExtension = 555;
endprogram
```



The following restrictions apply to structures:

- n You cannot assign the contents of one structure to another structure with the assignment operator, as in `newstruct = struct1`.
- n You must specify the size of all `STRING` declarations in a structure—InstallShield's autosizing feature does not work in `typedef` statements.
- n You cannot declare a structure within a function.
- n You cannot use the `BYREF` operator with a structure, nor can you pass a structure member in a parameter that was declared with the `BYREF` operator. To modify a member of a user-defined structure in a user-defined function, pass a pointer to the structure and then use the structure pointer operator (`->`) to access the data within the function.

Like C, InstallScript allows you to nest or embed data structures. For example, suppose you wanted to create a structure that could be used to define the upper left and lower right coordinates of a rectangle. Each coordinate consists of two values: an `x` value and a `y` value. You could define a structure that consists of four members: the `x` and `y` positions of the upper left corner and the `x` and `y` positions of the lower right corner.

However, since each `x` and `y` pair is a logical unit, you might first define a structure called `POINT` that has two members defining a vertical and horizontal position. Then you could define a structure called `RECT` that includes two members of type `POINT`, one to define the upper left coordinate, another to define the lower right coordinate. These two structures are shown below:

```
// Define a point structure.
typedef POINT
```

```

begin
    SHORT nX;
    SHORT nY;
end;

// Use nested point structures to define a rectangle structure.
typedef RECT
begin
    POINT UpperLeft;
    POINT LowerRight;
end;

```

When a structure is to be referenced by a pointer to that structure, you must use the structure pointer operator (->) to address members of the structure. In the example below, a variable of type RECT is declared, a pointer to the structure is declared, and then the address of the RECT variable is assigned to the pointer. Finally, the structure pointer operator is used to initialize each member to 0.

```

    RECT Rectangle;
    RECT POINTER pRect;

program
    pRect = &Rectangle;
    pRect->UpperLeft.nX = 0;
    pRect->UpperLeft.nY = 0;
    pRect->LowerRight.nX = 0;
    pRect->LowerRight.nY = 0;

    // . . .

endprogram

```

The following script presents a more complete demonstration of structure pointers, nested structures, and structure pointer dereferencing with the structure pointer operator.

```

// Use a structure to define a point.
typedef POINT
begin
    SHORT nX;
    SHORT nY;
end;

// Use nested structures to define a rectangle.
typedef RECT
begin
    POINT UpperLeft;
    POINT LowerRight;
end;

// Declare a rectangle structure variable.
RECT Rectangle;

// Define a pointer to a RECT structure.
RECT POINTER pRect;

// Declare a function to display structure contents.
prototype ShiftRectBy2(RECT POINTER);

program
    // Get a pointer to the Rectangle structure.
    pRect = &Rectangle;

    // Define the points that define the rectangle.
    pRect->UpperLeft.nX = 100;
    pRect->UpperLeft.nY = 400;
    pRect->LowerRight.nX = 200;
    pRect->LowerRight.nY = 100;

```

```

// Display point x and y values before calling ShiftRectBy2.
PrintfBox(INFORMATION,
    "BEFORE calling ShiftRectBy2",
    "pRect->UpperLeft.nX = %d\n" +
    "pRect->UpperLeft.nY = %d\n" +
    "pRect->LowerRight.nX = %d\n" +
    "pRect->LowerRight.nY = %d\n",
    pRect->UpperLeft.nX,
    pRect->UpperLeft.nY,
    pRect->LowerRight.nX,
    pRect->LowerRight.nY
);

// Shift the rectangle 2 up and 2 to the right.
ShiftRectBy2(pRect);

// Display point x and y values after calling ShiftRectBy2.
PrintfBox(INFORMATION,
    "AFTER calling ShiftRectBy2",
    "pRect->UpperLeft.nX = %d\n" +
    "pRect->UpperLeft.nY = %d\n" +
    "pRect->LowerRight.nX = %d\n" +
    "pRect->LowerRight.nY = %d\n",
    pRect->UpperLeft.nX,
    pRect->UpperLeft.nY,
    pRect->LowerRight.nX,
    pRect->LowerRight.nY
);

endprogram

// Define the rectangle shifting function.
function ShiftRectBy2(pR)
begin
    pR->UpperLeft.nX = pR->UpperLeft.nX + 2;
    pR->UpperLeft.nY = pR->UpperLeft.nY + 2;
    pR->LowerRight.nX = pR->LowerRight.nX + 2;
    pR->LowerRight.nY = pR->LowerRight.nY + 2;
end;

```

{button ,AL(`Address operator;BYREF_operator;Indirection operator;Member operator;Structure pointer operator',0,`,`)} [See also](#)

{button ,PI("`comment')} [Feedback](#)



Data types

InstallShield supports the following data types:

BOOL	Boolean data: either TRUE (1) or FALSE (0). Variables of this type should not be used to store any other values. Like C++, InstallScript evaluates non-zero values as TRUE; only the value of zero is evaluated as FALSE. Normally, the value of one is used to indicate TRUE.
CHAR	<p>Character data: a single 8-bit signed character. When a literal character appears in a script, it must be enclosed within single or double quotation marks. Note that you can assign a numeric ASCII value to a character value. To display a CHAR variable as a character, use the <u>format specifier</u> "%c" with the function <u>SprintfBox</u>. To display the numeric value of a CHAR variable use the specifier "%d".</p> <p>InstallShield character variable types are signed; therefore extended ASCII characters will be interpreted as negative numbers when interpreted numerically. To avoid this problem, assign the value to a number variable; then AND (&) the number variable with the value 255 before interpreting the number.</p>
HWND	Handle to a window. The HWND variable type also can be used to store any other type of handle valid in Windows. HWND variables are normally initialized using the <u>CmdGetHwndDlg</u> or <u>GetWindowHandle</u> functions. Internally, HWND variables are equivalent to the data type NUMBER.
INT	Equivalent to the NUMBER type; provided for convenience. When a variable of this type is specified in a function prototype or in a data structure definition, InstallShield automatically converts its value to a 2-byte value when necessary in 16-bit setups to ensure compatibility with external (DLL) functions.
LIST	Pointer to an InstallShield list. LIST variables are always initialized and uninitialized using the <u>ListCreate</u> and <u>ListDestroy</u> functions. Internally, LIST variables are equivalent to the data type NUMBER.
LONG	Equivalent to the NUMBER type; provided for convenience.
LPSTR	Equivalent to the POINTER type, provided for convenience.
NUMBER	<p>Signed four-byte integer. NUMBER is the recommended data type for storing numeric data. The data type is similar to the LONG variable type of other programming languages. It can hold any value between -2147483648 and +2147483647. Note that all numeric data types in InstallShield are equivalent to the NUMBER variable type.</p> <p>When a variable of this type is specified in a function prototype or in a data structure definition, InstallShield automatically converts its value to a 2-byte value when necessary in 16-bit setups to ensure compatibility with external (DLL) functions.</p>
POINTER	A pointer to data. POINTER variables are normally initialized by using the address of (&) operator to assign the address of a variable to the pointer variable.
SHORT	Equivalent to the NUMBER type; provided for convenience. When a variable of this type is specified in a function prototype or in a data structure definition, InstallShield automatically converts its value to a 2-byte value when necessary in 16-bit

setups to ensure compatibility with external (DLL) functions.

STRING

An array of characters. STRING variables, which are similar to arrays of characters in the C++ language, are NULL terminated. However, InstallShield does not support multiple NULL terminated strings in the same string variable.

InstallShield provides automatic sizing for string variable types. You can also set a size explicitly when declaring a string variable. 16-bit setups can store strings up to 512 characters in length. 32-bit setups can store strings up to 1024 characters in length.

To display a STRING variable, use the [SprintfBox](#) function with the format specifier "%s", or use the [MessageBox](#) function.



InstallScript does not provide unsigned or floating data types.

{button ,AL(`Data Types;Data structures;Hungarian notation;Pointers;String indexing;String size',0,`,`)} [See](#)
[also](#)

{button ,PI(`",`comment')} [Feedback](#)



Escape sequences

An escape sequence is a set of characters used to insert into a string certain special characters, such as tabs, carriage returns and quotation marks. Escape sequences in InstallScript are very much like those in C. They begin with a backslash, called an escape character; the backslash is followed by one or more characters that have special meaning. If the backslash is followed by characters other than those used in an escape sequence, the backslash is ignored. To print a backslash, you must use the escape sequence `\\`.

Perhaps the most commonly used escape sequence is `\n`, which inserts a newline character into a string. The string "This is line one, This is line two." will be displayed or printed on a single line. However, the string "This is line one, \nThis is line two." will be displayed or printed as shown below:

```
This is line one,  
This is line two.
```



The percent sign (%) also has a special function in InstallScript; it is used as the first character of a format specifier, which is a sequence of characters that is used with functions such as Sprintf and SprintfBox to indicate how the value stored in a variable should be displayed on screen.

The following table lists the escape sequences that are supported by InstallScript:

Escape Sequence	Performs the Following Action
<code>\n</code>	Inserts a carriage return and a line feed.
<code>\'</code>	Inserts a single quotation mark in the string.
<code>\"</code>	Inserts a double quotation mark in the string.
<code>\r</code>	Inserts a carriage return only. Does not insert a line feed.
<code>\t</code>	Inserts a tab character.
<code>\ooo</code>	Indicates an ASCII character—not an integer—in octal notation.
<code>\xhh</code>	Indicates an ASCII character—not an integer—in hexadecimal notation.
<code>\\</code>	Inserts a backslash.



To specify a UNC (Universal Naming Convention) path in an InstallScript string, you must use *two* backslash escape sequences (that is, four backslashes—`\\`), to create the double backslash at the start of the path name. For example, the path name `\ISServer\Public\Readme.txt` must be specified as follows:

```
"\\\\ISServer\\Public\\Readme.txt"
```

{button ,PI("",`comment')} Feedback



Format specifiers

Format specifiers are used with the functions `Sprintf` and `SprintfBox` to control the display of values that are stored in variables. A format specifier begins with a percent sign (%) and is followed by at least one or two characters. Format specifications follow the format shown below:

```
% [-] [#] [0] [width] [.precision] type
```

Each field of a format specification is a single character or number that represents a particular format option. The type field, for example, determines whether `Sprintf` or `SprintfBox` interprets the associated argument as a character, a string, or a number. The initial character % and the type field are both required. Items enclosed within brackets are optional. The simplest format specification contains only the percent sign and a type character, for example %s.

In the following example, the value of `svString` is displayed in a message box. The format specifier %s, which is assigned to `svFormat`, indicates to `SprintfBox` that the value of `svString` should be displayed as a string of characters.

```
STRING szTitle, szFormat, szString;
program
    szTitle = "Demonstrate format specifiers";
    szFormat = "%s";
    szString = "This is a string.";
    SprintfBox(INFORMATION, szTitle, szFormat, szString);
endprogram
```

The value assigned to `svFormat` may contain literal characters (including [escape sequences](#)) that are to be displayed along with the value of a variable. In the following example, an identifying label is displayed to the left of a number variable: "nNumber = 100."

```
STRING szTitle, szFormat;
NUMBER nNumber;
program
    szTitle = "Demonstrate format specifiers";
    szFormat = "nNumber = %d.";
    nNumber = 100;
    SprintfBox(INFORMATION, szTitle, szFormat, nNumber);
endprogram
```



To print a percent sign, you must insert two percent signs in the string assigned to `svFormat`. Assuming that the number to be printed is 100, the following format specification string displays "nNumber = 100%."

```
svFormat = "nNumber = %d%%."
```

The table below describes each of the fields in a format specifier.

Field	Meaning
-	If you include a hyphen after the percent character, the output value is aligned left and padded on the right to the width of the field with blanks or zeros. If you omit this field, the output value is right aligned and padded on the left.
#	Use this symbol to prefix hexadecimal values with 0x (lowercase) or 0X (uppercase).
0	Pads the output value with zeros to fill the field width. If you omit this field, the output value is padded with blanks.
width	Enter the minimum number of characters you want to place in this field. Type the width field as a non-negative integer. When you enter a width specification, the value is never truncated. If the number of characters in the output value is greater than the width specified, or if you omit the width field, every character of the value is displayed, subject to the value of the precision field.
precision	Enter the minimum number of digits you want in this field. If the number of digits in the argument is less than the precision value you enter, the output value on the left is padded with zeros. When the number of digits exceeds the precision value, the value is not truncated. If you enter a precision value of zero or omit it entirely, or if the period (.) appears without a number following it, the precision is set to 1. For strings, convert the maximum number of characters.
type	Format the corresponding argument as a character, a string, or a number. When two format specifier

letter combinations are shown, you can use one or the other, but not both at the same time. This is a required field. In this field you must enter one of the following characters:

- c Formats a single character of type CHAR. The Sprintf function ignores a character with a numeric value of zero.
- d, i Formats a single integer of type INT or of type NUMBER.
- ld, li Formats a single signed decimal integer of type LONG.
- lx, lX Formats a single unsigned hexadecimal integer of type LONG.
- s Formats a string (type STRING).

Each format specifier has a matching variable. The variables are listed from left to right after the string, with the first variable matching the first format specifier in the string, the second variable matching the second format specifier in the string, and so on. At run time, InstallShield inserts each variable's contents into the string at the location of its matching format specifier.

{button ,AL(`Escape sequences;Sprintf;SprintfBox',0,'')} [See also](#)
{button ,PI("",`comment')} [Feedback](#)



Global vs. local variables

Variables may be either global or local. A variable is global if it is declared outside of the main program block and not within a function. A variable is local if it is declared between the function declaration and the keyword begin within that function. Global variables are visible and available to all statements in a setup script that follow its declaration. Local variables are visible and available only within the function where they are declared.



InstallScript system variables are global and therefore are visible to the main program and to all functions in a script.

In the following example, the variable `nVisibleEverywhere` can be referenced by any statement in the script. The variable `nVisibleOnlyToFunctions` may be referenced only by the functions. The variable `nVisibleOnlyToSecondFunction` cannot be referenced by the main program or by `FirstFunction`. The variable `szString` is local to `FirstFunction`.

```
prototype FirstFunction();
prototype SecondFunction();

NUMBER nVisibleEverywhere;

program
    nVisibleEverywhere = 10;
    FirstFunction();
    SecondFunction();
endprogram

NUMBER nVisibleOnlyToFunctions;

function FirstFunction()
    STRING szString;
begin
    szString = "Local to FirstFunction";
    nVisibleOnlyToFunctions = 20;
end;

NUMBER nVisibleOnlyToSecondFunction;

function SecondFunction()
begin
    nVisibleOnlyToSecondFunction = 30;
end;
```

Although identifiers in a script must be unique, it is valid for a local variable to have a name identical to that of a global variable, or for one function to declare a local variable that has the same name as a local variable declared in another function. These exceptions are allowed because InstallShield qualifies local variable names based on the function with which they are associated. In the example below, the global variable `szVal` is not affected by the action of `AFunction`, which has a local variable of the same name; the function `MessageBox` displays the string "YES", which was the value assigned to the global variable `szVal`.

```
STRING szVal;
prototype AFunction();

program
    szVal = "YES";
    AFunction();
    MessageBox(szVal, INFORMATION);
endprogram

function AFunction()
    STRING szVal;
begin
    szVal = "NO";
end;
```

Parameter names in function definitions are considered to be local variables. If a global variable is passed to a function whose parameter has the same name as the global variable, the value of that global variable will not be changed (unless the parameter was specified with the [BYREF operator](#) in the function prototype). In the following example, AFunction has no effect on the global variable szVal; the script displays the string "YES".

```
STRING szVal;
prototype AFunction(STRING);

program
  szVal = "YES";
  AFunction(szVal);
  MessageBox(szVal, INFORMATION);
endprogram

function AFunction(szVal)
begin
  szVal = "NO";
end;
```

{button ,AL(`Variable data;BYREF operator',0,`,`)} [See also](#)

{button ,PI(`",`,`comment')} [Feedback](#)



Hungarian notation

InstallShield help topic, script templates, and Project Wizard-generated scripts employ an extended form of Hungarian notation, a naming convention that uses short, lowercase prefixes to indicate data type. For example, `iPointSize` denotes an integer variable; `szFileName` indicates a string variable.

In this Language reference, Hungarian notation is used in example scripts to indicate the data type of all variables. In function syntax descriptions, Hungarian notation is used for parameter names to indicate the type of data that may be passed in a parameter. For example, the syntax description of `BatchDeleteEx` shows that it takes two parameters:

```
BatchDeleteEx (szKey, nOptions);
```

The first, identified as `szKey`, may be a string variable or constant; the second, identified as `nOptions`, may be a number variable or constant.

In those cases where a variable parameter is required, the Language reference employs a special set of two-letter prefixes: the first letter indicates the data type; the second character is the letter `v`, for variable. In the syntax description for `GetDir`, the first parameter can be a string variable or constant, but the second parameter must be a variable.

```
GetDir (szPath, svDir);
```

Functions that require variable parameters generally return data to the caller in those parameters. Note that these special prefixes are not recommended for use in InstallScript scripts.

Because Hungarian notation makes it easy to recognize a variable's type, it is strongly recommended that you use Hungarian notation when you create variable names in your own scripts. The table below describes each of the prefixes used in InstallShield.

Prefix	Data type	When used in function syntax (InstallScript Language Reference)
b	Boolean (BOOL)	Boolean constant, literal, or variable.
bv	Boolean (BOOL)	Boolean variable only. Constants and literals not allowed.
c	Character (CHAR)	Character constant, literal, or variable.
const	Constant	Constant or literal. Variables not allowed.
h	Handle (HWND)	Handle variable.
i	Integer (INT)	Integer constant, literal, or variable.
l	Long integer (LONG)	Long integer constant, literal, or variable.
lv	Long integer (LONG)	Long integer variable only. Constants and literals not allowed.
list	List (LIST)	List variable.
n	Number (NUMBER)	Number constant, literal, or variable.
nv	Number (NUMBER)	Number variable only. Constants and literals not allowed.
p	Pointer (POINTER)	Pointer variable.
pstruct	Pointer to a defined structure type	Not used.
s	Short integer (SHORT)	Short integer constant, literal, or variable.
sz	String (STRING)	String constant, literal, or variable.
sv	String (STRING)	String variable only. Constants and literals not allowed.
struct	Defined structure type	Not used.

{button ,AL(`Data Types;Data structures;Pointers;String indexing;String size',0,' ')}
{button ,PI("",`comment')} Feedback

See also



Pointers

A pointer is a variable that contains the address of another variable. To declare a pointer, use the keyword `POINTER` followed by a variable name, as shown below:

```
POINTER pPointerName;
```

To declare a pointer that will be used to access the members of a data structure, precede the keyword `POINTER` with the structure type:

```
typedef RECT
begin
    SHORT sX;
    SHORT sY;
end;

RECT Rectangle;
RECT POINTER pRect;
```

Use the address operator (`&`) to assign the address of a variable to a pointer variable:

```
pPointerName = &MyStructure;
pNum = &nvNumber;
pString = &svString[0];
```

When defining a function that takes a pointer to a structure as a parameter, use the structure name with `POINTER` in the function prototype, as shown below. Note that any function prototype that specifies a pointer to a structure as one of its parameters must be declared after the structure declaration.

```
typedef RECT
begin
    SHORT sX;
    SHORT sY;
end;

RECT Rectangle;
RECT POINTER pRect;

prototype SizeRectangle (RECT POINTER);

program
    pRect = &Rectangle;
    SizeRectangle (pRect);
    // . . .
endprogram

function SizeRectangle (pRectangle)
begin
    pRectangle->sX = 10;
    pRectangle->sY = 5;
end;
```

{button ,AL(`Address operator;BYREF operator;Indirection operator;Member operator;Structure pointer operator',0,','')} [See also](#)

{button ,PI('','comment')} [Feedback](#)

Passing data by reference in a call to a DLL requires the use of the UseDLL and UnUseDLL functions, which are not available in the InstallShield for Microsoft Visual C++ edition.



String indexing

A string variable is an array of characters with a null terminator. You can reference individual characters within a string by specifying the string name followed by an index value within square brackets. Note that the first character in a string is in position 0.

In the example below, the function `BlankLeadingZeros` uses the string indexing technique to replace the leading zeros with blank characters in the string representation of a number.

```
prototype BlankLeadingZeros (BYREF STRING);
STRING szString;

program
    szString = "00001234";
    BlankLeadingZeros (szString);
    MessageBox (szString, INFORMATION);
endprogram

function BlankLeadingZeros (szString)
    INT iVal, iLength;
begin
    iVal = 0;
    iLength = StrLength (szString);
    while (szString[iVal] = "0") && (iVal <= iLength)
        szString[iVal] = " ";
        iVal = iVal + 1;
    endwhile;
end;
```

{button ,AL('Data types;String operators;String size',0,','')}

[See also](#)

{button ,PI('','comment')} [Feedback](#)



String size and autosize

InstallShield will automatically size a string buffer for you if you do not specify a size. By default, InstallShield autosizes the string buffer to 256 bytes. If you assign a string to that variable longer than 256 bytes (including the null terminator), InstallShield will increase the amount of space in memory reserved for that string variable (up to 512 bytes for 16-bit setups and 1024 bytes for 32-bit setups). However, be aware that when InstallShield autosizes a string, it does not decrease the amount of memory allocated if the string length decreases.

When manually declaring a string size, you must declare one byte for the null terminator. This means that if you want your string to hold up to 128 characters, you must declare it with a length of 129 to allow room for the null terminator. The minimum required size of a string is 2—one byte for one character and one byte for the null terminator.

When you are using a string that you have declared with a size, you must be aware of how that string might be used with other strings. For example, consider the following function call:

```
STRING szQuestion[20], szDefault[20], svResult[50];
program
    szQuestion = "Enter company name";
    szDefault = "InstallShield, Inc.";
    AskText (szQuestion, szDefault, svResult);
```

The size of the string svResult should be greater than or equal to the size of the string szDefault. If not, then szDefault, if accepted, will not fit into the svResult variable returned by the function. The easiest way to avoid conflicts is to let InstallShield autosize all strings.

{button ,AL(`Data types;String indexing',0,`,`)}}

[See also](#)

{button ,PI(`",`comment')} [Feedback](#)



Variable data

A variable is a named data item whose value can change during program execution. It must be declared in the following format:

```
data type    variable name[, variable name [...]];
```

A variable name may have a maximum of 32 characters. When more than one variable name is specified in a single declaration, the names must be separated by commas. Each a variable declaration must be terminated with a semicolon.

In the following example, seven variables are declared. Note that the last declaration creates three numeric variables.

```
BOOL    bValidEntry;  
LONG    lPopulation;  
STRING  szUserName[128]; //String explicitly sized  
STRING  szMessage;      //String autosized  
NUMBER  nFileSize, nDirSize, nDiskSpace;
```

Note that you can declare a string variable with or without an explicit size. String variables that are not declared with an explicit size are then sized automatically during a setup to accommodate the values assigned to them.

Autosizing is highly recommended for all string variables except those that will be passed to an external (DLL) function; in that case the string's size *must* be declared explicitly. The maximum size of a string is 512 bytes for 16-bit setups and 1024 bytes for 32-bit setups.



The names of InstallScript variables and functions are case-sensitive. For example, svItemCounter is not equivalent to svITEMCOUNTER.

{button ,AL(`Data types;Global vs local variables;Hungarian notation;String size and autosize',0,`,`')}} See
also

{button ,PI(`",`comment')} Feedback



Predefined constants

A predefined constant is an identifier reserved by InstallShield to represent a specific literal value. InstallShield uses predefined constants to represent certain data values that are passed to and returned by built-in functions. By using these predefined constant rather than literal values, you can improve the readability of your setup scripts.

You cannot change the value InstallShield assigns to a predefined constant. However, you can determine the value of a predefined constant by calling [sprintfBox](#), as shown in the example below, which displays the value of the predefined constant COMPONENT_FIELD_SELECTED:

```
sprintfBox (INFORMATION, "", "%d", COMPONENT_FIELD_SELECTED);
```

Although you can use a literal value in place of a predefined constant, InstallShield strongly recommends that you use predefined constants wherever indicated for a function.

The following list contains the predefined constants used by InstallShield. To see which functions use a specific predefined constant, click on it.

—

[_A](#) [B](#) [C](#) [D](#) [E](#) [F](#) [G](#) [H](#) [I](#) [L](#) [M](#) [N](#) [O](#) [P](#) [R](#) [S](#) [T](#) [U](#) [V](#) [W](#) [X](#) [Y](#)

[_MAX_LENGTH](#)

[_MAX_STRING](#)

A

[_A](#) [B](#) [C](#) [D](#) [E](#) [F](#) [G](#) [H](#) [I](#) [L](#) [M](#) [N](#) [O](#) [P](#) [R](#) [S](#) [T](#) [U](#) [V](#) [W](#) [X](#) [Y](#)

[AFTER](#)

[ALLCONTENTS](#)

[ALLCONTROLS](#)

[APPEND](#)

[ASKDESTPATH](#)

[ASKOPTIONS](#)

[ASKPATH](#)

[ASKTEXT](#)

B

[_A](#) [B](#) [C](#) [D](#) [E](#) [F](#) [G](#) [H](#) [I](#) [L](#) [M](#) [N](#) [O](#) [P](#) [R](#) [S](#) [T](#) [U](#) [V](#) [W](#) [X](#) [Y](#)

[BACK](#)

[BACKBUTTON](#)

[BACKGROUND](#)

[BACKGROUNDCAPTION](#)

[BADPATH](#)

[BADTAGFILE](#)

[BASEMEMORY](#)

[BEFORE](#)

[BILLBOARD](#)

[BINARY](#)

[BITMAP256COLORS](#)

[BITMAPFADE](#)

[BITMAPICON](#)

[BK_BLUE](#)

[BK_GREEN](#)

[BK_MAGENTA](#)

BK_MAGENTA1
BK_ORANGE
BK_PINK
BK_RED
BK_SMOOTH
BK_SOLIDBLACK
BK_SOLIDBLUE
BK_SOLIDGREEN
BK_SOLIDMAGENTA
BK_SOLIDORANGE
BK_SOLIDPINK
BK_SOLIDRED
BK_SOLIDWHITE
BK_SOLIDYELLOW
BK_YELLOW
BLACK
BLUE
BOOTUPDRIVE
BUTTON_CHECKED
BUTTON_ENTER
BUTTON_UNCHECKED
BUTTON_UNKNOWN

C

_A B C D E F G H I L M N O P R S T U V W X Y
CANCEL
CANCELBUTTON
CC_ERR_FILEFORMATERROR
CC_ERR_FILEREADERROR
CC_ERR_NOCOMPONENTLIST
CC_ERR_OUTOFMEMORY
CDROM
CDROM_DRIVE
CENTERED
CHANGEDIR
CHECKBOX
CHECKBOX95
CHECKLINE
CHECKMARK
CMD_CLOSE
CMD_MAXIMIZE
CMD_MINIMIZE
CMD_PUSHDOWN
CMD_RESTORE
COLORMODE256
COLORS
COMBOBOX_ENTER
COMBOBOX_SELECT
COMMAND

COMMANDEX
COMMON
COMP_DONE
COMP_ERR_CREATEDIR
COMP_ERR_DESTCONFLICT
COMP_ERR_FILENOTINLIB
COMP_ERR_FILESIZE
COMP_ERR_FILETOOLARGE
COMP_ERR_HEADER
COMP_ERR_INCOMPATIBLE
COMP_ERR_INTPUTNOTCOMPRESSED
COMP_ERR_INVALIDLIST
COMP_ERR_LAUNCHSERVER
COMP_ERR_MEMORY
COMP_ERR_NODISKSPACE
COMP_ERR_OPENINPUT
COMP_ERR_OPENOUTPUT
COMP_ERR_OPTIONS
COMP_ERR_OUTPUTNOTCOMPRESSED
COMP_ERR_SPLIT
COMP_ERR_TARGET
COMP_ERR_TARGETREADONLY
COMP_ERR_WRITE
COMP_INFO_ATTRIBUTE
COMP_INFO_COMPsize
COMP_INFO_DATE
COMP_INFO_INVALIDATEPASSWORD
COMP_INFO_ORIGsize
COMP_INFO_SETPASSWORD
COMP_INFO_TIME
COMP_INFO_VERSIONLS
COMP_INFO_VERSIONMS
COMP_NORMAL
COMP_UPDATE_DATE
COMP_UPDATE_DATE_NEWER
COMP_UPDATE_SAME
COMP_UPDATE_VERSION
COMPACT
COMPARE_DATE
COMPARE_SIZE
COMPARE_VERSION
COMPONENT_FIELD_CDROM_FOLDER
COMPONENT_FIELD_DESCRIPTION
COMPONENT_FIELD_DESTINATION
COMPONENT_FIELD_DISPLAYNAME
COMPONENT_FIELD_FILENEED
COMPONENT_FIELD_FTPLOCATION
COMPONENT_FIELD_HTTPLOCATION
COMPONENT_FIELD_MISC

COMPONENT_FIELD_OVERWRITE
COMPONENT_FIELD_PASSWORD
COMPONENT_FIELD_SELECTED
COMPONENT_FIELD_SIZE
COMPONENT_FIELD_STATUS
COMPONENT_FIELD_VISIBLE
COMPONENT_FILEINFO_COMPRESSED
COMPONENT_FILEINFO_COMPRESSENGINE
COMPONENT_FILEINFO_LANGUAGE
COMPONENT_FILEINFO_OS
COMPONENT_FILEINFO_POTENTIALLYLOCKED
COMPONENT_FILEINFO_SELFREGISTERING
COMPONENT_FILEINFO_SHARED
COMPONENT_INFO_ATTRIBUTE
COMPONENT_INFO_COMPSIZE
COMPONENT_INFO_DATE
COMPONENT_INFO_DATE_EX_EX
COMPONENT_INFO_LANGUAGE
COMPONENT_INFO_ORIGSIZE
COMPONENT_INFO_OS
COMPONENT_INFO_TIME
COMPONENT_INFO_VERSIONLS
COMPONENT_INFO_VERSIONMS
COMPONENT_INFO_VERSIONSTR
COMPONENT_VALUE_ALWAYSOVERWRITE
COMPONENT_VALUE_CRITICAL
COMPONENT_VALUE_HIGHLYRECOMMENDED
COMPONENT_VALUE_NEVEROVERWRITE
COMPONENT_VALUE_NEWERDATE
COMPONENT_VALUE_NEWERVERSION
COMPONENT_VALUE_OLDERDATE
COMPONENT_VALUE_OLDERVERSION
COMPONENT_VALUE_SAMEORNEWDATE
COMPONENT_VALUE_SAMEORNEWERVERSION
COMPONENT_VALUE_STANDARD
COMPONENT_VIEW_CHANGE
COMPONENT_VIEW_CHILDVIEW
COMPONENT_VIEW_COMPONENT
COMPONENT_VIEW_DESCRIPTION
COMPONENT_VIEW_MEDIA
COMPONENT_VIEW_PARENTVIEW
COMPONENT_VIEW_SIZEAVAIL
COMPONENT_VIEW_SIZETOTAL
COMPONENT_VIEW_TARGETLOCATION
COMPRESSHIGH
COMPRESSLOW
COMPRESSMED
COMPRESSNONE
CONTIGUOUS

CONTINUE
COPY_ERR_CREATEDIR
COPY_ERR_NODISKSPACE
COPY_ERR_OPENINPUT
COPY_ERR_OPENOUTPUT
COPY_ERR_TARGETREADONLY
COPY_ERR_MEMORY
CORECOMPONENTHANDLING
CPU
CUSTOM

D

_ A B C D E F G H I L M N O P R S T U V W X Y

DATA_COMPONENT
DATA_LIST
DATA_NUMBER
DATA_STRING
DATE
DEFAULT
DEFWINDOWMODE
DELETE_EOF
DIALOG
DIALOGCACHE
DIALOGTHINFONT
DIR_WRITEABLE
DIRECTORY
DISABLE
DISK
DISK_FREESPACE
DISK_TOTALSPACE
DISKID
DLG_ASK_OPTIONS
DLG_ASK_PATH
DLG_ASK_TEXT
DLG_ASK_YESNO
DLG_CANCEL
DLG_CDIR
DLG_CDIR_MSG
DLG_CENTERED
DLG_CLOSE
DLG_DIR_DIRECTORY
DLG_DIR_DRIVE
DLG_DIR_FILE
DLG_ENTER_DISK
DLG_ERR
DLG_ERR_ALREADY_EXISTS
DLG_ERR_ENDDLG
DLG_INFO_ALTIMAGE
DLG_INFO_CHECKMETHOD

DLG_INFO_CHECKSELECTION
DLG_INFO_ENABLEIMAGE
DLG_INFO_KUNITS
DLG_INFO_USEDECIMAL
DLG_INIT
DLG_MSG_ALL
DLG_MSG_INFORMATION
DLG_MSG_NOT_HAND
DLG_MSG_SEVERE
DLG_MSG_STANDARD
DLG_MSG_WARNING
DLG_OK
DLG_STATUS
DLG_USER_CAPTION
DRIVE
DRIVEOPEN
EDITBOX_CHANGE

E

_ A B C D E F G H I L M N O P R S T U V W X Y
EFF_BOXSTRIPE
EFF_FADE
EFF_HORZREVEAL
EFF_HORZSTRIPE
EFF_NONE
EFF_REVEAL
EFF_VERTSTRIPE
ENABLE
END_OF_FILE
END_OF_LIST
ENHANCED
ENTERDISK
ENTERDISK_ERRMSG
ENTERDISKBEEP
ENVSPACE
EQUALS
ERR_BADPATH
ERR_BADTAGFILE
ERR_BOX_BADPATH
ERR_BOX_BADTAGFILE
ERR_BOX_DISKID
ERR_BOX_DRIVEOPEN
ERR_BOX_EXIT
ERR_BOX_HELP
ERR_BOX_NOSPACE
ERR_BOX_PAUSE
ERR_BOX_READONLY
ERR_DISKID
ERR_DRIVEOPEN

EXCLUDE_SUBDIR
EXCLUSIVE
EXISTS
EXIT
EXTENDEDMEMORY
EXTENSION_ONLY

F

_ A B C D E F G H I L M N O P R S T U V W X Y

FADE_IN
FADE_OUT
FAILIFEXISTS
FALSE
FDRIVE_NUM
FEEDBACK
FEEDBACK_FULL
FEEDBACK_OPERATION
FEEDBACK_SPACE
FILE_ATTR_ARCHIVED
FILE_ATTR_DIRECTORY
FILE_ATTR_HIDDEN
FILE_ATTR_NORMAL
FILE_ATTR_READONLY
FILE_ATTR_SYSTEM
FILE_ATTRIBUTE
FILE_BIN_CUR
FILE_BIN_END
FILE_BIN_START
FILE_DATE
FILE_EXISTS
FILE_INSTALLED
FILE_INVALID
FILE_IS_LOCKED
FILE_LINE_LENGTH
FILE_LOCKED
FILE_MODE_APPEND
FILE_MODE_BINARY
FILE_MODE_BINARYREADONLY
FILE_MODE_NORMAL
FILE_NO_VERSION
FILE_NOT_FOUND
FILE_RD_ONLY
FILE_SIZE
FILE_SRC_EQUAL
FILE_SRC_OLD
FILE_TIME
FILE_WRITEABLE
FILENAME
FILENAME_ONLY

FINISHBUTTON
FIXED_DRIVE
FONT_TITLE
FREENVSPACE
FS_CREATEDIR
FS_DISKONEREQUIRED
FS_DONE
FS_FILENOTINLIB
FS_GENERROR
FS_INCORRECTDISK
FS_LAUNCHPROCESS
FS_OPERROR
FS_OUTOFSPACE
FS_PACKAGING
FS_RESETREQUIRED
FS_TARGETREADONLY
FS_TONEXTDISK
FULL
FULLSCREEN
FULLSCREENSIZE
FULLWINDOWMODE

G

_A B C D E F G H I L M N O P R S T U V W X Y
GREATER_THAN
GREEN

H

_A B C D E F G H I L M N O P R S T U V W X Y
HELP
HKEY_CLASSES_ROOT
HKEY_CURRENT_CONFIG
HKEY_CURRENT_USER
HKEY_DYN_DATA
HKEY_LOCAL_MACHINE
HKEY_PERFORMANCE_DATA
HKEY_USERS
HOURGLASS
HWND_DESKTOP
HWND_INSTALL

I

_A B C D E F G H I L M N O P R S T U V W X Y
IGNORE_READONLY
INCLUDE_SUBDIR
INDVFILESTATUS
INFO
INFO_DESCRIPTION
INFO_IMAGE

INFO_MISC
INFO_SIZE
INFO_SUBCOMPONENT
INFO_VISIBLE
INFORMATION
INVALID_LIST
IS_186
IS_286
IS_386
IS_486
IS_8514A
IS_86
IS_ALPHA
IS_CDROM
IS_CGA
IS_DOS
IS_EGA
IS_FIXED
IS_FOLDER
IS_ITEM
ISLANG_ALL
ISLANG_ARABIC
ISLANG_ARABIC_SAUDIARABIA
ISLANG_ARABIC_IRAQ
ISLANG_ARABIC_EGYPT
ISLANG_ARABIC_LIBYA
ISLANG_ARABIC_ALGERIA
ISLANG_ARABIC_MOROCCO
ISLANG_ARABIC_TUNISIA
ISLANG_ARABIC_OMAN
ISLANG_ARABIC_YEMEN
ISLANG_ARABIC_SYRIA
ISLANG_ARABIC_JORDAN
ISLANG_ARABIC_LEBANON
ISLANG_ARABIC_KUWAIT
ISLANG_ARABIC_UAE
ISLANG_ARABIC_BAHRAIN
ISLANG_ARABIC_QATAR
ISLANG_AFRIKAANS
ISLANG_AFRIKAANS_STANDARD
ISLANG_ALBANIAN
ISLANG_ALBANIAN_STANDARD
ISLANG_BASQUE
ISLANG_BASQUE_STANDARD
ISLANG_BULGARIAN
ISLANG_BULGARIAN_STANDARD
ISLANG_BELARUSIAN
ISLANG_BELARUSIAN_STANDARD
ISLANG_CATALAN

ISLANG_CATALAN_STANDARD
ISLANG_CHINESE
ISLANG_CHINESE_TAIWAN
ISLANG_CHINESE_PRC
ISLANG_CHINESE_HONGKONG
ISLANG_CHINESE_SINGAPORE
ISLANG_CROATIAN
ISLANG_CROATIAN_STANDARD
ISLANG_CZECH
ISLANG_CZECH_STANDARD
ISLANG_DANISH
ISLANG_DANISH_STANDARD
ISLANG_DUTCH
ISLANG_DUTCH_STANDARD
ISLANG_DUTCH_BELGIAN
ISLANG_ENGLISH
ISLANG_ENGLISH_UNITEDSTATES
ISLANG_ENGLISH_UNITEDKINGDOM
ISLANG_ENGLISH_AUSTRALIAN
ISLANG_ENGLISH_CANADIAN
ISLANG_ENGLISH_NEWZEALAND
ISLANG_ENGLISH_IRELAND
ISLANG_ENGLISH_SOUTHAFRICA
ISLANG_ENGLISH_JAMAICA
ISLANG_ENGLISH_CARIBBEAN
ISLANG_ESTONIAN
ISLANG_ESTONIAN_STANDARD
ISLANG_FAEROESE
ISLANG_FAEROESE_STANDARD
ISLANG_FARSI
ISLANG_FINNISH
ISLANG_FINNISH_STANDARD
ISLANG_FRENCH
ISLANG_FRENCH_STANDARD
ISLANG_FRENCH_BELGIAN
ISLANG_FRENCH_CANADIAN
ISLANG_FRENCH_SWISS
ISLANG_FRENCH_LUXEMBOURG
ISLANG_GERMAN
ISLANG_GERMAN_STANDARD
ISLANG_GERMAN_SWISS
ISLANG_GERMAN_AUSTRIAN
ISLANG_GERMAN_LUXEMBOURG
ISLANG_GERMAN_LIECHTENSTEIN
ISLANG_GREEK
ISLANG_GREEK_STANDARD
ISLANG_HEBREW
ISLANG_HEBREW_STANDARD
ISLANG_HUNGARIAN

ISLANG_HUNGARIAN_STANDARD
ISLANG_ICELANDIC
ISLANG_ICELANDIC_STANDARD
ISLANG_INDONESIAN
ISLANG_INDONESIAN_STANDARD
ISLANG_ITALIAN
ISLANG_ITALIAN_STANDARD
ISLANG_ITALIAN_SWISS
ISLANG_JAPANESE
ISLANG_JAPANESE_STANDARD
ISLANG_KOREAN
ISLANG_KOREAN_STANDARD
ISLANG_KOREAN_JOHAB
ISLANG_LATVIAN
ISLANG_LATVIAN_STANDARD
ISLANG_LITHUANIAN
ISLANG_LITHUANIAN_STANDARD
ISLANG_NORWEGIAN
ISLANG_NORWEGIAN_BOKMAL
ISLANG_NORWEGIAN_NYNORSK
ISLANG_POLISH
ISLANG_POLISH_STANDARD
ISLANG_PORTUGUESE
ISLANG_PORTUGUESE_BRAZILIAN
ISLANG_PORTUGUESE_STANDARD
ISLANG_ROMANIAN
ISLANG_ROMANIAN_STANDARD
ISLANG_RUSSIAN
ISLANG_RUSSIAN_STANDARD
ISLANG_SLOVAK
ISLANG_SLOVAK_STANDARD
ISLANG_SLOVENIAN
ISLANG_SLOVENIAN_STANDARD
ISLANG_SERBIAN
ISLANG_SERBIAN_LATIN
ISLANG_SERBIAN_CYRILLIC
ISLANG_SPANISH
ISLANG_SPANISH_ARGENTINA
ISLANG_SPANISH_BOLIVIA
ISLANG_SPANISH_CHILE
ISLANG_SPANISH_COLOMBIA
ISLANG_SPANISH_COSTARICA
ISLANG_SPANISH_DOMINICANREPUBLIC
ISLANG_SPANISH_ECUADOR
ISLANG_SPANISH_ELSALVADOR
ISLANG_SPANISH_GUATEMALA
ISLANG_SPANISH_HONDURAS
ISLANG_SPANISH_MEXICAN
ISLANG_SPANISH_MODERNSORT

ISLANG_SPANISH_NICARAGUA
ISLANG_SPANISH_PANAMA
ISLANG_SPANISH_PARAGUAY
ISLANG_SPANISH_PERU
IISLANG_SPANISH_PUERTORICO
ISLANG_SPANISH_TRADITIONALSORT
ISLANG_SPANISH_VENEZUELA
ISLANG_SPANISH_URUGUAY
SLANG_SWEDISH
SLANG_SWEDISH_FINLAND
SLANG_SWEDISH_STANDARD
ISLANG_THAI
ISLANG_THA_STANDARDI
ISLANG_TURKISH
ISLANG_TURKISH_STANDARD
ISLANG_UKRAINIAN
ISLANG_UKRAINIAN_STANDARD
ISLANG_VIETNAMESE
ISLANG_VIETNAMESE_STANDARD
IS_MIPS
IS_MONO
IS_OS2
ISOSL_ALL
ISOSL_WIN31
ISOSL_WIN95
ISOSL_NT351
ISOSL_NT351_ALPHA
ISOSL_NT351_MIPS
ISOSL_NT351_PPC
ISOSL_NT40
ISOSL_NT40_ALPHA
ISOSL_NT40_MIPS
ISOSL_NT40_PPC
IS_PENTIUM
IS_POWERPC
IS_RAMDRIVE
IS_REMOTE
IS_REMOVABLE
IS_SVGA
IS_UNKNOWN
IS_UVGA
IS_VALID_PATH
IS_VGA
IS_WIN32S
IS_WINDOWS
IS_WINDOWS95
IS_WINDOWSNT
IS_WINOS2
IS_XVGA

ISTYPE

L

_A B C D E F G H I L M N O P R S T U V W X Y

LANGUAGE

LANGUAGE_DRV

LESS_THAN

LINE_NUMBER

LISTBOX_ENTER

LISTBOX_SELECT

LISTFIRST

LISTLAST

LISTNEXT

LISTPREV

LOCKEDFILE

LOGGING

LOWER_LEFT

LOWER_RIGHT

M

_A B C D E F G H I L M N O P R S T U V W X Y

MAGENTA

MAINCAPTION

MATH_COPROCESSOR

MAX_STRING

MENU

METAFILE

MMEDIA_AVI

MMEDIA_MIDI

MMEDIA_PLAYASYNCH

MMEDIA_PLAYCONTINUOUS

MMEDIA_PLAYSYNCH

MMEDIA_STOP

MMEDIA_WAVE

MOUSE

MOUSE_DRV

N

_A B C D E F G H I L M N O P R S T U V W X Y

NETWORK

NETWORK_DRV

NEXT

NEXTBUTTON

NO

NO_SUBDIR

NO_WRITE_ACCESS

NONCONTIGUOUS

NONEXCLUSIVE

NORMAL

NORMALMODE

NOSET

NOTEXISTS

NOTRESET

NOWAIT

NULL

NUMBERLIST

O

_A B C D E F G H I L M N O P R S T U V W X Y

OFF

OK

ON

ONLYDIR

OS

OSMAJOR

OSMINOR

OTHER_FAILURE

OUT_OF_DISK_SPACE

P

_A B C D E F G H I L M N O P R S T U V W X Y

PARALLEL

PARTIAL

PATH

PATH_EXISTS

PAUSE

PERSONAL

PROFSTRING

PROGMAN

R

_A B C D E F G H I L M N O P R S T U V W X Y

RAM_DRIVE

REAL

RECORDMODE

RED

REGDB_APPPATH

REGDB_APPPATH_DEFAULT

REGDB_BINARY

REGDB_ERR_CONNECTIONEXISTS

REGDB_ERR_CORRUPTEDREGISTRY

REGDB_ERR_FILECLOSE

REGDB_ERR_FILENOTFOUND

REGDB_ERR_FILEOPEN

REGDB_ERR_FILEREAD

REGDB_ERR_INITIALIZATION

REGDB_ERR_INVALIDFORMAT

REGDB_ERR_INVALIDHANDLE

REGDB_ERR_INVALIDNAME
REGDB_ERR_INVALIDPLATFORM
REGDB_ERR_OUTOFMEMORY
REGDB_ERR_REGISTRY
REGDB_KEYS
REGDB_NAMES
REGDB_NUMBER
REGDB_STRING
REGDB_STRING_EXPAND
REGDB_STRING_MULTI
REGDB_UNINSTALL_NAME
REGKEY_CLASSES_ROOT
REGKEY_CURRENT_USER
REGKEY_LOCAL_MACHINE
REGKEY_USERS
REMOTE_DRIVE
REMOVE
REMOVEABLE_DRIVE
REPLACE
REPLACE_ITEM
RESET
RESTART
ROOT
ROTATE
RUN_MAXIMIZED
RUN_MINIMIZED
RUN_SEPARATEMEMORY

S

_ A B C D E F G H I L M N O P R S T U V W X Y
SELECTFOLDER
SELFREGISTER
SELFREGISTERBATCH
SELFREGISTRATIONPROCESS
SERIAL
SET
SETUPTYPE
SETUPTYPE_INFO_DESCRIPTION
SETUPTYPE_INFO_DISPLAYNAME
SEVERE
SHARE
SHAREDFILE
SILENTMODE
SPLITCOMPRESS
SPLITCOPY
SRCTARGETDIR
STANDARD
STATUS
STATUS95

STATUSBAR
STATUSDLG
STATUSEX
STATUSOLD
STRINGLIST
STYLE_BOLD
STYLE_ITALIC
STYLE_NORMAL
STYLE_SHADOW
STYLE_UNDERLINE
SW_HIDE
SW_MAXIMIZE
SW_MINIMIZE
SW_NORMAL
SW_RESTORE
SW_SHOW
SW_SHOWMAXIMIZED
SW_SHOWMINIMIZED
SW_SHOWMINNOACTIVE
SW_SHOWNA
SW_SHOWNOACTIVATE
SW_SHOWNORMAL
SYS_BOOTMACHINE
SYS_BOOTWIN
SYS_BOOTWIN_INSTALL
SYS_RESTART
SYS_SHUTDOWN
SYS_TODOS

T

_A B C D E F G H I L M N O P R S T U V W X Y
TEXT
TILED
TIME
TRUE
TYPICAL

U

_A B C D E F G H I L M N O P R S T U V W X Y
UPPER_LEFT
UPPER_RIGHT
USER_ADMINISTRATOR

V

_A B C D E F G H I L M N O P R S T U V W X Y
VALID_PATH
VARIABLE_LEFT
VARIABLE_UNDEFINED
VER_DLL_NOT_FOUND

VER_UPDATE_ALWAYS

VER_UPDATE_COND

VERSION

VIDEO

VOLUMELABEL

W

_A B C D E F G H I L M N O P R S T U V W X Y

WAIT

WARNING

WELCOME

WHITE

WIN32SINSTALLED

WIN32SMAJOR

WIN32SMINOR

WINDOWS_SHARED

WINMAJOR

WINMINOR

X

_A B C D E F G H I L M N O P R S T U V W X Y

XCOPY_DATETIME

Y

_A B C D E F G H I L M N O P R S T U V W X Y

YELLOW

YES

{button ,PI("",`comment')} Feedback



Flow control

Like most procedural languages, InstallScript processes statements sequentially, starting with the first statement in the program block and ending with the last. The linear flow of execution within a script can be controlled with conditional statements that perform branching and iteration. Branching is most commonly performed with an if statement that directs execution down one path or another. Iteration is performed with loop statements that execute one or more statements repeatedly, either for a set number of times or as long as a specified condition is met.

To control the flow of execution within scripts, InstallShield provides the following keywords:

abort

exit

for..endfor

goto

if..then..else..endif

repeat..until

return

switch..endswitch

while..endwhile



The process of a function calling itself directly is known as recursion. Indirect recursion occurs when one function calls a second function, and the second function calls the first function. Recursion is not recommended in setup scripts.

{button ,PI("",`comment')} [Feedback](#)



abort

When the script encounters an abort statement, the setup terminates. Then, unInstallShield runs in silent mode to reverse the incomplete installation, removing, for example, all partially or completely installed files that have been logged for uninstallation.

The abort statement is also encountered in the InstallShield default exit handler when the user exits the setup program before it has completed, for example, by pressing the Esc key, the F3 key, or the Cancel button of a dialog box. If you are defining your own exit handler, you should use the abort statement at the end of the handler definition so that InstallShield can remove the incompletely installed application.

{button ,PI("",`comment`)} [Feedback](#)



exit

When the setup program encounters an exit statement in the script it is executing, the setup process terminates. Ideally, each setup script contains, at most, one exit statement. If your script includes conditional expressions that might cause it to exit before the installation has completed, you should use abort instead of exit. In the main program block, use endprogram rather than exit as the last statement.

{button ,PI("", `comment')} Feedback



for...endfor

The for statement is designed to execute one or more statements a fixed number of times. It begins with the keyword for and an expression that specifies the number of times statements within the for structure are to be executed. The for structure ends with the keyword endfor. Note that the for statement itself is *not* terminated with a semicolon; however a semicolon is required after the endfor statement.

In the following example, the function MessageBox is called 10 times. On the first pass, iCount is set to 1, and since 1 is in the specified range (1 to 10), the message box is displayed. Then iCount is incremented by 1 and the for statement is resolved again. This time iCount = 2, still in the specified range; the message box is displayed a second time. When iCount is incremented after the tenth pass, its value becomes 11, which is outside the specified range; the for statement ends.

```
for iCount = 1 to 10
    MessageBox ("You'll see this 10 times", INFORMATION);
endfor;
```

You can count down from a higher number to a lower number by using the keyword downto in place of the keyword to. In the following example a message box is displayed three times. The first time the loop is entered, j is set to 20. Because downto specifies that the controlling variable be decremented and step 5 sets a decrement of 5 per loop, j will be equal to 15 the second time the loop is entered. The third time, j will be equal to 10.

```
for j = 20 downto 10 step 5
    MessageBox ("You will see this three times", INFORMATION);
endfor;
```

The default increment in a for statement is one (1), but you can use the keyword step to adjust the increment, as in the example below, which increases the value of iCount by 10 each time loop is executed. On the first pass, iCount = 10; on the second pass, iCount = 20; on the third pass, iCount = 30, and so on.

```
for iCount = 10 to 100 step 10
    MessageBox ("You'll see this 10 times", INFORMATION);
endfor;
```



You cannot define a label inside of a for statement.

{button ,AL(`for;while;repeat',0,`,`)} [See also](#)

{button ,PI(`",`comment')} [Feedback](#)



goto

The goto keyword is used to branch directly to the statement immediately following a specified label. In the following code fragment, the goto statement causes execution to continue with the MessageBox statement.

```
Name:
  AskText("Company name:", "", szSrc);
  if (szSrc = "") then
    MessageBox("Please enter the company name.", SEVERE);
    goto Name;
  endif;
```

A goto statement in the main program must specify a label that has been declared in the main program. A goto statement in a function must specify a label that has been declared in that function.



You cannot immediately precede the endprogram statement with a label.

{button ,AL(`if structure with goto',0,`,`)} [See also](#)

{button ,PI("",`comment')} [Feedback](#)



if

Use an if statement when you want your script to choose between two or more options. An if statement consists of the keyword if, a condition to be evaluated, the keyword then, and the keyword endif followed by a semicolon, as shown below:

```
if (condition) then
    // statements to be executed if condition is true
endif;
```

The condition can be one of the following:

- n A boolean or integer constant, variable or literal.
- n An expression that produces a boolean or integer result.
- n A function that returns an integer result.

The parentheses around the condition are optional, but highly recommended for readability.



Many InstallScript functions return a negative value when they fail. When using the result of InstallScript functions as the condition in an if statement, test for failure by using a statement like the one below:

```
if (FunctionA (ParameterOne) < 0) then
    // Statements to handle the failure
else
    // Statements when the function succeeds
```

InstallScript provides the following if statement structures:

[if-then structure](#)

[if-then-else structure](#)

[Nested if-then-else structure](#)

[elseif structure](#)

[if structure with goto](#)

{button ,PI("",`comment')} [Feedback](#)



if-then structure

The simplest if statement evaluates an expression and performs a specified action if the expression is true. If the expression is not true, InstallShield ignores the entire statement. For example:

```
if (szStringA = "exit") then
    AskYesNo ( "Are you sure you want to exit?" , NO );
endif;
```

If szStringA equals "exit", the test evaluates to TRUE (1) and the AskYesNo function is called. If szStringA contains anything else, the result is FALSE (0) and the entire statement is ignored.

The sample code below compares the values of the variable nDialog and the constant DLG_ERR. If they are equivalent, InstallShield executes the MessageBox function:

```
if (nDialog = DLG_ERR) then
    MessageBox ("Error has occurred", WARNING);
endif;
```



You may find that your if statement is easier to read when you place the expression to be evaluated in parentheses, but the parentheses are optional in InstallShield.

{button ,AL(`if;if-then structure;if-then-else structure;Nested if-then-else structure;elseif structure;if structure with goto'0,`,`')} [See also](#)

{button ,PI(`",`comment')} [Feedback](#)



if-then-else structure

An if statement can also specify one or more statements to be executed if the condition is false. This option is indicated with the keyword `else`, as shown below

```
if (condition) then
    // statements to be executed if condition is true
else
    // statements to be executed if condition is false
endif;
```

In the example below, if `szStringA` equals "exit", the test evaluates to TRUE (1), and the `AskYesNo` function is called. If `szStringA` is not equal to "exit", the result is FALSE (0), and the `MessageBox` function is called following the `else` statement.

```
if szStringA = "exit" then
    AskYesNo ("Are you sure you want to exit?" , NO );
else
    MessageBox ("Please wait... ", INFORMATION );
endif;
```

{button ,AL(`if;if-then structure;if-then-else structure;Nested if-then-else structure;elseif structure;if structure with goto',0,`,`')} [See also](#)

{button ,PI(``,`comment')} [Feedback](#)



Nested if-then-else structure

You can create nested if statements, in which one if statement is embedded in another:

```
if (first condition) then
  if (second condition) then
    // statements to be executed if second condition is true
  else
    // statements to be executed if second condition is false
  endif;
else
  if (third condition) then
    // statements to be executed if third condition is true
  else
    // statements to be executed if third condition is false
  endif;
endif;
```

In the following example, if the value of szStringA is "exit", AskYesNo is called. If the value of szStringA is "continue", the program displays a message box. If szStringA is not equal to either of those values, the user-defined function UserErrorHandler is called.

```
if szStringA = "exit" then
  AskYesNo ("Are you sure you want to exit?" , NO );
else
  if szStringA = "continue" then
    MessageBox ("Please wait...", INFORMATION );
  else
    UserErrorHandler;
  endif;
endif;
```

{button ,AL(`if-then structure;if-then-else structure;Nested if-then-else structure;elseif structure;if structure with goto',0,`,`')} [See also](#)

{button ,PI(`",`comment')} [Feedback](#)



elseif structure

InstallScript provides the elseif statement to create if structures in which the else branch of one if statement leads to another if statement:

```
if (first condition) then
    // statements to be executed if condition is true
elseif (second condition) then
    // statements to be executed if condition is false
elseif (third condition) then
    // statements to be executed if third condition is false
endif;
```

In the following example, if szStringA equals "exit", AskYesNo is called. If szStringA is not equal to "exit", the program continues to the elseif statement to test if szStringA is equal to "continue". If szStringA is equal to "continue", the result is TRUE and MessageBox is called. If szStringA is not equal to "continue", the program moves to the next elseif, and so on.

```
if szStringA = "exit" then
    AskYesNo ("Are you sure you want to exit?" , NO );
elseif szStringA = "continue" then
    MessageBox ("Please wait...", INFORMATION );
elseif szStringA = "reboot" then
    goto StartHere;
endif;
```



You cannot define a label inside an if statement.

{button ,AL('if;if-then structure;if-then-else structure;Nested if-then-else structure;elseif structure;if structure with goto',0,'')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



if structure with goto

InstallScript supports a special form of the if statement that can be used only with goto statements:

```
if condition goto labelname;
```

This special structure is has the following features:

- n The condition must be followed by a goto statement.
- n The keyword then is not used.
- n The keyword endif is not used.

In the following example, the user will be prompted to enter a company name as long as szSrc is a null string ("").

```
Name:
  AskText("Company name:", "", szSrc);
  if (szSrc = "") goto Name;
```

{button ,AL('if;if-then structure;if-then-else structure;Nested if-then-else structure;elseif structure;if structure with goto;goto',0,'')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



repeat...until

The repeat statement is analogous to the do...while loop in the C language. It also is very similar to the InstallScript while statement. There are two main differences between repeat and while in InstallShield:

- n The repeat statement must loop at least once. A while statement might not loop at all.
- n A while statement terminates when the expression evaluates as false. A repeat statement terminates when the expression evaluates as true.

Follow the steps below to create a repeat loop:

1. Set the variable you will be using in the conditional test as you would for a while loop.
2. Type repeat on its own line with no punctuation.
3. Build the operation(s) that you want repeated.
4. Add the operation that changes the test variable (for example, nCount = nCount + 1, or nCount = SomeVariable).
5. End the loop with an until statement containing the conditional test in parentheses.

The following example demonstrates repeat loop syntax:

```
nCount = 1;
repeat
    MessageBox("Count is less than 5", INFORMATION);
    nCount = nCount + 1;
until (nCount = 5);
```



You cannot define a label inside of the repeat statement.

{button ,AL(`for;while;repeat',0,`,`)} [See also](#)

{button ,JI(`LANGREF.HLP>Examples',`Nested_while_example')}

[Example](#)

{button ,PI(`",`comment')} [Feedback](#)



return

Use the return statement to return from a label called by the Handler function or to return a value from a user-defined function. When a return statement is encountered, program flow returns to the point at which the handler label or the function was called. When used to return from a call to a user-defined function, the return statement can return a specified long integer (LONG) value.

The return value of most user-defined functions will be either 0 (zero), indicating the success of the function, or a value less than zero (<0), indicating failure. You can assign a number to the return value by using a return statement above the end statement in the function block, as shown below:

```
return 52;
end;
```

This attribute allows you to return the value of a local variable to the program block, even though the variable itself is destroyed:

```
function MyFunc (ParamOne , ParamTwo)
NUMBER nNumber;
begin
    nNumber = ParamOne + ParamTwo;
    . . .
    . . .
    return nNumber;
end;
```



By default, the return value is stored in a variable of type LONG.

{button ,PI("",`comment')} [Feedback](#)



switch...endswitch

The switch statement is similar to the if...elseif statement. Use the switch statement to execute one of several different sections of code, depending on the value of an expression. The switch statement evaluates the expression and then branches to the case statement whose constant value matches the result of the expression. If no match is found among the case statements, control passes to a default statement, if one has been specified.

To create a switch statement, follow the steps listed below:

1. First type the keyword switch, followed by the expression to be evaluated. The expression, which can be a constant, variable, arithmetic expression, logical expression, or function result, must be enclosed within parentheses. Do not punctuate this line.
2. For each option, type the keyword case and one or more constants followed by a colon. If more than one constant is specified, delimit them with commas. Note that only constants can be specified here. Specifying a variable name, string table reference, function result, or other type of expression after the keyword case will result in an error.
3. For each case, follow the colon with the statement or statements to be executed for that option. Terminate each statement with a semicolon.
4. After all case statements have been specified, use the keyword default, followed by a colon (:), to control the program when the expression does not match any of the stated cases.
5. Close the block with the keyword endswitch, followed by a semicolon (;).

The following script displays the current video resolution of the computer on which it is executed:

```
STRING szMsg, svResult;
NUMBER nvResult;
program
  GetSystemInfo(VIDEO, nvResult, svResult);
  switch (nvResult)
    case IS_UNKNOWN: szMsg = "The user's video is unknown.";
    case IS_EGA      : szMsg = "EGA resolution.";
    case IS_VGA      : szMsg = "VGA resolution.";
    case IS_SVGA     : szMsg = "Super VGA (800 x 600) resolution.";
    case IS_XVGA     : szMsg = "XVGA (1024 x 768) resolution.";
    case IS_UVGA     : szMsg = "Greater than 1024 x 768 resolution.";
    default          : szMsg = "Error";
  endswitch;
  MessageBox(szMsg, INFORMATION);
endprogram
```



Only one case block will be executed each time a switch statement is executed. After InstallShield executes a case block, it executes the next statement after the endswitch.

A switch block can be quite useful inside of a while loop. By using the case statements as flags, you can create a loop with optional exit points.



while...endwhile

Use the while statement when you want to execute one or more statements repeatedly, as long as a particular condition is true. If the condition is not true when the statement is first executed, the loop is not performed at all.

To create a while loop, follow these steps:

1. Set the variable you are using as the condition to an initial state.
2. Type the keyword while, followed by the conditional test in parentheses. Do not punctuate this line.
3. Build the operation(s) that you want repeated.
4. Add the operation that changes the test variable (for example, nCount = nCount + 1, or nCount = SomeVariable).
5. End the loop by typing endwhile, followed by a semicolon.

In the following example, the message box is displayed four times.

```
nCount = 1;
while (nCount < 5)
    MessageBox ("This is still true.", INFORMATION);
    nCount = nCount + 1;
endwhile;
```

Because nCount is assigned an initial value of 1, the while statement evaluates TRUE the first time it is executed; the message box is displayed and nCount is incremented by 1. After the fourth pass through the loop, nCount is equal to 5; the while statement evaluates FALSE and the program continues executing with the statement after endwhile.

You cannot define a label within a while block. You can, however, nest while statements in InstallShield. Remember that you must end each while block with endwhile.

{button ,AL(`for;while;repeat',0,`,`')} [See also](#)

{button ,JI(`LANGREF.HLP>Examples',`Nested_while_example')}

[Example](#)

{button ,PI(`",`comment')} [Feedback](#)



Nested while example

```
/* This Script illustrates a nested while loop.
 * This script searches for the specified type of files and
 * will show the number of lines in each file.
 */

#define SOURCE DIR "c:\\example";

LIST    listID;
STRING  svTarget, svResult, filename, svLine, szPath, szFileName;
NUMBER  nResult, nOp, nFileHandle, count;

program

    count = 0;
    nOp = RESET;
    svTarget = SOURCE DIR;
    listID = ListCreate (STRINGLIST);

while FindAllFiles (svTarget, "*.txt", svResult, nOp) = 0;

    // To get the name of the file in the fully specified path name
    StrGetTokens(listID, svResult, "\\");
    ListCurrentString(listID, filename);
    // Set the file mode to normal.
    OpenFileMode(FILE_MODE_NORMAL);
    szFileName = filename;
    szPath      = svTarget;
    // The following opens the file for editing.
    OpenFile(nFileHandle, szPath, szFileName);

/*-----*\
 *
 * The following retrieves each line of text from the open file and increments
 * a count to find the number of lines.
 *
 *-----*/

    while (GetLine (nFileHandle, svLine) = 0)
        count = count + 1;
    endwhile;

    sprintfBox(INFORMATION, "The Total lines in the file", "The No. of lines in the
file %s is %d", filename, count);

    count = 0;
    // The following closes the file
    CloseFile(nFileHandle);

// Continue searching files where last file was left off
nOp = CONTINUE;

    if (FindAllFiles (svTarget, "*.txt", svResult, nOp) < 0) then
        abort;
    endif;
endwhile;

endprogram
```

{button ,PI("",`comment')} Feedback



Functions

A function is a named set of instructions that operate together to perform a certain task. Every function has the following characteristics:

- n A function is named. Each function has a unique name. When you call the function by name, you know which set of instructions will run, and you can be sure of consistent results. You can also call a function from within another function.
- n A function is independent. In most cases, any function can perform its instructions without interfering with other parts of the program.
- n A function performs a certain task. A task is any single job that the script must perform, such as displaying a bitmap, compressing a file, or creating a folder.
- n A function can return a value to the script. When the script executes, it performs the instructions of the function. Based on the result of the instructions, a function can return information to the script.

InstallShield allows you to use three types of functions in your setup script:

- n Built-in functions, which are supplied by InstallShield or included for Sd dialog boxes.
- n User-defined functions, which you create.
- n Functions you can call in a DLL.



Like the C programming language, InstallShield does not support nested function blocks.

{button ,AL(`Built in Functions;Building a Function;Calling a Function;Declaring Functions;Calling a DLL Function;Returning a Value from a Function',0,`,`)}` See also

{button ,PI("",`comment')} Feedback



Built-in functions

InstallShield has over 250 built-in functions that you can use in your setup scripts to create program groups and items, manipulate folders, work with lists, monitor the status of the setup, create dialog boxes, manipulate files, and much more. Because the InstallShield Script Compiler already recognizes these function names, you do not have to declare them before you can use them.

In order to make a successful call to a built-in function, you must know both its name and its format. All of the built-in functions are listed in the [Built-in function index](#). To display a complete description of any function in that list, just click on its name. In the help topic for that function you will find the function's format.

For example, [AskYesNo](#) is a built-in function that displays a query in a dialog box and then waits for the end user to respond by clicking a button, either Yes or No. AskYesNo has the following format:

```
AskYesNo (szQuestion, nDefault);
```

The format shows the correct spelling of the function name, which is followed by the function's parameter list, enclosed in parentheses. In the help topic for a built-in function, each parameter is expressed in [Hungarian notation](#), which indicates the type of data that must be passed in that position. AskYesNo requires two parameters: the first parameter is a string; the second is a number.



Like C, InstallScript is case sensitive. Be sure to pay close attention to the capitalization of letters in the built-in function names.

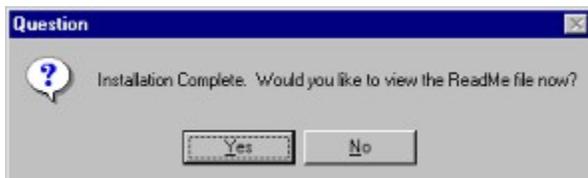
To use this or any function in your script, pass the required number of parameters, and make sure that the data you pass in each parameter is the type indicated for that position. If you pass the incorrect number of parameters, or if you pass the wrong type of data in any parameter positions, the script will not compile.



If you pass a string literal to a function, you must enclose it in single or double quotation marks: "Please wait while files are transferred", or 'This is a string', or "C:\Myfolder\Myfile.ext".

The help topic for each built-in function also provides a description of its parameters. For AskYesNo, szQuestion is the question to be displayed in the dialog box; nDefault indicates which button to preselect, Yes or No. One of two [predefined constants](#) may be passed in nDefault: YES or NO.

Consider the following dialog box, in which the Yes button is preselected:



To display this dialog box, you would call AskYesNo as shown below:

```
AskYesNo ("Installation Complete. Would you like to view the ReadMe file now?", YES);
```

{button ,AL(^EZ Batch File Fn;Advanced Batch File Fn;Built in Dialog Box Fn;EZ CONFIG SYS File Fn;Advanced Config File Fn;Custom Dialog Box Fn;Data Structure Memory Access Fn;Extensibility Fn;File Directory Fn;Information Fn;List Processing Fn;Long Filename Fn;Miscellaneous Fn;Path String Modification Fn;INI File Profile File Fn;Registry Fn;Sd Dialog Box Fn;Shell Fn;String Fn;Version Checking Fn;Visual Interface Fn',0,'')}
[See also](#)

{button ,PI("",`comment')} [Feedback](#)



Declaring functions

The first step in creating a user-defined function is to declare the function. The keyword `prototype` tells the InstallShield Script Compiler that the line contains a function definition. Use the following steps to declare a function:

1. Type the keyword `prototype`.
2. On the same line enter the function name.
3. After the function name, enter the data types of the parameters. The list must be enclosed in parentheses and separated by commas.
4. If there are no parameters, put empty parentheses to the right of the function name.
5. End the line with a semicolon (`;`).

In the following example, `FunctionName` is a function containing three parameters. The arguments passed when calling `FunctionName` must be, in order, an `INT`, a `STRING`, and a `SHORT`. `CopyBitmapExample` has no parameters. `FileTransfer` has five parameters—three `LONG` variables and two `STRING` variables.

```
prototype FunctionName (INT, STRING, SHORT);
prototype CopyBitmapExample ();
prototype FileTransfer (LONG, LONG, LONG, STRING, STRING);
```

When declaring DLL functions, use the format `<DLL filename>.<function name>` for the name of the DLL function. For example:

```
prototype MyDLL.MyFunction (INT, INT);
```

The above declaration signals to the InstallShield Script Compiler that the program will call a function named `MyFunction`, with two `INT` parameters, in a file named `Mydll.dll`.

{button ,AL('Building a function;Calling a function;Returning a value from a function',0,','')}[See also](#)

{button ,PI('','comment')} [Feedback](#)



Building a function

After you declare a function prototype, you need to define the function itself in the function block. Each function block contains only one function. Place the function blocks after the exit statement of the program block. User-defined functions must be the last items in the script.

1. Start the function body with the keyword `function`, followed by the function name.
2. To the right of the function name, type the names of the arguments that you are using in parentheses. The arguments must, of course, correspond in data type to the parameters you declared in the declare block.

Steps 1 and 2 create what is known as a function header. Function headers do not end with a semicolon in InstallScript.

3. Next, declare any local variables you will be using in the function. Then type the keyword `begin` on a line by itself, without punctuation.
4. After the `begin` line, add whichever statements you need in order to accomplish your particular task.
5. You may also use a `return` statement, particularly if you want to return a specific value from the function. (See below for information on returning values from a function.)
6. Always end your function with the keyword `end`.

A sample function block is shown below:

```
function SetupScreen ()
    number nDx, nDy;
begin
    GetExtents( nDx, nDy );
    Enable (FULLWINDOWMODE);
    Enable (INDVFILESTATUS);
    Enable (BITMAP256COLORS);
    Enable (DIALOGCACHE);

    SetTitle ("Installing " + APP_NAME, 24, WHITE);
    SetColor (BACKGROUND, BK_BLUE); // Dark blue.
    SetColor (STATUSBAR, BLUE); // Bright blue.

    SetTitle ("Setup", 0, BACKGROUNDCAUTION); // Caption bar text.
    Enable (BACKGROUND);
    Delay (1);
end;
```



User-defined functions can return a `LONG` value with a [return](#) statement. Other types of data can be returned in parameters that have been declared with the [BYREF operator](#).

{button ,AL(`Declaring functions;Calling a function;Returning a value from a function',0,)} [See also](#)

{button ,PI("",`comment')} [Feedback](#)



Returning a value from a function

Like InstallShield's built-in and Sd dialog functions, user-defined functions can be designed to return a numeric value to the caller. To return a value from a function, you must include a return statement, followed by the value to be returned, before the function's end statement. If you do not include a return statement or if you do not specify a value after the keyword return, the value returned by the function is unpredictable.

Many programmers use return statements to return error codes that indicate the success or failure of a function call. Most of InstallShield's built-in and Sd dialog functions use a return statement for just that purpose. The return statement also is used commonly to create functions that return the result of an operation performed on parameters passed to the function, as in the example below, which returns the area of a rectangle:

```
function RectangleArea (nLength, nWidth)
  INT nVal;
  begin
    nVal = nLength * nWidth;
    return nVal;
  end;
```

The keyword return can be followed by a constant, variable, numeric expression, or function call. In the example below, RectangleArea has been modified to eliminate the assignment statement; the arithmetic expression follows the keyword return:

```
function RectangleArea (nLength, nWidth)
  begin
    return nLength * nWidth;
  end;
```

The numeric value returned by a function can be ignored by the calling program or function, tested in a conditional expression, or assigned to a variable. In the following example, the return value from RectangleArea is assigned to the variable nArea:

```
nArea = RectangleArea (nLong, nWide);
```

In the next example, the result of RectangleArea is tested in a conditional expression:

```
if RectangleArea(nLong, nWide) > nMaxArea then
  MessageBox("Area exceeds maximum allowed.", INFORMATION);
endif;
```



To return more than one value or non-numeric values, use the BYREF operator to define parameters that are passed by reference.

{button ,AL(' Building a function;Calling a function;Declaring functions;Returning a value from a function;BYREF operator;return',0,'')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



Calling a function

You call all functions, whether user-defined, built-in, Sd, or external, in the same way. Type the name of the function, followed by the parameters you want to pass to the function. For example:

```
MyFunction (MyAge, MyHeight, MyWeight);
```

{button ,AL(`Building a function;Declaring functions;Returning a value from a function;BYREF operator;return'0,`,`')}}[See also](#)

{button ,PI("",`comment`')}} [Feedback](#)



Overview: InstallScript lists

Lists are used to store related information, such as strings or numbers. InstallScript lists are very similar to single-linked lists in the C language. InstallShield list functions are very flexible, allowing you to return information in an order different from the order in which it was stored and access and use that information in a variety of ways.

List functions

InstallShield provides a number of functions for creating and manipulating lists. There are three types of InstallScript list functions:

- n Functions ending in -String that work with string lists only
- n Functions ending in -Item that work with number lists only
- n Functions that work with either string or number lists

InstallShield also has many secondary list-related functions that use or create lists.

List structure

Lists are used to store related information, either strings or numbers. All the information in a list must be of the same data type, and the number of elements is limited only by the available memory.

An InstallScript list has two parts. The first part is the head, which InstallShield uses internally. The head of the list contains general information about the list, such as whether it contains strings or numbers. The head also contains pointers to the beginning and end of the list.

The second part of the list is the list body. The list body contains the actual strings or numbers. You can have as many strings or numbers in a list as the memory in the system will allow.



Remember that lists cannot contain both numbers and strings. A list must have only strings or only numbers.

Variables representing lists can be declared as type LIST or type LONG. Lists exist only in memory, meaning they are destroyed when the setup is complete. If a list is local to a function, the list is destroyed when the function returns control to the calling code.

{button ,AL('The InstallScript functions for processing lists',0,'')} See also
{button ,PI('','comment')} [Feedback](#)



The InstallScript functions for processing lists

There are two types of lists, string lists and number lists. Two sets of functions are provided to work with lists: functions ending in -String that work with string lists only, and functions ending in -Item that work with number lists only. There is a third major group of InstallScript functions that work with either string or number lists. You cannot use number list functions on string lists and vice versa.

In addition, there are many secondary list-related functions that use or create lists. Below are all InstallScript list processing functions, according to category.

List functions for processing string lists

ListAddString

Adds a string to the list.

ListCurrentString

Returns the current string in the list.

ListDeleteString

Deletes the current string in the list.

ListFindString

Attempts to find a string element in a list. If found, the element becomes the current element of the list.

ListGetFirstString

Retrieves the first string from a string list.

ListGetNextString

Retrieves the element after the current element from a string list.

ListReadFromFile

Reads a text file into a list.

ListSetCurrentString

Sets the current element of a string list.

ListWriteToFile

Writes a string list into a file.

List functions for processing number lists

ListAddItem

Adds an item to the list.

ListCurrentItem

Returns the current item in the list.

ListDeleteItem

Deletes the current item in the list.

ListFindItem

Attempts to find a numeric element in a list. If found, the element becomes the current element of the list.

ListGetFirstItem

Retrieves the first element from a number list.

ListGetNextItem

Retrieves the element after the current element from a number list.

ListSetCurrentItem

Sets the current element of a number list.

List functions for processing both string and number lists

ListCount

Sees how many string or numeric elements a specified list contains.

ListCreate

Creates a new string or number list.

ListDestroy

Destroys a list.

ListSetIndex

Sets the current element of the list as an index.

Secondary list-related functions

ComponentListItems

Creates a string list of component items for use with component-related functions.

CtrlGetMultCurSel

Places the currently selected lines from a multi-selection list box control into a string list.

CtrlGetMLEText

Places the text of a multi-line edit field control into a string list.

CtrlSetList

Places the contents of a string list into a single- or multi-selection list box or combo box control.

CtrlSetMLEText

Uses a string list to set the text of a multi-line edit box control.

GetGroupNameList

Places all program group names currently existing in the Program Manager shell into a string list.

StrGetTokens

Parses a string into a list of tokens, which are stored in a string list.

{button ,PI("", `comment')} Feedback



Creating and destroying lists

Before creating a list, decide what type of list you want to build: a string list or a number (item) list. To create the list, call the [ListCreate](#) function as necessary:

```
// This builds the list head for a string list.  
listID1 = ListCreate (STRINGLIST);
```

or

```
// This builds the list head for a number (item) list.  
listID2 = ListCreate (NUMBERLIST);
```

ListCreate automatically builds the head of the list and returns its ID number. The ID is used in all subsequent functions that operate on the list. Therefore, you must always create a list using ListCreate before you use any other list function. You must store the return value from ListCreate in a variable of type LIST or type LONG.

This fragment creates a number list and then a string list. It also tests each one to make sure that the lists were created successfully.

```
// This creates an empty list for strings.  
listID1 = ListCreate (STRINGLIST);  
if (listID1 = LIST_NULL) then  
    MessageBox ("Unable to create the string list", SEVERE);  
endif;  
  
// This will create an empty list for numbers.  
listID2 = ListCreate (NUMBERLIST);  
if (listID2 = LIST_NULL) then  
    MessageBox ("Unable to create the number list", SEVERE);  
endif;
```

When you are finished using a list, you will typically want to destroy the list to free the memory for other uses. [ListDestroy](#) destroys the list and its contents. This example creates a list referenced by listID, adds a string to the list, and then destroys the entire list.

```
listID = ListCreate (STRINGLIST);  
if (listID = LIST_NULL) then  
    MessageBox ("Unable to create list.", SEVERE);  
    abort;  
endif;  
  
ListAddString (listID, "This is a string in the list", AFTER);  
ListDestroy (listID);
```

If you do not destroy a list using ListDestroy, the list will be destroyed when the setup is complete. If the list is local to a function, the list is destroyed when the function returns control to the calling code.

{button ,PI("",`comment')} [Feedback](#)



Traversing lists

InstallShield provides these functions for traversing lists incrementally and non-incrementally:

ListCount

Sees how many string or numeric elements a specified list contains.

ListCurrentItem

Returns the current item in the list.

ListCurrentString

Returns the current string in the list.

ListFindItem

Attempts to find a numeric element in a list. If found, the element becomes the current element of the list.

ListFindString

Attempts to find a string element in a list. If found, the element becomes the current element of the list.

ListGetFirstItem

Retrieves the first element from a number list.

ListGetFirstString

Retrieves the first string from a string list.

ListGetNextItem

Retrieves the element after the current element from a number list.

ListGetNextString

Retrieves the element after the current element from a string list.

ListSetIndex

Sets the current element of the list as an index.

InstallShield uses single-linked lists, which means that unless you use functions that set indexes or search for specific elements in lists, you can traverse lists incrementally in one direction only: from the first element to the last.

InstallShield allows you to traverse lists in non-incremental fashion by means of indexes and by searching for particular elements in lists. Refer to the individual function descriptions for more details.

Most list traversing and list access operations are carried out relative to the current list element. Furthermore, most of the functions used to traverse and access lists establish a current element as a result of their action. Therefore, making an element the current element in a list is not an isolated action; it is a byproduct of another action.

If a list is empty, adding an element to the list will establish a current element. If a list is not empty, then making an element the current element is best accomplished by traversing the list or searching for a particular element in the list.



Lists are often processed within while loops, usually checking for END_OF_LIST. An infinite loop can result if the list is not valid. If you're processing lists in a while loop, make sure that you have created the list with the ListCreate function, and that you haven't destroyed the list with the ListDestroy function.

```
{button ,AL('Find a particular element in a list;Get the first and next elements in a list;Set an index in a list',0,',')}  
  See also
```

```
{button ,PI('','comment')} Feedback
```



Get the first and next elements in a list

Call the [ListGetFirstString](#) function or the [ListGetFirstItem](#) function to return the first string element or number element, respectively, from a list. The element you retrieve then becomes the current element in the list.

Call the [ListGetNextString](#) function or the [ListGetNextItem](#) function to return the string element or number element after the current element in a list. The element you retrieve then becomes the current element in the list.

```
// Create the empty list of strings.
listID = ListCreate (STRINGLIST);

// Test for a valid list.
if (listID = LIST_NULL) then
    MessageBox ("List not created", SEVERE);
endif;

// Add some strings to the list.
ListAddString (listID, "String 1", AFTER);
ListAddString (listID, "String 2", AFTER);
ListAddString (listID, "String 3", AFTER);

// Traverse the list and display the strings in a message box.
lResult = ListGetFirstString(listID, szDriveName);
while (lResult != LIST_NULL)
    MessageBox (szDriveName, INFORMATION);
    lResult = ListGetNextString (listID, szDriveName);
endwhile;
```

{button ,PI("",`comment')} [Feedback](#)



Find a particular element in a list

You can traverse lists non-incrementally. Call the [ListFindString](#) function or the [ListFindItem](#) function when you want to search for a specific string or number element in a list. These two functions begin their search at the current element and continue forward through the list from that point.

To start a search from the beginning of a list, call the [ListGetFirstString](#) or the [ListGetFirstItem](#) function before calling the ListFindString or the ListFindItem function.

When the ListFindString or the ListFindItem function finds the specified string or number, it becomes the current element in the list.

In the script fragment below, a number list is created, and the number 1 (in nItem) is added as the first element. Then, ListFindItem searches the list for the number 1, and deletes it, if found. Finally, the list is destroyed.

```
listID = ListCreate (NUMBERLIST);  
if (listID = LIST_NULL) then  
    MessageBox ("Unable to create list.", SEVERE);  
    abort;  
endif;  
nItem = 1;  
ListAddItem (listID, nItem, AFTER);  
if (ListFindItem (listID, nItem) = 0) then  
    ListDeleteItem (listID);  
endif;  
ListDestroy (listID);
```



The ListFindString and ListFindItem functions look only for the first instance of the specified string or number at or after the current element.

{button ,PI("",`comment')} [Feedback](#)



Set an index in a list

InstallShield provides includes the [ListSetIndex](#) function, which lets you make an element the current element using an index number. If you know the location of a particular element in a list, you can call the ListSetIndex function to access that element immediately. You can traverse a list in either direction by using the index to set a specific element in a list to the current element. The index of the list starts at 0 (zero).

The ListSetIndex function works on both string and number lists. After you set the indexed element as the current element, call either the [ListCurrentItem](#) function or the [ListCurrentString](#) function to return the value of the indexed item.

This example demonstrates traversing a list non-incrementally using ListSetIndex.

```
listID = ListCreate (STRINGLIST);
GetGroupNameList (listID);
nCheck = ListSetIndex (listID, LISTFIRST);

while (nCheck != END_OF_LIST)
    ListCurrentString (listID, svString);
    MessageBox (svString, INFORMATION);
    nCheck = ListSetIndex (listID, LISTNEXT);
endwhile;

ListDestroy (listID);
```

The [ListCount](#) function tells you how many elements are in a list. The ListCount function is used mainly for general information purposes, although it can be used to establish an upper index value in conjunction with ListSetIndex. For example, you can call ListCount to get the number of elements in a list, and use that value with ListSetIndex to traverse a list. The above example, which uses a while loop, is rewritten below using an InstallScript for loop based on the number of elements in the list.

```
listID = ListCreate (STRINGLIST);
GetGroupNameList (listID);

// Get the number of elements in the list.
nItems = ListCount (listID);

// Display the number of elements in the list.
PrintfBox (INFORMATION, "", "i = %d", nItems);

// Loop for nItems times beginning with zero,
// displaying each list element in turn in a message box.
for i = 0 to (nItems - 1)
    ListSetIndex (listID, i);
    ListCurrentString (listID, svString);
    MessageBox (svString, INFORMATION);
    nCheck = ListSetIndex (listID, LISTNEXT);
endfor;

ListDestroy (listID);
```

{button ,PI("",`comment')} [Feedback](#)



Adding elements to lists

InstallShield provides these functions for adding elements to lists:

ListAddItem

Adds an item to the list.

ListAddString

Adds a string to the list.

ListReadFromFile

Reads a text file into a list.

ListAddString and ListAddItem add a single element to the list you specify. Remember that, regardless of where you place the new string in the list, it becomes the current string. Use the parameters BEFORE and AFTER to indicate where you want to place the new element in the list relative to the current element. If you are working with a newly-created list, using either BEFORE or AFTER will add the string to the first element position in the list.

Adding elements to a list and the resulting effects on the list order and the element in the current position are most easily explained by example. The examples below use string lists and ListAddString, but the same principles and steps apply to using ListAddItem and number lists. Consider these scenarios:

{button ,JI('`,`Adding_an_element_to_an_empty_list')} Adding an element to an empty list.

{button ,JI('`,`Adding_an_element_before_the_current_element')} Adding an element before the current element.

{button ,JI('`,`Adding_an_element_after_the_current_element')} Adding an element after the current element.

{button ,JI('`,`Adding_elements_before_and_after_the_current_element')} Adding elements before and after the current element.

{button ,PI('`,`comment')} Feedback



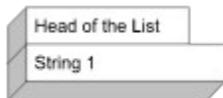
Adding an element to an empty list

The first string you add to the list goes immediately after the head of the list. This string ("String 1" below) becomes the current string in the list. The script fragment shown below results in a list like that depicted after it:

```
// Create the empty list of strings.
listID = ListCreate (STRINGLIST);

// Test for a valid list
if (listID = LIST_NULL) then
    MessageBox ("List not created", SEVERE);
endif;

// Add some strings to the list.
szString = "String 1";
ListAddString (listID, szString, AFTER);
```



{button ,PI("",`comment`)} [Feedback](#)



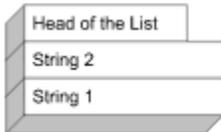
Adding an element before the current element

If the current string is the first string in the list and you add a new string before it, the new string becomes the first string in the list. The string that was formerly in the first element position now resides in the second element position. The new string at the first element position is now the current string:

```
// Create the empty list of strings.
listID = ListCreate (STRINGLIST);

// Test for a valid list
if (listID = LIST_NULL) then
    MessageBox ("List not created", SEVERE);
endif;

// Add some strings to the list.
szString = "String 1";
ListAddString (listID, szString, AFTER);
szString = "String 2";
ListAddString (listID, szString, BEFORE);
```



{button ,PI("",`comment')} [Feedback](#)



Adding an element after the current element

If the current string is the first string in the list, and you add the new string after the current string, the new string becomes the second string in the list, as well as the new current string. Refer to the script fragment below and to the illustration following it:

```
// Create the empty list of strings.
listID = ListCreate (STRINGLIST);

// Test for a valid list
if (listID = LIST_NULL) then
    MessageBox ("List not created", SEVERE);
endif;

// Add some strings to the list.
szString = "String 1";
ListAddString (listID, szString, AFTER);
szString = "String 2";
ListAddString (listID, szString, AFTER);
```



{button ,PI("",`comment')} [Feedback](#)



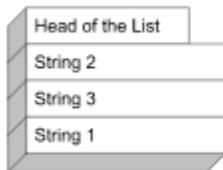
Adding elements before and after the current element

As another example, the code segment shown below creates a new list and puts "String 1" in the first position. "String 2" is then added before "String 1", leaving "String 2" in the first position as the current string. Next, "String 3" is added after the current string, resulting in the list depicted below.

```
// Create the empty list of strings.
listID = ListCreate (STRINGLIST);

// Test for a valid list
if (listID = LIST_NULL) then
    MessageBox ("List not created", SEVERE);
endif;

// Add some strings to the list.
szString = "String 1";
ListAddString (listID, szString, AFTER);
szString = "String 2";
ListAddString (listID, szString, BEFORE);
szString = "String 3";
ListAddString (listID, szString, AFTER);
```

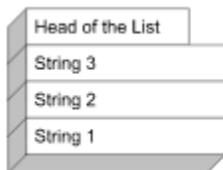


In the example above, if "String 3" were added before the current string, the result would be as shown below, with "String 3" becoming the current string. This code segment is shown below:

```
// Create the empty list of strings.
listID = ListCreate (STRINGLIST);

// Test for a valid list
if (listID = LIST_NULL) then
    MessageBox ("List not created", SEVERE);
endif;

// Add some strings to the list.
szString = "String 1";
ListAddString (listID, szString, AFTER);
szString = "String 2";
ListAddString (listID, szString, BEFORE);
szString = "String 3";
ListAddString (listID, szString, BEFORE);
```





Delete elements from a list

Call the [ListDeleteString](#) function to delete the current string from a list. Or, call the [ListDeleteItem](#) function to delete the current number from a list. If there are no more elements to delete, these functions return the END_OF_LIST constant.

Note that since ListDeleteString and ListDeleteItem delete the current element, you must reset the current element to the element that you want deleted. After deletion, the next element in the list becomes the current element. You reset the element by any of the methods described in [Traversing lists](#).

The example below illustrates the use of several list functions, including ListDeleteString. The ListDeleteItem function is used in the same manner as that, except that the list is a number list and the variables are number variables.

```
// Create the empty list of strings.
listID = ListCreate (STRINGLIST);

if (listID = LIST_NULL) then // Test for a valid list.
    MessageBox ("List not created", SEVERE);
endif;

// Add some strings to the list.
szString = "String 1";
ListAddString (listID, szString, AFTER);
szString = "String 2";
ListAddString (listID, szString, AFTER);
szString = "String 3";
ListAddString (listID, szString, AFTER);
ListDeleteString (listID); // Delete the current string.

// Reset the current string in the list.
lResult = ListCurrentString (listID, svString);
// svString contains "String 2."
```

{button ,PI("",`comment')} [Feedback](#)



Change existing elements in a list

Call the ListSetCurrentString function to change the value of an element in a string list. Remember that only the current element may be changed, so be sure to make the string you want to update the current string in the list.

Call the ListSetCurrentItem function to change the value of an element in a number list. Again, the item you want to update must be the current element in the list.

The example below demonstrates calling ListSetCurrentItem to change the value of the current item in a number list. The ListSetCurrentString function works in the same manner, but with a string list and string variables.

```
// Create a list and verify its creation.
listID = ListCreate (NUMBERLIST);

if (listID = LIST_NULL) then
    MessageBox ("Unable to create list.", SEVERE);
    abort;
endif;

// Add items (1078 and 304) to the list.
nItem = 1078;
ListAddItem (listID, nItem, AFTER);
nItem = 304;
ListAddItem (listID, nItem, AFTER);
// Current item is the second item (304).

// Now set current item to new value (305).
nItem = 305;
ListSetCurrentItem (listID, nItem);
ListDestroy (listID);
```

{button ,PI("",`comment')} [Feedback](#)



Read a file into a list

Call the [ListReadFromFile](#) function to read an entire file into a string list. Each line in the file becomes an element in the list. The ListReadFromFile function provides an easy way to load a list, rather than building it one item at a time.

{button ,PI("",`comment')} [Feedback](#)



Write a list to a file

Call the ListWriteToFile function to write the contents of a string list to a file. Each element in the list becomes a line in the file.

The example script below reads the Autoexec.bat file into the listFile string list and then writes that string list to a file named Autoexec.bak. Notice the use of the VarSave function to save the existing values of and TARGETDIR so they can be temporarily reassigned. The original values of SRCDIR and TARGETDIR are restored at the end of the example using the VarRestore function.

```
listFile    = ListCreate (STRINGLIST);
szPath      = "C:\\";
szFileOld   = "C:\\Autoexec.bat";
szFileNew   = "C:\\Autoexec.bak";
VarSave (SRCTARGETDIR);
SRCDIR = szPath;
TARGETDIR = szPath;

if (ListReadFromFile (listFile, szFileOld) < 0) then
    MessageBox ("ListReadFromFile failed.", SEVERE);
endif;

ListWriteToFile (listFile, szFileNew);
VarRestore (SRCTARGETDIR);
```

{button ,PI("",`comment')} Feedback



Operators

An operator is a symbol used in an expression to perform a specific action with one or two operands. An operand may be a constant, variable or function result. The operators in InstallShield are very much like those in the C language. As with C language operators, certain InstallShield operators, such as + and ^, vary in function according to the data type of the operands specified in the expression.

{button ,AL(`Address operator;Arithmetic operators;Assignment operator;Bit operators;BYREF operator;Indirection operator;Logical operators;Member operator;Relational operators;String operators;Structure pointer operator',0,'')}
See also

{button ,PI("",`comment')} [Feedback](#)



Address operator (&)

The Address operator is a unary operator that can be used to obtain the memory address of any variable in your script. The operator itself should precede the variable name, with no intervening space. You can use the address operator to assign the address of a variable to a pointer variable or to pass the address of a variable as an argument in a function call. You can send the address to C programs and operate on them as you would any standard C pointer.

In the example below, the address of a data structure is assigned to a pointer variable:

```
typedef DIMENSIONS
begin
    SHORT sLength;
    SHORT sWidth;
end;

DIMENSIONS rectangle;
DIMENSIONS POINTER pointerObject;

program
    pointerObject = &rectangle;
    // . . .
endprogram
```



If you use the address operator with a local variable, be aware that the local variable exists only during the life of the function in which it is declared. After the function returns, the address of the local variable will no longer be valid.

{button ,AL(`Indirection operator;Pointers',0,',')} [See also](#)

{button ,PI("", `comment')} [Feedback](#)



Arithmetic operators (+, -, *, /)

Arithmetic operators perform mathematical operations such as addition and subtraction with operands. There are two types of mathematical operators, unary and binary. Unary operators perform an action with a single operand. Binary operators perform actions with two operands. In complex expressions—those that include two or more operations—the order of evaluation depends on operator precedence.

{button ,AL(`Unary Arithmetic Operators;Binary Arithmetic Operators;Arithmetic Operator Precedence',0,`,`)}See also

{button ,PI(`",`comment')} Feedback



Unary arithmetic operators

Unary operators are arithmetic operators that perform an action on a single operand. The InstallShield Script Compiler recognizes two unary operators, negative (-) and positive (+).

Negative

The negative unary operator reverses the sign of an expression from positive to negative or vice-versa. If $nSize = 5$, then $-nSize = -5$. If $nLuckyNum + nBadOmen = -13$, then $-(nLuckyNum + nBadOmen) = 13$. The net result of using the negative unary operator before an expression is the same as multiplying the expression by -1.

Positive

The positive unary operator has the same net result on an expression as multiplying that expression by 1. It does not change the sign of a negative number to positive.

{button ,AL(`Unary Arithmetic Operators;Binary Arithmetic Operators;Arithmetic Operator Precedence',0,`,`)}[See also](#)

{button ,PI(`",`comment')} [Feedback](#)



Binary arithmetic operators

The InstallShield Script Compiler recognizes the binary arithmetic operators listed in the following table.

Symbol	Operation	Example	Description
+	Addition	$x + y$	Adds two operands.
-	Subtraction	$x - y$	Subtracts the second operand from the first operand.
*	Multiplication	$x * y$	Multiplies two operands.
/	Division	x / y	Divides the first operand by the second operand.



You should include a space both before and after an arithmetic operator to make your script more readable and to maintain a consistent look to your code.

{button ,AL(^Unary Arithmetic Operators;Binary Arithmetic Operators;Arithmetic Operator Precedence',0,','')}[See also](#)

{button ,PI('','comment')} [Feedback](#)



Arithmetic operator precedence

When the InstallShield Script Compiler encounters a complex expression—one that includes two or more simple expressions—it evaluates those expressions one at a time. The order in which expressions are evaluated is determined by operator *precedence*. The InstallShield Script Compiler evaluates arithmetic operators using the same order of precedence that the C language uses:

1. Negative (-) unary.
2. Multiplication and division.
3. Addition and subtraction.

If an expression contains two or more operators with the same precedence level, the operator to the left is processed first. For example, in the expression $15 / 3 * 7$, the InstallShield Script Compiler first performs the division ($15 / 3$), then multiplies the result by 7.

When a lower order precedence operation must be performed first, it should be surrounded by parentheses. For example, if the addition must be performed before the multiplication in the expression $30 / 3 + 7$, place parentheses around $3 + 7$. The parentheses in the expression $30 / (3 + 7)$ cause the compiler to add 3 and 7 first, yielding 10, and then divide 30 by 10 for a final result of 3.

You can nest parentheses within an expression. InstallShield allocates 20 temporary locations for calculating nested expressions. Therefore, you can nest 19 levels of operations within parentheses. The InstallShield Script Compiler performs the innermost operation first and works its way outward.

For example, in the expression $36 - (3 * (2 + 6 - 4))$, the InstallShield Script Compiler first performs the operation $2 + 6$, which yields 8, then subtracts 4 from 8, yielding 4, then multiplies 3 by 4, yielding 12, and finally subtracts 12 from 36, yielding 24. Note that within the inner parentheses, InstallShield performed the addition operation first because it was the further left of two operators with equal precedence.

{button ,AL('Unary Arithmetic Operators;Binary Arithmetic Operators;Arithmetic Operator Precedence',0,'')}See also

{button ,PI('','comment')} Feedback



Assignment operator (=)

Use the assignment operator (=) to copy a constant, literal, variable, expression result, or function result to a variable of the same type, as shown in the example code fragment below:

```

STRING szName;
LONG   nValue;
BOOL   bDone;
HWND   hInstance;
INT    iStyle;
LIST   LISTINFO;

program
    szName = "InstallShield";
    nValue = 15;
    bDone = FALSE;
    hInstance = 0;
    iStyle = DLG_MSG_STANDARD|DLG_CENTERED;
    LISTINFO = ListCreate(STRINGLIST);

```

InstallShield automatically sizes string variables that are declared without an explicit length, as in the example above. By default, InstallShield autosizes the string variable to 256 bytes. If you assign a string to that variable longer than 256 bytes (including the null terminator), InstallShield will increase the amount of space in memory reserved for that string variable (up to 512 bytes under Windows 3.1 and up to 1,024 bytes under Windows 95 and NT).

If you declare a string variable with an explicit length, you must make it long enough to receive the string you assign to it. In the example below, the string literal contains 51 characters. Therefore, both szStringVarA and szStringVarB must have a declared length of at least 52, which is just large enough to accommodate the string itself and the null terminator that is added automatically to the end of the string.

```

STRING szStringVarA[52], szStringVarB[52]
program
    szStringVarA = "This is a sample string that is 51 characters long.";
    szStringVarB = szStringVarA;

```



Unlike C++, InstallScript does not support multiple assignment operations in a single statement. In InstallScript, the statement `a = b = c` is equivalent to the C++ statement `a = b == c`. That is, the first operator is interpreted as an assignment operator; the second is interpreted as a relational operator. If `b` is equal to `c`, the value 1 (TRUE) is assigned to `a`; if `b` is not equal to `c`, the value 0 (FALSE) is assigned to `a`.

{button ,AL('Relational operators',0,'')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



BYREF operator

By default, the parameters of a user-defined function are passed by value; that is, a copy of the data specified by each parameter is passed to the function. Because the function operates on a copy, the original data cannot be changed by the function. For that reason, variables, constants and literals can all be passed by value.

When you want a user-defined function to operate directly on a variable it receives in a parameter rather than on a copy of that variable, you must specify the BYREF operator with the parameter type declaration in the function prototype, as shown below:

```
prototype StrInvert( BYREF STRING );
```

The BYREF operator indicates that the parameter is to be passed by reference; that is, the actual variable is to be passed to the function, and any changes made to that variable will be visible to the caller when the function returns. Parameters that are passed by reference are often referred to as variable parameters because they require a variable. Constants and literals cannot be passed by reference.

Note that when a user-defined function has more than one parameter, the BYREF operator must be specified with each variable parameter. In the example below, the first and third parameters are passed by reference; the second parameter is passed by value:

```
prototype StrChangeChar( BYREF STRING, CHAR, BYREF BOOL);
```

not in visual c++ edition

You can also use the keyword BYREF to declare variable parameters for DLL functions. However, it is not necessary to use the BYREF operator when prototyping string parameters because all string parameters are *always* passed to DLL functions by reference.



User-defined structure members cannot be passed by reference; attempting to do so will result in a compiler error. Instead you must pass a pointer to the entire structure by reference and then use the structure pointer operator (->) to access or modify the data elements of the structure.

{button ,AL('Declaring functions;Returning a value from a function;Calling a function;Calling a DLL function',0,',')}
See also

{button ,PI('','comment')} Feedback



Relational operators (<, >, =, <=, >=, !=)

Relational operators compare one expression to another within the context of a conditional statement, such as an if or while statement. For example, the following statement asks "Is x greater than 20?"

```
if (x > 20) then
```

The answer to the questions can be only TRUE (1) or FALSE (0). The InstallShield Script Compiler recognizes the relational operators listed in the table below:

Symbol	Operation	Example	Description
=	Equal	x = y	True if x is equal to y.
>	Greater than	x > y	True if x is greater than y.
<	Less than	x < y	True if x is less than than y.
>=	Greater than or equal to	x >= y	True if x is greater than or equal to y.
<=	Less than or equal to	x <= y	True if x is less than or equal to y.
!=	Not equal to	x != y	True if x is not equal to y.

When you use a relational operator in an if..else statement, the program will follow one of two actions:

- n If the expression is TRUE, statements that follow the if *are* executed. Statements that follow the else *are not* executed.
- n If the expression is FALSE, statements that follow the if *are not* executed. Statements that follow the else *are* executed.

Relational operators have an overall lower precedence than arithmetic operators. Among just relational operators, less than, less than or equal to, greater than, and greater than or equal to have precedence over equal and not equal.



Unlike C, which uses == to test for equality, InstallScript's assignment operator and relational operator use the same symbol (=).



You cannot use assignment and relational operators in the same conditional expression. For example, the following will fail:

```
if ((listID = ListCreate (NUMBERLIST)) =
LIST_NULL) then
. . .
endif;
```

{button ,AL('String comparisons;Relational Operator Precedence',0,'')}[See also](#)

{button ,PI('','comment')} [Feedback](#)



String comparisons

When InstallShield compares two strings, it starts by comparing the initial character in the first string with the initial character in the second string. If those characters are equal, InstallShield then compares the characters in the next position of each string. If those characters are also equal, it moves on to the characters in the next position, continuing in sequence until it encounters one of the following conditions:

1. Two characters in the same relative position in the two strings do not match. In this case, InstallShield bases its resolution on the comparison of those two characters. If the character in string one has a greater value than the character in the corresponding position of string two, then string one is greater; otherwise string two is greater.
2. The end of one string is encountered without finding unequal characters in corresponding positions. In this case, the strings are of unequal length and therefore they are not equal.
3. The end of both strings is reached without finding unequal characters in corresponding positions. In this case, the strings are equal in length and all characters match; therefore the strings are equal.

Consider the following example:

```
STRING svString1, svString2;
program
  svString1 = 'trusting';
  svString2 = 'TRUTHFUL';
  if svString1 = svString2 then
    MessageBox("Equal", INFORMATION);
  else
    MessageBox("Not Equal", INFORMATION);
  endif;
endprogram
```

String comparisons are not case sensitive. Because an uppercase character is equal to its lowercase counterpart, InstallShield finds that the first three characters of "trusting" and the first three characters of "TRUTHFUL" are equivalent. The comparison ends with the test of the characters in the fourth position of each string. Since 's' is lower than 't' in the ASCII character table, svString1 does not equal svString2. The remaining characters are not compared and the else branch is executed.



The value of each character is based on its ASCII value. For information about the ASCII values of specific characters and symbols, refer to any basic programming manual.



Relational operator precedence

Relational operators have an overall lower precedence than arithmetic operators. This means that InstallShield performs all arithmetic operations before beginning to evaluate logical operations. Logical expressions are evaluated from left to right, unless the order of operations is modified by parentheses. Among relational operators the order of precedence is as follows:

- n First, less than (<), less than or equal to (<=), greater than (>), and greater than or equal to (>=).
- n Then, equal to (=) and not equal to (!=).

Therefore, when combining arithmetic and relational expressions, the InstallShield Script Compiler evaluates precedence in the following order:

1. Negative (minus) unary.
2. Multiplication and division.
3. Addition and subtraction.
4. Less than (<), less than or equal to (<=), greater than (>), and greater than or equal to (>=).
5. Equal to (=) and not equal to (!=).

In the expression $6 + 7 > y$, the InstallShield Script Compiler adds 6 and 7 together and then compares the result (13) to y . To change the order of precedence, use parentheses. For example, in the expression $6 + (7 > y)$, the InstallShield Script Compiler first determines if 7 is greater than y . If it is, then 1 (the numeric value of TRUE) is added to 6. If it is not, then 0 (the numeric value of FALSE) is added to 6.

When a statement contains arithmetic, relational, and logical operators, the InstallShield Script Compiler evaluates precedence in the following order:

1. Negative (minus) unary has first precedence.
2. Multiplication and division have second precedence.
3. Addition and subtraction have third precedence.
4. NOT (!) has fourth precedence.
5. Less than (<), less than or equal to (<=), greater than (>), and greater than or equal to (>=) have fifth precedence.
6. Equal to (=) and not equal to (!=) have sixth precedence.
7. AND (&&) has seventh precedence.
8. OR (||) has eighth precedence.

Use logical operators in if and while statements the same way you used relational operators. The example below adds `nExampleSize` and `nHelpSize` if `bInstallExample` and `bInstallHelp` are TRUE.

```
if (bInstallExample && bInstallHelp) then
    nTotalSize = nExampleSize + nHelpSize;
endif;
```

The next example sets the `bPublicFile` to TRUE if *either* `bInstallProgram1` or `bInstallProgram2` is TRUE.

```
if (bInstallProgram1 || bInstallProgram2) then
    bPublicFile = TRUE;
endif;
```



You cannot use the `&&` or `||` operators within an argument to a function. Instead, follow the above example and assign the value of the logical expression to a Boolean variable and then call the function with the variable as an argument.



Logical operators (&&, ||, !)

Logical operators allow you to ask more than one relational question at the same time. For example, using a logical operator you can ask if *y* is greater than 7 and *szFilePath* contains "C:\\programf\\Ishield". Logical operators return either a TRUE (1) or FALSE (0) value. Like relational operators, they are used most often in if and while statements.

The InstallShield Script Compiler recognizes the logical operators listed in the table below:

Symbol	Operation	Example	Description
&&	AND	<code>exp1 && exp2</code>	True only if both <code>exp1</code> and <code>exp2</code> are true; otherwise, false.
	OR	<code>exp1 exp2</code>	True if either <code>exp1</code> or <code>exp2</code> is true; false (0) only if both are false.
!	NOT	<code>!exp1</code>	False if <code>exp1</code> is true; true if <code>exp1</code> is false.

Logical operators have a lower precedence than arithmetic or relational operators. Among logical operators, the AND operator has higher precedence than the OR operator.



Unlike C++, InstallShield performs complete boolean evaluations of logical expressions. Consider the following if statement:

```
if (iVar = 10) && (MyFunction( ) = 0) then
    MessageBox("That is so true.", INFORMATION);
endif;
```

`MyFunction` will be called even if the expression to the left of the logical operator is false. To obtain the effect of short circuit boolean evaluations (in which the expression to the right of the `&&` is resolved only if the expression to the left of the `&&` is true), use a nested if statement, as shown below:

```
if (iVar = 10) then
    if (MyFunction( ) = 0) then
        MessageBox("That is so true.", INFORMATION);
    endif;
endif;
```

{button ,AL('Relational operators',0,'')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



String operators (^, +, %)

The string operators allow you to directly manipulate strings without the use of functions. Note that string operators are not case sensitive. The InstallShield Script Compiler supports the following string operations:

Append to Path (^)

Adds additional paths to a path or a filename.

Concatenate (+)

Appends one string to the end of another string.

Find String (%)

Locates a substring in another string.



Do not use parentheses to enclose expressions on either side of a string operator. For example, avoid statements like the following:

```
szPath = szTestPath ^ (AUTOFILE + ".BAT");
```

Instead, create expressions without parentheses for use with string operators, as shown below:

```
szFile = AUTOFILE + ".BAT";  
szPath = szTestPath ^ szFile;
```

{button ,PI("",`comment')} [Feedback](#)



Append to path (^)

Use the append to path (^) operator when you are combining two paths or a path and a filename. The operator automatically checks to see if you have added the proper number of backslashes when appending a filename or a subdirectory to a path. If you type:

```
szStringVar = "C:\\MYPATH\\" ^ "YOURPATH\\FILENAME";
```

the output from szStringVar will be:

```
C:\MYPATH\YOURPATH\FILENAME
```

If you forgot to add the backslashes after MYPATH, as shown below:

```
szStringVar = "C:\\MYPATH" ^ "YOURPATH\\FILENAME";
```

the result of the operation is still valid. (The InstallShield ^ operator adds the backslashes for you.)

{button ,AL(^ Append to path;Concatenate;Find string',0,','')} [See also](#)

{button ,PI("",`comment')} [Feedback](#)



Concatenate (+)

Use the concatenate string operator (+) to join one string to the end of another. Concatenating two strings results in a new, third string. In the following example, two string constants are concatenated and the resulting string value is assigned to the string variable szThirdString; after the statement has been executed, the value of szThirdString is "First string Second string".

```
szThirdString = "First string " + "Second string";
```

Operands in a concatenation expression may be string literals, string constants, or string variables. In the example below, a string constant and a string literal are concatenated; the resulting string value is assigned to szThirdString.

```
#define FIRST_STRING "This is the first string"
STRING szThirdString;

program
    szThirdString = FIRST_STRING + "Second string ";
endprogram
```



When assigning the result of a concatenation expression to a string variable, you must ensure that the concatenated string is not too long for the string variable to which it is assigned. A statement that assigns a string to a variable of insufficient size will produce a runtime error ("Error 401").

{button ,AL(`Append to path;Concatenate;Find string',0,`,`)} [See also](#)

{button ,PI(`",`comment')} [Feedback](#)



Find string (%)

Use the String Find operator (%) to determine if one string is a substring within another string. The following example tests szStringVarA to determine if it contains the string "sample." If it does, then MessageBox is called to display a message.

```
szStringVarA = "This is a sample string.";
if (szStringVarA % "sample") then
    MessageBox("Operation complete", INFORMATION);
endif;
```

The character comparison is not case sensitive. In the following example, the message box will be displayed:

```
szStringVarA = "This is a sample string.";
if (szStringVarA % "SAMPLE") then
    MessageBox("Operation complete", INFORMATION);
endif;
```



The InstallScript function [StrFind](#) also determines if a substring is contained within another string. If the substring is found, StrFind returns its position within the string.

{button ,AL(`Append to path;Concatenate;Find string',0,`,`)} [See also](#)

{button ,PI(`",`comment')} [Feedback](#)



Bit operators (&, |, ~, ^, <<, >>)

Bit, or bitwise, operators allow you to manipulate individual bits in a numeric variable. In order to use the operators effectively, you need to be familiar with binary notation. This topic gives you an overview of binary operators, but it does not try to teach you binary notation.

Bit operators work like logical operators, with one exception: logical operators work with expressions, bit operators work with bits. The InstallShield Script Compiler recognizes the bitwise operators listed in the table below:

Symbol	Operator	Explanation
&	BitAND	BitAND sets a bit in the result to 1 (TRUE) only if the bits in both operands are 1s; otherwise, the result is 0 (FALSE).
	BitOR	Bit inclusive OR sets a bit in the result to 0 only if the bits in the operands are 0; otherwise, the result is 1.
^	BitXOR	Bit exclusive OR sets a bit in the result to 1 if the corresponding bits in the operands are different (one 1, the other 0). Otherwise, the result is 0.
~	BitNOT	BitNOT is a unary operator that reverses every bit in the operand, changing every 1 to a 0 and vice versa, as in the following example: ~00001100 The result is 11110011.
<<	Shift left	Moves bits a specified number of bits to the left. For example, the expression shown below moves bits three spaces to the left: 00001100 << 3 The result is 01100000.
>>	Shift right	Moves bits a specified number of bits to the right. For example, the expression shown below moves bits two spaces to the right: 00001100 >> 2 The result is 00000011.

Shift operations work the same way in InstallShield that they do in the C language. When you shift by two bits to the right (>>2), the two bit values furthest to the right are lost. The other bit values are shifted to the right two places, and 0 values are shifted in to the empty bits.

You can use shift operations to multiply and divide a value by a power of 2. Left-shifting an integer by n has the same effect as multiplying the number by 2 to the power of n. Right-shifting an integer by n has the same effect as dividing the number by 2 to the power of n.

{button ,PI("",`comment')} [Feedback](#)



Indirection operator (*)

The indirection operator is a unary operator that can be used to obtain the value stored at the memory location referenced by a pointer variable. The indirection operator must precede the pointer variable name, with no intervening space.

In the following example, `nvalue` is a number and `pnumber` is a pointer to a number. The assignment statement is used to copy to `nvalue` the number being pointed to by `pnumber`.

```
nvalue = *pnumber;
```

The indirection operator can also be used to pass a value to a function that takes a number value as a parameter, as in the following example:

```
somefunction(*pnumber);
```

The following limitations apply to the indirection operator:

- n The indirection operator can be used with number pointers only.
- n The indirection operator cannot be used to assign a value to a memory location.
- n The indirection operator cannot be used as an argument to a function whose parameter has been defined with the BYREF operator.
- n The indirection operator cannot be used to declare a pointer.
- n The indirection operator cannot be used to declare a variable parameter in a function declaration.

{button ,AL(`Address operator;Pointers',0,`,`')}

[See also](#)

{button ,PI("",`comment')} [Feedback](#)



Member operator (.)

Use the member operator to reference individual elements in a structure variable. The member operator must appear between the structure variable name and the element name, with no intervening space. In the example below, a literal value is assigned to each element in a structure variable.

```
typedef SETTINGSREC
begin
    BOOL bSwitchOn;
    STRING szMssg[255];
    INT nVal;
end;

SETTINGSREC settings;

program
    settings.bSwitchOn = FALSE;
    settings.szMssg = "Off";
    settings.nVal = 0;
```

{button ,AL(`Data structures',0,`,`)} [See also](#)

{button ,PI("",`comment')} [Feedback](#)



Structure pointer operator (->)

Use the structure pointer operator to reference individual elements in a structure by means of a pointer variable. The structure pointer operator must appear between the pointer variable name and the element name, with no intervening space. In the example below, a literal value is assigned to each element in a structure.

```
typedef  DIMENSIONS
begin
    SHORT sLength;
    SHORT sWidth;
end;

DIMENSIONS Table;

NUMBER  nvNumValue;
DIMENSIONS POINTER  pointerObject;

program
    pointerObject = &Table;
    pointerObject->sLength = 500;
    pointerObject->sWidth = 750;
```



You can use only one structure pointer in an expression. If structure A contains a member (Bptr) that is a pointer to structure B, which contains a member (Cptr) that is a pointer to structure C, you cannot reference a member of C from A. The expression A.Bptr->Cptr->Cmember is invalid in InstallScript.

{button ,AL(`Data structures;Pointers',0,`,`)} [See also](#)

{button ,PI(`",`comment')} [Feedback](#)



Preprocessor directives

Preprocessor directives are instructions to the InstallShield compiler; they are executed as the script is being compiled. Preprocessor directives can instruct the compiler to include other source files in the compilation, to define constants, to include or exclude statements based on compile-time conditions, and to display a user-defined error message. Although InstallShield directives are very similar to directives in the C language, they are not exactly alike.

Preprocessor directives begin with a number sign (#) and can be inserted anywhere in a script. Each directive must appear on a line by itself and must *not* be terminated with a semicolon. Note that the expressions used in conditional directives can include constants that have been defined with the directive #define; they *cannot* include InstallScript variables.

The following Boolean operators are supported in #if, #ifdef, #ifndef, and #elif statements:

- n Logical OR (||)
- n Logical AND (&&)
- n Relational (=, !=, >, >=, <, <=)

InstallShield supports the following preprocessor directives:

#define

Creates a symbolic constant.

#elif

Combines #else and #if into one statement.

#else

Indicates alternatives if the test fails.

#endif

Ends a preprocessor conditional directive (#if, #ifdef, #ifndef).

#error

Displays a user-defined error message.

#if

Compiles if the conditional statement is true.

#ifdef

Compiles if a numeric constant is defined.

#ifndef

Compiles if a numeric constant is not defined.

#include

Includes the contents of another file.

#undef

Undefines a constant that was defined previously with #define.



A preprocessor directive does not end with a semicolon.
Preprocessor directives cannot line-wrap. They must be less than 250 characters in length.

{button ,AL('Defining constants from the command Line;Using preprocessor statements to debug the script',0,'')}

See also

{button ,PI('','comment')} Feedback



#define

Use `#define` to define a number or string constant. Once you define a constant and assign it a value, InstallShield replaces the constant with that value wherever it appears. For example, the following `#define` statement sets the value of `MAX_SIZE` to 145:

```
#define MAX_SIZE 145
```

The following example declares a string constant with the `#define` directive:

```
#define STR_MESSAGE "This is a message."
```

Once you have defined `STR_MESSAGE`, you can use it anywhere in the script. A string message you want displayed in `SprintfBox` or `MessageBox` cannot be longer than 255 characters. If you want to display more than 255 characters, split your string into two or more parts before displaying it. (The 255-character limit includes spaces, escape sequences, and other special characters.)



Another way to define constants is in the Preprocessor Defines field on the Compile tab of the [Setup Settings dialog box](#). If you add or change a preprocessor define in the Setup Settings dialog box, you must recompile your setup before the changes will take effect.

There are a few restrictions regarding the `#define` directive that you should keep in mind:

- n InstallShield supports the use of `#define` to define only macros that involve the simple lexical substitution of a number or a string. You cannot define macros with expressions using multiple terms or operators.
- n Constants you declare with a `#define` statement cannot begin with numbers.
- n Many InstallShield functions use [predefined constants](#). If you try to define one of the predefined constants, the InstallShield Script Compiler will generate an error message.
- n InstallShield assigns a value of zero (0) to an undefined constant.

{button ,AL('Defining Constants from the Command Line',0,',';')}

[See also](#)

{button ,PI('','comment')} [Feedback](#)



Defining constants from the command line

You can define InstallShield Script Compiler constants from the command line, instead of in the script, using the /D option. For example, the command line statement `COMPILE /D MAX_BUF=150 /D MAX_LENGTH=255 SETUP.RUL` is equivalent to the define statements shown below:

```
#define MAX_BUF 150
#define MAX_LENGTH 255
```

The following restrictions apply when defining constants from the command line:

- n You can define only numeric constants from the command line.
- n If you define a constant at the command line that you defined with a #define statement in the script, you will generate an error.
- n If you define a constant at the command line that you defined as a variable in the script, the value of the constant will be lost at run time.

{button ,PI("",`comment')} [Feedback](#)



#error

Use the `#error` directive to abort compilation and display a user-defined error message. The message to display must be entered immediately after `#error` and be separated from the directive by at least one space.

In the example below, the constant `PRODUCTID` must equal 1 or 2. If the value of `PRODUCTID` is not within that range, the constant `PRODUCTNAME` is not defined and a user-defined error message is displayed.

```
#define PRODUCTID 1
#if (PRODUCTID = 1)
    #define PRODUCTNAME "Lite"
#elif (PRODUCTID = 2)
    #define PRODUCTNAME "Professional"
#endif
#ifdef PRODUCTNAME
    #error PRODUCTID out of range.
#endif
```

{button ,PI("",`comment`)} [Feedback](#)



#if. .#else. .#endif

Use the `#if` statement to select which lines to compile. You can switch various sections of the installation on and off, making the script more flexible. The `#if` statement works in the same manner as the run-time if statement:

```
#if (A = 1)
// compile if A equals 1
. . .
#else
// compile
. . .
#endif
```

Note that the format of the `#if` statement is also identical to that of the run-time if statement; you must end the `#if` statement with the keyword `#endif`.

You can test only numeric constants with an `#if` or `#elif` statement.

{button ,PI("",`comment')} [Feedback](#)



#ifdef and #ifndef

Use the `#ifdef` statement when you want to compile a section only if a specified expression has been defined with `#define`. Use `#ifndef` when you want to compile a section only if a specified expression has not been defined. For example:

```
#ifdef A
    // Compile if A is defined.
    .
    .
    .
#endif
#ifndef A
    // Skip if A is defined; otherwise compile.
    .
    .
    .
#endif
```

You can also use `#ifdef` and `#ifndef` with `#else` and `#elif`:

```
#ifdef nFilePath
    //statements
#else
    //statements
#endif
```



Preprocessor defines can be entered in the Preprocessor Defines field on the Compile tab of the [Setup Settings dialog box](#). If you add or change a preprocessor define in the Setup Settings dialog box, you must recompile your setup before the changes will take effect.

There are certain restrictions you should keep in mind when using `#ifdef` and `#ifndef` statements:

- n You cannot place comments on the same line as `#ifdef` and `#ifndef` directives.
- n Do not place run-time flow-control statements (if, while, switch, etc.) inside `#ifdef` or `#ifndef` directives or use `#ifdef` or `#ifndef` directives inside run-time flow-control statements. They will not compile correctly.
- n Do not test a constant that has a value of 0 (zero) with an `#ifdef` or `#ifndef` statement.
- n You can test only numeric constants with an `#ifdef` or `#ifndef` statement.

{button ,PI("",`comment')} [Feedback](#)



#elif

The `#elif` compile-time statement is similar in function to the `elseif` run-time statement. It combines the `#if` and `#else` statements, allowing you to specify another condition. For example:

```
#if (A = 1)
    // compile if A equals 1
    . . .
#elif (A = 2)
    // compile if A equals 2
    . . .
#elif (B = 3)
    // compile if B equals 3
    . . .
#else
    // if none of the #elif conditions are true, compile
    // the following portion
    . . .
#endif
```

Note that if you use `#elif`, you still end the section with only one `#endif`.

{button ,PI("",`comment')} [Feedback](#)



Using preprocessor statements to debug the script

Use the `#define` and `#ifdef` statements to create an internal debugger in the script. Follow the steps below when you want to use the preprocessor statements to debug:

1. Wherever you want to insert a debug statement in the script, start with the following `#ifdef` directive:

```
#ifdef DEBUG
```

2. On the following line(s), type the debugging statement(s).

3. On a separate line after the debugging statement(s), type:

```
#endif
```

4. For debugging purposes, compile the script with the following command at the prompt:

```
C:\> Compile -DDEBUG=1 MYSCRIPT.RUL
```

5. In the final compile, use the following command at the command prompt:

```
C:\>Compile MYSCRIPT.RUL
```

Here is an example of a debugging section using an `#ifdef` statement:

```
#ifdef DEBUG
    if nResult < 0 then
        WriteLine (LogFileHandle, "PlaceBitmap failed")
    endif
#endif
```



Be careful that you do not change the logic of the script when using preprocessor statements to debug the code.

{button ,PI("",`comment')} [Feedback](#)



#include

Use the #include statement to include the contents of another script in the main installation script. When you use #include, the compiler treats the additional source script as if it were part of the main installation script. Additional scripts, or include files, may contain variable declarations, other compiler directives, and program statements.

For example, you can create a separate file that contains all the user-defined constant definitions and then insert it into the script file using the #include statement. If you need to redefine any of the constants at a later date, they are all in one central location.

Before you compile, place the #include script files in the same subdirectory as the installation script file. When compiling from the IDE, InstallShield searches for include files in the following order:

1. The project script file directory
2. The InstallShield include directory

If the include file is not in either of these locations, specify a fully qualified filename in the #include statement. When using the #include statement, specify a filename or path by enclosing the filename or path in double quotation marks ("filename").

When using #include directives in your script, keep the following points in mind:

- n InstallShield does not handle paths with more than 260 characters, including the filename.
- n The InstallShield preprocessor does not interpret a backslash character in an #include directive as a control character; when specifying a path, use a single backslash instead of double backslashes to separate folder names.
- n Do not include C language header files in the script. InstallShield's compiler does not recognize some of C's constructs. Create header files using InstallScript only.

The following example shows a section of an installation script that uses the #include statement to include the contents of Support.rul and other files. Each of the source scripts referenced by the #include statements has been written for a specific purpose and is then added to the script when the script is compiled:

```
//The following include file contains installation-specific routines.
#include "SUPPORT.RUL"
// Local include file containing variable and prototype declarations.
#include "DECLARE.RUL"
// Include scripts from the LIBRARY directory.
#include "..\LIBRARY\WINSUB.H"
#include "..\LIBRARY\SYSCHK.H"

// Include scripts from the DIALOGS directory.
#include "..\DIALOGS\WELCOME\WELCOME.H"
#include "..\DIALOGS\REGINS\REGINS.H"
#include "..\DIALOGS\ICONS\ICONS.H"
```

{button ,PI("",`comment')} [Feedback](#)



#undef

Use this directive to undefine a constant that was previously defined with #define. If specify a constant with this directive that was not previously defined with the #define directive, an error will occur when you attempt to compile your script. In the following example, two constants are first defined and then undefined:

```
#define MAX_BUF 150
#define MAX_LENGTH 255
// . . .
#undef MAX_BUF
#undef MAX_LENGTH
```

{button ,PI("",`comment')} [Feedback](#)



Setup scripts

InstallShield makes designing your setup easy with InstallScript, its simple but powerful programming language. InstallScript is similar to the C language. It has a defined format and regulated syntax. It uses certain data types, each with specific properties. It also allows you to create custom functions.

InstallScript, however, does not provide the full range of programming functionality that C does. InstallScript was designed to do one thing—create setups. And it does so better than any programming language in the world.

Regardless of your programming background, you can learn quickly to build your setup with InstallScript. InstallShield comes with over 250 built-in functions that do most of the work for you.

{button ,AL(` Structure of a setup script;Declarations;Program block;Function block;Building a function',0,`,`)}}See also

{button ,PI(`",`comment')}} Feedback



Structure of a setup script

Every setup script includes data declarations, a program block, and function block(s).

- n Declarations may appear ahead of the program block, between the program block and the function block, and between a function statement and the begin statement for that function
- n The program block must appear before the function blocks.
- n Function blocks must follow the program block.

The general outline of an InstallShield script is shown below:

```
// Constant definitions, global data declarations, and function declarations
// Program block
// Function block
```

{button ,AL(`Declarations;Program block;Function block;Building a function',0,`,`)}}

[See also](#)

{button ,PI("",`comment')} [Feedback](#)



Declarations

Every setup script begins with global data declarations. Here, you define constants and declare each of the global variables and user-defined functions that you will be using. Declarations instruct the InstallShield Script Compiler that the script will be using the listed items at a later time. Declarations also build an association between a function and its attributes or values. You do not need to declare any of the built-in functions, since the InstallShield Script Compiler already recognizes the function names.

Below are some examples of constant definitions, data declarations, and function declarations:

```
// Constant definitions
#define PRODUCT "InstallShield"
#define LIMIT 100

// Variable declarations
CHAR cVal;
NUMBER nVal;
STRING szName;

// Function declarations
prototype DisplayMsg (NUMBER, STRING);
prototype GetName (BYREF STRING);
```

{button ,AL(`Constant data;Data structures;Data types;Global vs local variables;Hungarian notation;Declaring functions;BYREF operator',0,`,`)} [See also](#)

{button ,PI(`",`comment')} [Feedback](#)



Program block

A setup script must contain one and only one program block. This section of the script follows constant definitions, global data declarations and function declarations. All user-defined functions follow the program block.

The program block begins with the keyword `program` and ends with the keyword `endprogram`, as shown below:

```
program
  // Program statements appear here.
endprogram
```



You cannot immediately precede the `endprogram` statement with a label.

{button ,AL(`Declarations;Structure of a setup script;Function block;Building a function',0,`,`) }

[See also](#)

{button ,PI(`",`comment')} [Feedback](#)



Function block

All functions that have been declared with a prototype statement must be defined in the function block, which follows the keyword `endprogram` in a setup script. Additional global data declarations may be made in the function block, either between the `endprogram` statement and the first function declaration or between function declarations. However, data declared in the function block is visible only to functions that are defined after the data declaration.

{button ,AL(`Declarations;Functions;Structure of a setup script;Program block;Building a function',0,`,`)} [See](#)
[also](#)

{button ,PI(`",`comment')} [Feedback](#)



Punctuation rules

Like any programming language, InstallScript has syntax rules that regulate its usage. If you are familiar with the C language, the basic syntax of InstallScript will seem quite familiar. Even for those who do not know any programming languages, the syntax is not difficult to pick up.

The following punctuation reminders apply to all sections of the script:

- n Most statements end with a semicolon (;). This includes many one-word statements, such as end;, exit;, return;, and so on.
- n Preprocessor statements, such as #define and #include, never end with a semicolon.
- n The keywords program, endprogram, and begin are placed on separate lines by themselves and receive no punctuation. The function line that begins each function block receives no punctuation, either.
- n End a label, such as start: or starthere:, with a colon (:).
- n Enclose parameter lists within parentheses. When you have more than one parameter, separate them with commas.

{button ,PI("",`comment')} [Feedback](#)



Embedding quotation marks

You can insert double quotation marks as part of a string literal using one of two methods. If you begin the string literal with double quotation marks, you must use the `\` escape character to embed double quotation marks. You can, however, begin the literal with a single quotation mark and then type the double quotation mark:

```
//These two statements will both yield embedded double quotation marks
szQuote1 = "Who said, \"Quitters never win\"?";
szQuote2 = 'The same guy who said, "I quit. "';
```

To embed a single quotation mark, either use the `\` escape sequence or open the string literal with double quotation marks:

```
//These two statements will both yield embedded single quotation marks
szQuote1 = 'Who said, \"Nice guys finish last\"?';
szQuote2 = "The same guy who said, 'I win.'";
```



Your setup scripts must use the standard quotation marks (`"` and `'`) found to the right of the semicolon (`;`) key on the standard U.S. keyboard. Do not use open or closed typographer's quotation marks.

{button ,PI("",`comment')} [Feedback](#)



Using white space

Like C and other programming languages, InstallScript does not recognize white space (spaces and tabs, carriage returns) except in a string literal. We recommend that you use white space to make your script easier to follow. For example, the following section of code is dense and difficult to decipher:

```
#define DISK_DRIVE "C:\\\"
STRING szDrive, svString;
NUMBER nSpace, nResult;
program
szDrive = DISK_DRIVE;
nSpace = GetDiskSpace(szDrive);
nResult = NumToStr(svString, nSpace);
if (nResult < 0) then
MessageBox("NumToStr failed.", SEVERE);
abort;
endif;
sprintfBox(INFORMATION, "Info", "Disk Space: %s", svString);
endprogram
```

Adding white space with indentation makes the same code much easier to read:

```
#define    DISK_DRIVE    "C:\\\"
    STRING    szDrive, svString;
    NUMBER    nSpace, nResult;

program
    szDrive = DISK_DRIVE;
    nSpace  = GetDiskSpace(szDrive);
    nResult = NumToStr(svString, nSpace);
    if (nResult < 0) then
        MessageBox("NumToStr failed.", SEVERE);
        abort;
    endif;
    sprintfBox(INFORMATION, "Info",
        "Disk Space: %s", svString);

endprogram
```

{button ,PI("",`comment')} [Feedback](#)



Writing comments

InstallScript gives you two ways to create comments in a script. You can use either method to add explanatory text to your script or to exclude ("comment out") certain parts of your script for testing and debugging purposes.

One way to create a comment is to enclose a block of text between the character pairs `/*` and `*/`. This method makes it easy to write a comment over multiple lines:

```
/* This is a line of sample code that shows you  
   how to use the InstallShield function PlaceBitmap. */
```

The second way is to insert the characters `//` into a line. The compiler ignores everything to the right of the double slashes on that line only.

```
// This is a line of sample code showing the  
// InstallShield function PlaceBitmap.
```

You can begin comments anywhere in a script—with one exception. Comments may not be placed on the same line as an `#ifdef` or `#ifndef` statement. Simply write comments before or after these statements. Otherwise, the compiler will return an error.

{button ,PI("",`comment')} [Feedback](#)



System variables

System variables are predefined string variables that contain information such as the source path, the target path, the Windows folder, and the Windows\System folder. You cannot declare these variables in your script. InstallShield automatically initializes system variables when the setup process begins. Once the system variable is initialized, you can access it and change its value, although, in general, we recommend that you do not.

The following table lists and describes InstallShield's system variables:

{button ,JI("LangRef>main","Batch_Install")}	<u>BATCH_INSTALL</u>
Indicates whether or not a locked file was encountered while transferring files with the LOCKEDFILE or SHAREDFILE option.	
{button ,JI("LangRef>main","CMDLINE")}	<u>CMDLINE</u>
Specifies a command line parameter to be passed to Setup.exe.	
{button ,JI("LangRef>main","COMMONFILES")}	<u>COMMONFILES</u>
Stores the fully qualified path to the common files folder.	
{button ,JI("LangRef>main","ERRORFILENAME")}	<u>ERRORFILENAME</u>
Stores the name of the file at which an error occurred.	
{button ,JI("LangRef>main","FOLDER_DESKTOP")}	<u>FOLDER_DESKTOP</u>
Stores the fully qualified path to the Windows 95 or Windows NT 4.0 Desktop folder.	
{button ,JI("LangRef>main","FOLDER_PROGRAMS")}	<u>FOLDER_PROGRAMS</u>
Stores the fully qualified path to the Windows 95 or Windows NT 4.0 Start Menu\Programs folder.	
{button ,JI("LangRef>main","FOLDER_STARTMENU")}	<u>FOLDER_STARTMENU</u>
Stores the fully qualified path to the Windows 95 or Windows NT 4.0 Start Menu folder.	
{button ,JI("LangRef>main","FOLDER_STARTUP")}	<u>FOLDER_STARTUP</u>
Stores the fully qualified path to the Windows 95 or Windows NT 4.0 StartUp folder.	
{button ,JI("LangRef>main","INFOFILENAME")}	<u>INFOFILENAME</u>
Specifies the name of a backup file created by InstallShield.	
{button ,JI("LangRef>main","ISRES")}	<u>ISRES</u>
Contains the fully qualified filename of the _isres.dll loaded for the current instance of InstallShield.	
{button ,JI("LangRef>main","ISUSER")}	<u>ISUSER</u>
Contains the fully qualified filename of the _isuser.dll loaded for the current instance of InstallShield.	
{button ,JI("LangRef>main","ISVERSION")}	<u>ISVERSION</u>
Contains the version of Setup.exe that is running.	
{button ,JI("LangRef>main","MEDIA")}	<u>MEDIA</u>
Stores the name of the current media library.	
{button ,JI("LangRef>main","MODE")}	<u>MODE</u>
Indicates whether the program Setup is running in normal or silent mode.	
{button ,JI("LangRef>main","PROGRAMFILES")}	<u>PROGRAMFILES</u>
Stores the fully qualified path to the Windows 95 or Windows NT 4.0 Program files folder.	
{button ,JI("LangRef>main","SELECTED_LANGUAGE")}	<u>SELECTED_LANGUAGE</u>
Contains the numeric language code of the language the setup is using.	
{button ,JI("LangRef>main","SHELL_OBJECT_FOLDER")}	<u>SHELL_OBJECT_FOLDER</u>
Contains the fully qualified name of the folder that was selected in SelectFolder; this variable can be referenced by the Displayed Folder/Group Name property.	
{button ,JI("LangRef>main","SRCDIR")}	<u>SRCDIR</u>
Contains the fully qualified path to the source folder.	
{button ,JI("LangRef>main","SRCDISK")}	<u>SRCDISK</u>
Contains the name of the drive with the source disk.	
{button ,JI("LangRef>main","SUPPORTDIR")}	<u>SUPPORTDIR</u>
Contains the fully qualified path to the folder used for temporary files.	
{button ,JI("LangRef>main","TARGETDIR")}	<u>TARGETDIR</u>

Contains the fully qualified path to the target folder on the hard disk.

{button ,JI("LangRef>main","TARGETDISK");TARGETDISK

Contains name of the target destination disk.

{button ,JI("LangRef>main","UNINST"); UNINST

Contains the fully qualified filename of the unInstallShield program.

{button ,JI("LangRef>main","WINDIR"); WINDIR

Contains the fully qualified path to the folder that contains the main operating environment.

{button ,JI("LangRef>main","WINDISK"); WINDISK

Contains the name of the disk that contains the main operating environment.

{button ,JI("LangRef>main","WINSYSDIR"); WINSYSDIR

Contains fully qualified path to the Windows\System folder.

{button ,JI("LangRef>main","WINSYSDISK");WINSYSDISK

Contains the name of the disk that contains the Windows\System folder.

{button ,PI("",`comment`); Feedback



BATCH_INSTALL

This system variable is set to TRUE if InstallShield encounters a locked file while transferring files with the LOCKEDFILE or SHAREDFILE option. If BATCH_INSTALL is set to TRUE, you must call [CommitSharedFiles](#) or [SdFinishReboot](#) so that the system will update the files when it is next rebooted. If BATCH_INSTALL is set to FALSE, no locked files were found and the setup process can terminate normally.

{button ,PI("",`comment')} [Feedback](#)



CMDLINE

The user may specify a Setup.exe command line parameter at startup. InstallShield places the command line parameter string in the system variable CMDLINE. Note, however, that IS-defined command line switches are not passed to CMDLINE. For more information about Setup.exe command line switches, click [here](#).

{button ,PI("",`comment')} [Feedback](#)



COMMONFILES

This system variable contains the fully qualified name of the folder defined by Windows to store files shared by applications that are installed on the system. In English Windows, that folder is named "Common Files" and is located in the program files folder. (In other language versions of Windows, the common files folder name is localized appropriately by default.) The common files folders is the recommended default location for folders that are shared by applications.

The value of the InstallShield system variable COMMONFILES is obtained from the key "CommonFilesDir" in the Windows registry during setup initialization. Note that since this key is located under the key HKEY_LOCAL_MACHINE, it does not exist on 16-bit platforms; for that reason, 16-bit setups cannot determine this value and will instead set COMMONFILES to the folder \ProgramF\CommonF on the drive with the Windows folder. This same behavior occurs when a 16-bit setup is run on a 32-bit platform.

{button ,PI("",`comment')} Feedback



ERRORFILENAME

This system variable stores the name of the file that was involved in an error. For example, if an error occurs while copying a specific file with a built-in function, InstallShield will set ERRORFILENAME to the name of the file that caused the error. Not all file-operation functions use ERRORFILENAME.

{button ,AL(`Do',0,`,`')} [See also](#)

{button ,PI("",`comment')} [Feedback](#)



FOLDER_DESKTOP

This system variable stores the fully qualified path to the Desktop folder, which holds the program folders and items that are displayed on the user's desktop. It is valid only with 32-bit setup running under operating systems that use the explorer shell: Windows 95 and higher and Windows NT 4.0 and higher. In 16-bit setups and in 32-bit setups running under the Program Manager shell, this variable will contain a null string ("").

The value of FOLDER_DESKTOP changes as follows to ensure that groups and folders are created at the proper location on all explorer shell operating systems.

- n Under Windows NT 4.0, the location pointed to by FOLDER_DESKTOP changes when the default group/folder type is changed from Common to Personal or Personal to Common by calling ProgDefGroupType.
- n Under Windows 95, the location pointed to by FOLDER_DESKTOP does not change unless the current user logs out and a different user logs in. Note that this value will change only if user profiles have been enabled; otherwise the default FOLDER_DESKTOP location is used for all users.



Once user profiles have been enabled on Windows 95, FOLDER_DESKTOP cannot be set to the default location; it will always point to the current user's desktop folder location. For that reason, it is not possible to install common software on Windows 95 once profiles have been enabled. This is a limitation of Windows 95.

{button ,PI("",`comment')} Feedback



FOLDER_PROGRAMS

This system variable stores the fully qualified path to the Start Menu\Programs folder, which is displayed when you select Programs from the Start Menu. Note that the menu uses the small version of group, folder, and program icons.

FOLDER_PROGRAMS is valid only with 32-bit setup running under operating systems that use the explorer shell: Windows 95 and higher and Windows NT 4.0 and higher. In 16-bit setups and in 32-bit setups running under the Program Manager shell, this variable will contain a null string ("").

The value of FOLDER_PROGRAMS changes as follows to ensure that groups and folders are created at the proper location on all explorer shell operating systems.

- Under Windows NT 4.0, the location pointed to by FOLDER_PROGRAMS changes when the default group/folder type is changed from Common to Personal or Personal to Common by calling ProgDefGroupType.
- Under Windows 95, the location pointed to by FOLDER_PROGRAMS does not change unless the current user logs out and a different user logs in. Note that this value will change only if user profiles have been enabled; otherwise the default FOLDER_PROGRAMS location is used for all users.



Once user profiles have been enabled on Windows 95, FOLDER_PROGRAMS cannot be set to the default location; it will always point to the current user's desktop folder location. For that reason, it is not possible to install common software on Windows 95 once profiles have been enabled. This is a limitation of Windows 95.

{button ,PI("", `comment')} [Feedback](#)



FOLDER_STARTMENU

This system variable stores the fully qualified path to the Start Menu folder, which is displayed when you Click the Windows Start button. Note that the menu uses the small version of group, folder, and program icons.

FOLDER_STARTMENU is valid only with 32-bit setup running under operating systems that use the explorer shell: Windows 95 and higher and Windows NT 4.0 and higher. In 16-bit setups and in 32-bit setups running under the Program Manager shell, this variable will contain a null string ("").

The value of FOLDER_STARTMENU changes as follows to ensure that groups and folders are created at the proper location on all explorer shell operating systems.

- n Under Windows NT 4.0, the location pointed to by FOLDER_STARTMENU changes when the default group/folder type is changed from Common to Personal or Personal to Common by calling ProgDefGroupType.
- n Under Windows 95, the location pointed to by FOLDER_STARTMENU does not change unless the current user logs out and a different user logs in. Note that this value will change only if user profiles have been enabled; otherwise the default FOLDER_STARTMENU location is used for all users.



Once user profiles have been enabled on Windows 95, FOLDER_STARTMENU cannot be set to the default location; it will always point to the current user's desktop folder location. For that reason, it is not possible to install common software on Windows 95 once profiles have been enabled. This is a limitation of Windows 95.

{button ,PI("",`comment')} [Feedback](#)



FOLDER_STARTUP

This system variable stores the fully qualified path to the Startup folder, which contains the program folders and items that are launched when Windows starts. It is valid only with 32-bit setup running under operating systems that use the explorer shell: Windows 95 and higher and Windows NT 4.0 and higher. In 16-bit setups and in 32-bit setups running under the Program Manager shell, this variable will contain a null string ("").

The value of FOLDER_STARTUP changes as follows to ensure that groups and folders are created at the proper location on all explorer shell operating systems.

- n Under Windows NT 4.0, the location pointed to by FOLDER_STARTUP changes when the default group/folder type is changed from Common to Personal or Personal to Common by calling ProgDefGroupType.
- n Under Windows 95, the location pointed to by FOLDER_STARTUP does not change unless the current user logs out and a different user logs in. Note that this value will change only if user profiles have been enabled; otherwise the default FOLDER_STARTUP location is used for all users.



Once user profiles have been enabled on Windows 95, FOLDER_STARTUP cannot be set to the default location; it will always point to the current user's desktop folder location. For that reason, it is not possible to install common software on Windows 95 once profiles have been enabled. This is a limitation of Windows 95.

{button ,PI("",`comment')} [Feedback](#)



INFOFILENAME

When you use [BatchFileSave](#) to save a batch file or [ConfigFileSave](#) to save a Config.sys file, you can specify that InstallShield create a backup of the file as it existed before you updated it. InstallShield assigns the name of that backup file to the system variable INFOFILENAME. If you want to alert the user to the existence of the backup file, use the function `MessageBox` to display the value of INFOFILENAME.

{button ,PI("",`comment')} Feedback



ISRES

During setup initialization, InstallShield decompresses _isres.dll from _user1.cab and copies it to a temporary folder on the target system, giving it a unique name so that it does not interfere with other InstallShield installations. The fully qualified name of this file, which contains setup resources, is assigned to the system variable ISRES.

{button ,PI("",`comment')} Feedback



ISUSER

During setup initialization, InstallShield decompresses _isuser.dll, if present, from _user1.cab and copies it to the temporary folder Supportdir on the target system, giving the file a unique name so that it does not interfere with other InstallShield installations. The fully qualified name of this file, which contains user-defined setup resources, is assigned to the system variable ISUSER.

{button ,PI("",`comment')} Feedback



ISVERSION

When the setup script starts running, InstallShield gets the version of Setup.exe that is running and assigns it to the system variable ISVERSION. The version number also appears in the Setup program's About box.

{button ,PI("",`comment')} Feedback



MEDIA

This system variable stores the name of the current media library or script-created component set. During setup initialization, MEDIA is assigned the value of 'DATA', which corresponds to the DATAx.cab file that was created by the media build. If you change the value of this system variable to refer to a script-created component set, you must change the value back to 'DATA' before calling [ComponentMoveData](#), [CreateShellObjects](#), or [CreateRegistrySet](#).

{button ,PI("",`comment`)} [Feedback](#)



MODE

The system variable MODE holds one of the following constant values:

- n SILENTMODE—Indicates that Setup.exe is running in silent mode.
- n NORMALMODE—Indicates Setup.exe is running in normal mode.

You can use the system variable MODE in if statements to control the flow of your script based on mode, as shown below:

```
if (MODE = SILENTMODE) then
    // Perform silent setup actions and events.
else
    // Perform normal setup actions and events.
endif;
```

{button ,PI("",`comment')} [Feedback](#)



PROGRAMFILES

This system variable contains the fully qualified name of the folder defined by Windows to store applications. In English Windows, that folder is named "Program Files" and is located off the root of the drive on which Windows is installed. (In other language versions of Windows, the folder name is localized appropriately by default.) The program files folders is the recommended default location for application folders.



If your company distributes more than one application, you may prefer to create a company folder inside the program files folder and then create application folders within the company folder. For example, when InstallShield5.5 is installed, it is placed by default in the following folder:

PROGRAMFILES\InstallShield\InstallShield5.5 Professional Edition

The value of the InstallShield system variable PROGRAMFILES is obtained from the "ProgramFilesDir" key in the Windows registry during setup initialization. Note that since this key is located under the key HKEY_LOCAL_MACHINE, it does not exist on 16-bit platforms; for that reason, 16-bit setups cannot determine this value and will instead set PROGRAMFILES to the folder \ProgramF on the drive with the Windows folder. This same behavior occurs when a 16-bit setup is run on a 32-bit platform.

{button ,PI("",`comment')} [Feedback](#)



SELECTED_LANGUAGE

This numeric system variable contains the ID of the language that the setup is using to display prompts and messages. For a description of the method by which a setup determines the language to use, click [here](#).

{button ,PI("",`comment')} Feedback



SHELL_OBJECT_FOLDER

This system variable is used to store a folder name that can be referenced in the Displayed folder name field on the Shell Object Property sheet of the InstallShield IDE. If the Displayed folder name field is set to <SHELL_OBJECT_FOLDER>, assign a valid folder name to the system variable; then call [CreateShellObjects](#) to create the shell objects in a program folder with that name.

{button ,PI("",`comment')} [Feedback](#)



SRCDIR

This system variable contains the fully qualified path to the source folder that contains the compiled setup script. During setup initialization, InstallShield assigns to SRCDIR the fully qualified path to the folder in which your .ins file is located.

{button ,PI("",`comment')} Feedback



SRCDISK

This system variable contains the name of the drive with the source disk. During setup initialization, InstallShield assigns to SRCDISK the name of the drive that holds the disk containing the setup script file, Setup.ins. For example, if you start Setup.exe from a floppy disk in the A drive, then if that disk contains the file Setup.ins, InstallShield assigns the value "A:" to SRCDISK. Note that InstallShield includes the colon (:) with the drive letter.



If you intend to reference the root folder of the drive specified by this variable, you must append a backslash to it (specified as two backslashes). For example, if the value of SRCDISK is A:, the following statement refers to the root folder of that drive:

```
SRCDISK + "\\ "
```

{button ,PI("",`comment')} Feedback



SUPPORTDIR

During setup initialization, InstallShield locates a folder on the target system into which it can copy temporary files and files that were compressed into _sys1.cab and _user1.cab. InstallShield assigns to SUPPORTDIR the fully qualified path to that folder.

{button ,PI("", `comment')} Feedback



TARGETDIR

During setup initialization, InstallShield assigns to the system variable TARGETDIR the fully qualified path to a target folder on the hard disk. This folder will be the one containing the file Win.ini, usually the Windows folder. Some InstallScript functions use this variable when performing file operations. You must set this variable to the folder you want to target before calling these functions.

```
{button ,AL(`ComponentSetTarget ;CompareFile;CopyFile;DeleteFile;RenameFile;VarRestore;VerSearchAndUpdate  
File;VerFindFileVersion;VerUpdateFile;XCopyFile;Select the destination folder for a component',0,'')} See  
also
```

```
{button ,PI("",`comment')} Feedback
```



TARGETDISK

During setup initialization, InstallShield assigns the name of the target disk drive to the system variable TARGETDISK. This drive will be the one containing the file Win.ini, usually the C: drive. Note that InstallShield includes the colon (:) with the drive letter.



If you intend to reference the root folder of the drive specified by this variable, you must append a backslash to it (specified as two backslashes). For example, if the value of TARGETDISK is C:, the following statement refers to the root folder of that drive:

```
TARGETDISK + "\\\"
```

{button ,PI("",`comment')} Feedback



UNINST

This system variable contains the fully qualified filename of the unInstallShield executable file (IsUninst.exe or IsUn16.exe). Under the Program Manager shell, you can use the -f switch to concatenate the system variable UNINST and uninstallation log filename (returned in DeinstallStart's second parameter) to create the uninstaller icon in your application's program folder:

```
szProgram = UNINST;  
szProgram = szProgram + " -f" + svUninstLogFile;  
AddFolderIcon(svFolder, "unInstallShield",  
    szProgram, WINDIR, "", 0, "", REPLACE);
```

{button ,PI("",`comment')} [Feedback](#)



WINDIR

InstallShield initializes the system variable WINDIR by assigning to it the name of the folder that contains the main operating environment, for example C:\Windows.

{button ,PI("",`comment')} [Feedback](#)



WINDISK

InstallShield initializes the system variable WINDISK by assigning to it the ID of the disk drive that contains the main operating environment. This drive will be the one containing the Windows program, usually the C: drive. Note that InstallShield includes the colon (:) with the drive letter.



If you intend to reference the root folder of the drive specified by this variable, you must append a backslash to it (specified as two backslashes). For example, if the value of WINDISK is C:, the following statement refers to the root folder of that drive:

```
WINDISK + "\\ "
```

{button ,PI("",`comment')} [Feedback](#)



WINSYSDIR

During setup initialization, InstallShield initializes the system variable WINSYSDIR by assigning to it the name of the Windows\System folder, which is usually C:\Windows\System.

{button ,PI("",`comment')} Feedback



WINSYSDISK

During setup initialization, InstallShield initializes the system variable WINSYSDISK by assigning to it the name of the disk drive that contains the Windows\System folder, usually the C: drive. Note that InstallShield includes the colon (:) with the drive letter.



If you intend to reference the root folder of the drive specified by this variable, you must append a backslash to it (specified as two backslashes). For example, if the value of WINSYSDISK is C:, the following statement refers to the root folder of that drive:

```
WINSYSDISK + "\\\"
```

{button ,PI("",`comment')} Feedback

This feature is available only in the InstallShield Professional Edition and International Edition. Click here  for a list of differences between those editions and the InstallShield for Microsoft Visual C++ edition.

InstallShield help is attempting to send you to the InstallShield Web site (www.installshield.com).



Unavailable in Visual C++ edition

This feature is unavailable in the InstallShield for Microsoft Visual C++ edition.

We welcome your suggestions for improving InstallShield help. To share your ideas, please visit <http://www.installshield.com/talk/> or email doc@installshield.com



Built-in function index

A

[A](#) [B](#) [C](#) [D](#) [E](#) [F](#) [G](#) [H](#) [I](#) [L](#) [M](#) [N](#) [O](#) [P](#) [Q](#) [R](#) [S](#) [U](#) [V](#) [W](#) [X](#)

[AddFolderIcon](#)

Adds an icon to a folder.

[AddProfString](#)

Adds a non-unique key to a section of an .ini file.

[AppCommand](#)

Controls the Program Manager.

[AskDestPath](#)

Presents a dialog box that requests a destination location for the setup.

[AskOptions](#)

Presents a dialog box that prompts the end user to select options by using check boxes or radio buttons.

[AskPath](#)

Presents a dialog box that prompts the end user to enter a path.

[AskText](#)

Presents a dialog box that prompts the end user to enter text.

[AskYesNo](#)

Presents a dialog box that prompts the end user to respond to a question by clicking on a Yes or No button.

B

[A](#) [B](#) [C](#) [D](#) [E](#) [F](#) [G](#) [H](#) [I](#) [L](#) [M](#) [N](#) [O](#) [P](#) [Q](#) [R](#) [S](#) [U](#) [V](#) [W](#) [X](#)

[BatchAdd](#)

Adds an environment variable to a batch file.

[BatchDeleteEx](#)

Deletes a line in the batch file.

[BatchFileLoad](#)

Loads a batch file into memory for editing with advanced batch functions.

[BatchFileSave](#)

Saves a batch file that has been loaded with BatchFileLoad.

[BatchFind](#)

Finds items in a batch file.

[BatchGetFileName](#)

Retrieves the fully qualified filename of the default batch file.

[BatchMoveEx](#)

Moves an item within a batch file.

[BatchSetFileName](#)

Specifies a batch file to be the default batch file.

C

[A](#) [B](#) [C](#) [D](#) [E](#) [F](#) [G](#) [H](#) [I](#) [L](#) [M](#) [N](#) [O](#) [P](#) [Q](#) [R](#) [S](#) [U](#) [V](#) [W](#) [X](#)

[CallDLLFx](#)

Calls a function from an external DLL.

[ChangeDirectory](#)

Makes the specified directory the current directory.

[CloseFile](#)

Closes an open file.

CmdGetHwndDlg

Retrieves the handle of a dialog.

CommitSharedFiles

Records locked .dll and .exe files, ensuring that the files are updated when Windows is restarted.

ComponentAddItem

Adds a new component to a script-created component set.

ComponentCompareSizeRequired

Determines if enough free disk space exists for the selected components.

ComponentDialog

Presents a dialog box that allows the end user to select components and specify a destination location.

ComponentError

Returns additional error information when a component function fails.

ComponentFileEnum

Enumerates the files associated with a component.

ComponentFileInfo

Retrieves all available information about a file inside a file media library.

ComponentFilterLanguage

Enables and disables filtering based on language.

ComponentFilterOS

Enables and disables filtering based on operating system (OS).

ComponentGetData

Retrieves information about a component.

ComponentGetItemSize

Determines the size of a specified component.

ComponentIsItemSelected

Determines if the specified component has been selected by the end user.

ComponentListItems

Creates a list of the components in a file media library or a script-created component set.

ComponentMoveData

Transfers and decompresses files associated with selected components in the file media library.

ComponentSelectItem

Selects or deselects components.

ComponentSetData

Sets properties and data for the specified component.

ComponentSetTarget

Specifies a user-defined variable to place in a component's <TARGETDIR> field.

ComponentSetupTypeEnum

Enumerates the setup types associated with the specified file media library.

ComponentSetupTypeGetData

Retrieves data associated with a specified setup type that has been created in the InstallShield IDE.

ComponentSetupTypeSet

Selects all components associated with the specified setup type.

ComponentTotalSize

Calculates the total size, in bytes, of selected components and subcomponents.

ComponentValidate

Validates the password of the entire file media library or a specified component in the file library media.

ConfigAdd

Adds a statement to a system configuration file that has been loaded in memory.

ConfigDelete

Deletes an item from a system configuration file.

ConfigFileLoad

Loads a system configuration file into memory for editing.

ConfigFileSave

Saves a system configuration file that has been loaded into memory with ConfigFileLoad.

ConfigFind

Searches for an item in a system configuration file.

ConfigGetFileName

Retrieves the fully qualified filename of default system configuration file.

ConfigGetInt

Retrieves a value from a system configuration file.

ConfigMove

Moves an item within a system configuration file.

ConfigSetFileName

Specifies the fully qualified filename of a system configuration file.

ConfigSetInt

Sets a value in a system configuration file.

CopyBytes

Copies a specified number of bytes from one string to another.

CopyFile

Copies a file from one folder to another.

CreateDir

Creates a new folder.

CreateFile

Creates a file with the specified filename.

CreateProgramFolder

Creates a program folder.

CreateRegistrySet

Creates one of the sets of registry entries specified in the Resources pane's Registry Entries folder

CreateShellObjects

Creates the folders and shortcuts (or groups and icons) specified in the Resources pane's Shell Objects folder.

CtrlClear

Deletes the contents of an edit, static, list, or combo box control.

CtrlDir

Fills a list box or combo box with either a folder listing or a file listing.

CtrlGetCurSel

Returns the selected item from a list box or combo box.

CtrlGetMLEText

Retrieves the text from a multi-line edit or static field.

CtrlGetMultCurSel

Returns the selected items from a multi-selection list box.

CtrlGetState

Retrieves the state of a radio button, check box, or push button control from a dialog box.

CtrlGetSubCommand

Retrieves the operation performed on the control after a WaitOnDialog function call.

CtrlGetText

Retrieves the text from an edit field, a static field, or the edit field of a combo box.

CtrlPGroups

Retrieves a list of program group names that exist on the target system.

CtrlSelectText

Selects the text displayed in an edit field.

CtrlSetCurSel

Finds and sets the current selection in a list box or combo box.

CtrlSetFont

Sets the font type and style for a control in the dialog box.

CtrlSetList

Places the contents of a list into a list box or combo box.

CtrlSetMLEText

Sets the text in a multi-line edit field.

CtrlSetMultCurSel

Sets the current selection in a multi-selection list box.

CtrlSetState

Sets the current state of a check box, radio button, or push button control.

CtrlSetText

Sets the text in an edit field, a static text field, or the edit field of a combo box.

D

A B C D E F G H I L M N O P Q R S U V W X

DefineDialog

Registers a custom dialog box with InstallShield.

DeinstallSetReference

Specifies a reference file to be checked before the uninstallation process begins.

DeinstallStart

Creates the application uninstallation key and sets the [UninstallString] value under that key.

Delay

Delays the execution of the setup script.

DeleteDir

Deletes a folder.

DeleteFile

Deletes a file.

DeleteFolderIcon

Removes an icon or item from a program folder.

DeleteProgramFolder

Removes a program folder from the target system.

DialogSetInfo

Changes display elements in the dialog boxes presented by some built-in dialog box functions.

Disable

Disables the display of a user interface object.

Do

Executes the currently defined EXIT and HELP handlers.

DoInstall

Launches another InstallShield setup.

E

A B C D E F G H I L M N O P Q R S U V W X

Enable

Enables the display of a user interface object.

EndDialog

Closes a custom dialog box.

EnterDisk

Presents a dialog box that prompts the end user for a specific disk.

ExistsDir

Determines whether or not the specified directory exists.

ExistsDisk

Determines whether or not the specified disk exists.

ExitProgMan

Closes a Program Manager shell that was launched during the setup.

EzBatchAddPath

Modifies the default batch file by adding a path name to the search path in a PATH command or to the value assigned to an environment variable.

EzBatchAddString

Adds a line of text to the default batch file.

EzBatchReplace

Replaces a statement in the default batch file.

EzConfigAddDriver

Adds a device driver statement to the default system configuration file.

EzConfigAddString

Adds a statement or line of text to the default system configuration file.

EzConfigGetValue

Retrieves the value of a system configuration file parameter, such as FILES or BUFFERS.

EzConfigSetValue

Sets the value of a system configuration file parameter, such as FILES or BUFFERS.

EzDefineDialog

Registers a custom dialog box with InstallShield.

F

A B C D E F G H I L M N O P Q R S U V W X

FileCompare

Compares one file with another.

FileDeleteLine

Deletes a line in a text file.

FileGrep

Searches a text file for specified text.

FileInsertLine

Inserts a line in a text file.

FindAllDirs

Finds all subfolders under the specified folder.

FindAllFiles

Finds all files in the specified folder and its subfolders that match a file specification.

FindFile

Finds the first file in the specified folder that matches a file specification.

FindWindow

Retrieves the handle to a window.

G

A B C D E F G H I L M N O P Q R S U V W X

GetDir

Deletes the drive designation from a path name or fully qualified filename.

GetDisk

Retrieves the disk drive designation from a path name or fully qualified filename.

GetDiskSpace

Returns the amount of free space on a disk.

GetEnvVar

Returns the current value of an environment variable.

GetExtents

Returns the dimensions of the screen.

GetFileInfo

Retrieves a file's attributes, date, time, and size.

GetFolderNameList

Retrieves all subfolder names and shortcuts in the specified folder.

GetFont

Retrieves the handle of a font.

GetGroupNameList

Retrieves a list of all program folder names.

GetItemNameList

Retrieves all program items names from a specified program group under the Program Manager shell.

GetLine

Retrieves a line of text from an open file.

GetMemFree

Returns the amount of memory that is available to an application running under Microsoft Windows.

GetMode

Returns the mode in which Microsoft Windows is currently running.

GetProfInt

Returns an integer from an .ini file.

GetProfString

Returns a string from an .ini file.

GetSystemInfo

Retrieves system information.

GetValidDrivesList

Returns a listing of all available drives on the target system.

GetWindowHandle

Returns the handle of the main installation window.

H

A B C D E F G H I L M N O P Q R S U V W X

Handler

Specifies a label to branch to in response to exit and help events.

HIWORD

Retrieves the high-order word from a 32-bit integer.

I

A B C D E F G H I L M N O P Q R S U V W X

InstallationInfo

Creates registry keys based on a company name, product name, and product version number.

Is

Provides file and path checking services, searches for a math coprocessor, tests for administrator status under Windows NT, and determines whether Microsoft Windows is running from a shared copy on a network.

L

A B C D E F G H I L M N O P Q R S U V W X

LaunchApp

Launches another program.

LaunchAppAndWait

Launches another program and waits for that program to terminate.

ListAddItem

Adds an item to a list.

ListAddString

Adds a string to a list.

ListCount

Returns the number of string or numeric elements in a specified list.

ListCreate

Creates a new string or number list.

ListCurrentItem

Returns the current item in a list.

ListCurrentString

Returns the current string in a list.

ListDeleteItem

Deletes the current item in a list.

ListDeleteString

Deletes the current string in a list.

ListDestroy

Destroys a list.

ListFindItem

Makes the specified item the current item in a numeric list.

ListFindString

Makes the specified item the current item in a string list.

ListGetFirstItem

Retrieves the first element from a number list.

ListGetFirstString

Retrieves the first string from a string list.

ListGetNextItem

Retrieves the element after the current element from a number list.

ListGetNextString

Retrieves the element after the current element from a string list.

ListReadFromFile

Reads a text file into a list.

ListSetCurrentItem

Sets the current element of a number list.

ListSetCurrentString

Sets the current element of a string list.

ListSetIndex

Uses an index to set the current element of a list.

ListWriteToFile

Writes a string list to a file.

LongPathFromShortPath

Creates a long filename from a short filename.

LongPathToQuote

Inserts or removes double quotation marks around a long filename.

LongPathToShortPath

Creates a short filename from a long filename.

LOWORD

Retrieves the low-order word from a 32-bit integer.

M

A B C D E F G H I L M N O P Q R S U V W X

MessageBeep

Produces a standard warning beep.

MessageBox

Displays a message in a dialog box.

N

A B C D E F G H I L M N O P Q R S U V W X

NumToStr

Converts a number to a string.

O

A B C D E F G H I L M N O P Q R S U V W X

OpenFile

Opens an existing file.

OpenFileMode

Sets the mode in which files will be opened with the OpenFile function.

P

A B C D E F G H I L M N O P Q R S U V W X

ParsePath

Retrieves the drive, path, filename, or extension from a path.

PathAdd

Adds a path to the search path in the path buffer.

PathDelete

Deletes a path from the search path in the path buffer.

PathFind

Finds a specific path in the path buffer or any path that includes a specified name.

PathGet

Retrieves the current value of the path buffer.

PathMove

Rearranges the path buffer.

PathSet

Assigns a value to the path buffer.

PlaceBitmap

Inserts an image into the installation window.

PlaceWindow

Sets the position of user interface objects.

PlayMMedia

Plays an audio or video file.

ProgDefGroupType

Designates groups as either Personal or Common in a Windows NT environment.

Q

A B C D E F G H I L M N O P Q R S U V W X

QueryProgGroup

Returns information about a specified group.

QueryProgItem

Returns information about a specified program item or subfolder.

QueryShellMgr

Returns the name of the current shell manager.

R

A B C D E F G H I L M N O P Q R S U V W X

RGB

Returns a custom color value based on the specified red, green and blue values.

ReadBytes

Reads the specified number of bytes from a binary file.

RebootDialog

Presents a dialog box that enables the end user to choose to restart Windows or reboot the computer.

RegDBConnectRegistry

Opens a connection to a remote registry.

RegDBCreateKeyEx

Creates a key in the registry. Also allows you to associate a class object with a registry key (advanced users only).

RegDBDeleteKey

Deletes the specified key from the registry.

RegDBDeleteValue

Deletes a value from a specified registry key.

RegDBDisconnectRegistry

Closes the connection to a remote registry.

RegDBGetAppInfo

Retrieves a value from under the application information key.

RegDBGetItem

Retrieves values under the per application paths key or the application uninstallation key.

RegDBGetKeyValueEx

Retrieves a value from under a key in the registry.

RegDBKeyExist

Determines whether a registry key exists.

RegDBQueryKey

Queries a key for its subkeys and value names.

RegDBSetAppInfo

Sets a value under the application information key.

RegDBSetDefaultRoot

Sets the root key.

RegDBSetItem

Assign values under the per application paths key or the application uninstallation key.

RegDBSetKeyValueEx

Sets registry entries.

ReleaseDialog

Frees the memory associated with a dialog box.

ReloadProgGroup

Reloads (updates) a program group.

RenameFile

Renames a file.

ReplaceFolderIcon

Replaces an icon in a specified folder.

ReplaceProfString

Replaces a string in a profile (.ini) file.

S

[A](#) [B](#) [C](#) [D](#) [E](#) [F](#) [G](#) [H](#) [I](#) [L](#) [M](#) [N](#) [O](#) [P](#) [Q](#) [R](#) [S](#) [U](#) [V](#) [W](#) [X](#)

SdAskDestPath

Presents a dialog box that allows the end user to specify a destination location for the setup.

SdAskOptions

Presents a dialog box that has greater flexibility than the standard function [AskOptions](#).

SdAskOptionsList

Presents a dialog box that allows the end user to select and deselect items from a list.

SdBitmap

Displays a bitmap in a dialog box.

SdComponentDialog

Displays a dialog box that allows end users to select folders and components to install.

SdComponentDialog2

Displays a dialog box that allows the end user to select folders, components, and subcomponents to install.

SdComponentDialogAdv

Displays a dialog box that allows the end user to select the components to install. This function uses check boxes. It also provides additional data to help determine the best location for the installation.

SdComponentMult

Displays a dialog box that allows the end user to select the components and subcomponents to install. Additional information about disk space is also provided to help determine the best location for the installation.

SdConfirmNewDir

Prompts the end user to confirm the folder selection.

SdConfirmRegistration

Prompts the end user to confirm the information entered in dialog boxes presented by [SdRegisterUser](#) or [SdRegisterUserEx](#).

SdDisplayTopics

Displays a list of topics.

SdFinish

Displays a dialog box that informs the end user that the setup is complete and offers a choice of options, such as whether to view an information file or launch an application.

SdFinishReboot

Displays a dialog box that informs the user that the setup is complete and offers a choice of options for restarting Windows and the computer.

SdInit

Prepares a setup for calls to Sd dialog box functions.

SdLicense

Displays a license agreement and gives the end user the option of accepting or rejecting the license terms.

SdMakeName

Creates a section name for a custom dialog. This section name is used in writing to and reading from an .iss file, which is used by InstallShield Silent.

SdOptionsButtons

Displays a dialog box with user-defined buttons that provide an end user with various options.

SdProductName

Inserts your product name in certain static fields of the script dialog boxes.

SdRegisterUser

Displays a dialog box in which the end user can enter a user name and company name.

SdRegisterUserEx

Displays a dialog box for entering a user name, company name, and application serial number.

SdSelectFolder

Presents a dialog box that allows the end user to select a folder from a list of program folders.

SdSetupType

Displays a dialog box that enables the end user to select one of the three standard setup types: Typical, Compact, or Custom.

SdSetupTypeEx

Displays a dialog box that enables the end user to select standard or custom setup types.

SdShowAnyDialog

Displays a general-purpose dialog box from a resource DLL. You cannot receive any input from the end user when showing a dialog box with SdShowAnyDialog.

SdShowDlgEdit1

Displays a dialog box that has one single-line edit field and other static controls.

SdShowDlgEdit2

Displays a dialog box that has two single-line edit fields and other static controls.

SdShowDlgEdit3

Displays a dialog box that has three single-line edit fields and other static controls.

SdShowFileMods

Presents a dialog box that previews the changes that may be made to a file and allows the end user to approve the changes, reject the changes, or request that the changes be written to a file.

SdShowInfoList

Displays a scrollable list of messages in a dialog box.

SdShowMsg

Displays a message in a small window.

SdStartCopy

Presents a dialog box that displays the options and settings that have been specified by the end user.

SdWelcome

Displays a general-purpose greeting.

SeekBytes

Positions the file pointer in a binary file.

SelectDir

Presents a dialog box that allows the end user to select a folder. SelectDir creates the folder if it does not exist.

SelectFolder

Presents a dialog box that allows the end user to select a folder from a list of program folders.

SendMessage

Sends a message to another window or application.

SetColor

Changes the color of various user-interface elements.

SetDialogTitle

Creates custom dialog box titles.

SetDisplayEffect

Sets the display effect for bitmap and metafile images.

SetErrorMsg

Changes the text of error messages.

SetErrorTitle

Changes the title of error message boxes.

SetFileInfo

Sets the attributes, date, time and size of a file.

SetFont

Sets the font type and style for a control in the dialog box.

SetStatusWindow

Sets the text for the status window and sets the initial percentage completed value that is displayed in the progress indicator.

SetTitle

Sets the text and color of the title in the main window.

SetupType

Presents a dialog box that allows the end user to select a typical, compact, or custom setup.

ShowGroup

Displays the specified program group.

ShowProgramFolder

Displays the specified program folder.

SilentReadData

Instructs InstallShield Silent to read the .iss file dialog data for a custom dialog box.

SilentWriteData

Instructs InstallShield Silent to write to the .iss file dialog data for a custom dialog box.

SizeWindow

Specifies the size of most user interface objects.

Sprintf

Returns a formatted string composed of one or more character, numeric or string values.

SprintfBox

Displays a formatted string composed of one or more character, numeric, or string values.

StatusUpdate

Links the information gauges to the progress indicator and sets the percent complete to display in the progress indicator after the next file transfer operation.

StrCompare

Compares one string to another.

StrFind

Finds a string in another string.

StrGetTokens

Gets a token from a string based on specified delimiters.

StrLength

Returns the number of characters in a string.

StrRemoveLastSlash

Removes the last backslash in a path string.

StrSub

Returns a substring from a string.

StrToLower

Converts all alphabetic characters in string to lowercase.

StrToNum

Converts a string to a number.

StrToUpper

Converts all alphabetic characters in string to uppercase.

System

Exits to DOS, restarts Windows, or reboots the computer.

U

A B C D E F G H I L M N O P Q R S U V W X

UnUseDLL

Unloads a DLL from memory.

UseDLL

Loads a DLL into memory.

V

[A](#) [B](#) [C](#) [D](#) [E](#) [F](#) [G](#) [H](#) [I](#) [L](#) [M](#) [N](#) [O](#) [P](#) [Q](#) [R](#) [S](#) [U](#) [V](#) [W](#) [X](#)

[VarRestore](#)

Restores the values of the system variables SRCDIR and TARGETDIR that were saved by the last call to VarSave.

[VarSave](#)

Saves the current value of the system variables SRCDIR and TARGETDIR.

[VerCompare](#)

Compares two strings containing version information.

[VerFindFileVersion](#)

Searches for the specified file and retrieves its version and location.

[VerGetFileVersion](#)

Retrieves the version of a specified file.

[VerSearchAndUpdateFile](#)

Replaces an existing file with a more recent version. If the specified file does not exist, the more recent version is installed.

[VerUpdateFile](#)

Replaces an existing file with a more recent version. If the specified file does not exist, the more recent version is *not* installed.

W

[A](#) [B](#) [C](#) [D](#) [E](#) [F](#) [G](#) [H](#) [I](#) [L](#) [M](#) [N](#) [O](#) [P](#) [Q](#) [R](#) [S](#) [U](#) [V](#) [W](#) [X](#)

[WaitOnDialog](#)

Presents a custom dialog box.

[Welcome](#)

Presents a dialog box that displays welcome information.

[WriteBytes](#)

Writes a specified number of bytes to a binary file at the current file pointer location.

[WriteLine](#)

Writes a string to a text file.

[WriteProfString](#)

Writes a string to an .ini file.

X

[A](#) [B](#) [C](#) [D](#) [E](#) [F](#) [G](#) [H](#) [I](#) [L](#) [M](#) [N](#) [O](#) [P](#) [Q](#) [R](#) [S](#) [U](#) [V](#) [W](#) [X](#)

[XCopyFile](#)

Copies one or more files from a source folder to a target folder, including subfolders if specified.

{button ,PI("",`comment')} [Feedback](#)



Ez batch file functions

Ez batch file functions make modifications to the default batch file. Unless changed by a call to BatchSetFileName, the default batch file is the Autoexec.bat file that was executed by the system during the boot sequence. It is important to note that each Ez batch file function opens the default batch file and then saves it automatically after making the specified modification. You do not make calls to open or save when using Ez batch file functions.

EzBatchAddPath

Modifies the default batch file by adding a path name to the search path in a PATH command or to the value assigned to an environment variable.

EzBatchAddString

Adds a line of text to the default batch file.

EzBatchReplace

Replaces a statement in the default batch file.

{button ,PI("",`comment')} Feedback



EzBatchAddPath

Syntax

EzBatchAddPath (szKey, szPath, szRefDir, nPosition);

Description

The EzBatchAddPath function modifies the default batch file by adding a path name either to the search path in a PATH command or to the value assigned to an environment variable. Unless changed by a call to BatchSetFileName, the default batch file is the Autoexec.bat file that was executed by the system during the boot sequence.



This function does not support long filenames. In a 32-bit setup, call LongPathToShortPath to convert the long path to its short path equivalent before passing it to EzBatchAddPath.

To determine the fully qualified name of the default batch file, call BatchGetFileName. To change the name of the batch file to be used by EzBatchAddPath, call BatchSetFileName.



Do not mix Ez batch file functions and advanced batch file functions. After calling BatchFileLoad to load a batch file in memory, you cannot call any of the Ez batch file functions until you call BatchFileSave to save the file.

Parameters

szKey

Specifies the name of the environment variable to modify. For example, to modify the PATH statement, specify "PATH" here. If the specified environment variable is not found in the default file batch file, a complete SET statement is created for that environment variable and inserted into the file.

szPath

Specifies the path name to add to the current value of the environment variable. A delimiting semicolon is inserted to separate it from other path names in the search path.

szRefDir

Specifies the reference key (a path name) relative to which you are adding the new path name specified by szPath. If this is a null string (""), the path name is added to the beginning or end of the search path, depending on the value of nPosition. If the path name specified by szRefDir is not found in the search path, the value of szKey is added to the end.

nPosition

Specifies where in the search path to add the new path name.

BEFORE

The new path name is inserted before the path name specified by szRefDir. If szRefDir contains a null string (""), the path name is added to the front of the search path.

AFTER

The new path name is inserted after the path name specified by szRefDir. If szRefDir contains a null string (""), the path name is added to the end of the search path.

Return values

0

EzBatchAddPath successfully added the path name to the batch file.

< 0

EzBatchAddPath was unable to add the path name to the batch file.

Comments

- n EzBatchAddPath may fail if the default batch file is hidden or read-only.
- n EzBatchAddPath does not make a backup copy of the file it modifies.

{button ,JI('LANGREF.HLP>Examples','EzBatchAddPath_example')} [Example](#)

{button ,AL('EzBatchAddString;EzBatchReplace',0,'')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



EzBatchAddPath example

```
/*-----*\
*
* This example illustrates the usage of the EzBatchAddPath function.
*
* BatchSetFileName is called to set the source batch file to EXAMPLE_BATCH.
*
* EzBatchAddPath is then called to add C:\WINDOWS to the beginning of the
* PATH statement.
*
* The second call adds C:\UTILS after the WINDOWS keyword in the PATH
* statement.
*
* NOTE: In order for this script to run properly, it is advised to set the
*       EXAMPLE_BATCH constant to a valid .BAT file on the target system.
*
\*-----*/

#define EXAMPLE_BATCH "EXAMPLE\\EXAMPLE.BAT"

    STRING szKey, szPath, szRefDir, szTitle, szMsg;
    NUMBER nPosition;

program

    BatchSetFileName(EXAMPLE_BATCH);

    szKey      = "PATH";
    szPath     = "C:\\WINDOWS";
    szRefDir   = "";
    nPosition  = BEFORE;
    if (EzBatchAddPath(szKey, szPath, szRefDir, nPosition) < 0) then

        MessageBox("EzBatchAddPath failed.", SEVERE);
    else

        szTitle = "EzBatchAddPath example";
        szMsg   = "Successful.\n\n%s key added to %s.";
        sprintfBox(INFORMATION, szTitle, szMsg, szPath, szKey);
    endif;

    szKey      = "PATH";
    szPath     = "C:\\UTILS";
    szRefDir   = "WINDOWS";
    nPosition  = AFTER;
    if (EzBatchAddPath(szKey, szPath, szRefDir, nPosition) < 0) then

        MessageBox("EzBatchAddPath failed.", SEVERE);
    else

        sprintfBox(INFORMATION, szTitle, szMsg, szPath, szKey);
    endif;

endprogram

// Source file: Is5fn062.rul
```

{button ,PI("",`comment`)} Feedback



EzBatchAddString

Syntax

EzBatchAddString (szLine, szRefKey, nOptions);

Description

The EzBatchAddString function adds a line of text to the default batch file; unless changed by a call to BatchSetFileName, the default batch file is the Autoexec.bat file that was executed by the system during the boot sequence. To determine the fully qualified name of the default batch file, call BatchGetFileName. To change the name of the batch file to be used by EzBatchAddPath, call BatchSetFileName.



Do not mix Ez batch file functions and advanced batch file functions. After calling BatchFileLoad to load a batch file in memory, you cannot call any of the Ez batch file functions until you call BatchFileSave to save the file.

Parameters

szLine

Specifies the line of text to add to the file. Unless you specify NOSET in nOptions, this function assumes that szLine is an environment variable; the text "SET " is inserted at the front of the string before szLine is written to the default batch file.



Batch files do not support long path names completely. If you are using this function to add a line that contains a long path name, call LongPathToShortPath to convert the long path to its short path equivalent before adding it to the string to be placed in the batch file.

szRefKey

Specifies the reference key relative to which you want to add szLine in the default batch file. EzBatchAddString searches the default batch file for the reference key and places the contents of szLine before or after that line, depending on the value of nOptions.

nOptions

Specifies the option to use; pass one of the following predefined constants in this parameter:

BEFORE

szLine is added before the line containing szRefKey. If szRefKey is a null string (""), szLine is added as the first line of the file.

AFTER

szLine is added after the line containing szRefKey. If szRefKey is a null string (""), szLine is added as the last line of the file.

REPLACE

szLine replaces an existing line in the file. If multiple lines with same key exist, EzBatchAddString replaces only the last line that contains the key.

NOSET

Specifies that the text "SET " should not be inserted at the front of the string in szLine. This constant can be Ored with BEFORE, AFTER, and REPLACE.

When the reference key you are searching for is a DOS command or program name (not an environment variable), OR the constant COMMAND with one of the other option constants, as shown below:

```
EzBatchAddString (szLine, szRefKey, AFTER | COMMAND);
```

Return values

0

EzBatchAddString successfully added the text string to the specified batch file.

< 0

EzBatchAddString was unable to add the text string.

Comments

- n EzBatchAddString looks for the appropriate reference keyword in the parameter szRefKey. For example, the keyword for an environment variable is the name of the environment variable itself.
- n The EzBatchAddString function may fail if the default batch file is hidden or read-only.
- n EzBatchAddString does not make a backup copy of the file it modifies.

{button ,JI('LANGREF.HLP>Examples','EzBatchAddString_example')} [Example](#)

{button ,AL('EzBatchAddPath;EzBatchReplace',0,'','')} See also

{button ,PI('','comment')} [Feedback](#)



EzBatchAddString example

```
/*-----*\
*
* This example illustrates the uses of the EzBatchAddString function.
*
* The first time EzBatchAddString is called, the line "TEMP=C:\EXAPP3" will
* be added to the batch file after the PATH statement.
*
* The second time, the line "rem This is the ..." is added to the first line
* in the batch file.
*
* The third time, the line "C:\EXAPP3\EXAPP.EXE" is added after the path
* statement. The COMMAND argument indicates the line is a command variable.
*
* The fourth time, the line "TEMP=C:\EXAPP4" replaces a line with the same
* key, TEMP. This will replace the line "TEMP=C:\EXAPP3", which was created
* when EzBatchAddString was called for the first time.
*
* NOTE: In order for this script to run properly, it is advised to set the
*       BATCH_FILE constant to a valid batch file on the target system.
*
\*-----*/

#define BATCH_FILE "EXAMPLE\EXAMPLE.BAT"

    STRING szBatchFile, szLine, szRefKey;
    NUMBER nOptions;

program

    // Set the batch file to be edited to BATCH_FILE.
    szBatchFile = BATCH_FILE;
    BatchSetFileName(szBatchFile);

    szLine = "TEMP=C:\EXAPP3";
    szRefKey = "PATH";
    nOptions = AFTER;

/*-----*\
*
* EzBatchAddString adds a line after the path statement to the BATCH_FILE
* file.
*
\*-----*/
    if (EzBatchAddString(szLine, szRefKey, nOptions) < 0) then

        MessageBox("First call to EzBatchAddString failed.", SEVERE);
    else

        MessageBox("First call to EzBatchAddString successful.", INFORMATION);
    endif;

    szLine = "rem This is the first line of the file.";
    szRefKey = "";
    nOptions = BEFORE|NOSET;

/*-----*\
*
* EzBatchAddString adds a line to the top of the BATCH_FILE file.
*
\*-----*/
```

```

\*-----*/
  if (EzBatchAddString(szLine, szRefKey, nOptions) < 0) then

    MessageBox("Second call to EzBatchAddString failed.", SEVERE);
  else

    MessageBox("Second call to EzBatchAddString successful.", INFORMATION);
  endif;

  szLine   = "C:\\EXAPP3\\EXAPP.EXE";
  szRefKey = "PATH";
  nOptions = AFTER|NOSET|COMMAND;

/*-----*\
*
* EzBatchAddString adds a command line after path statement to the BATCH_FILE
* file.
*
\*-----*/
  if (EzBatchAddString(szLine, szRefKey, nOptions) < 0) then

    MessageBox("Third call to EzBatchAddString failed.", SEVERE);
  else

    MessageBox("Third call to EzBatchAddString successful.", INFORMATION);
  endif;

  szLine   = "TEMP=C:\\EXAPP4";
  szRefKey = "PATH";
  nOptions = AFTER|NOSET|REPLACE|COMMAND;

/*-----*\
*
* EzBatchAddString replaces a line in the BATCH_FILE file.
*
\*-----*/
  if (EzBatchAddString(szLine, szRefKey, nOptions) < 0) then

    MessageBox("Fourth call to EzBatchAddString failed.", SEVERE);
  else

    MessageBox("Fourth call to EzBatchAddString successful.", INFORMATION);
  endif;

endprogram

// Source file: Is5fn063.rul

```

{button ,PI("",`comment')} Feedback



EzBatchReplace

Syntax

EzBatchReplace (szNewString);

Description

The EzBatchReplace function replaces an existing line of text in the default batch file; unless changed by a call to BatchSetFileName, the default batch file is the Autoexec.bat file that was executed by the system during the boot sequence. To determine the fully qualified name of the default batch file, call BatchGetFileName. To change the name of the batch file to be used by EzBatchAddPath, call BatchSetFileName.



Do not mix Ez batch file functions and advanced batch file functions. After calling BatchFileLoad to load a batch file in memory, you cannot call any of the Ez batch file functions until you call BatchFileSave to save the file.

Parameters

szNewString

Specifies the new string to insert in place of an existing line in the file.



Batch files do not support long path names completely. If you are using this function to add a line that contains a long path name, call LongPathToShortPath to convert the long path to its short path equivalent before adding it to the string to be placed in the batch file.

Return values

0

EzBatchReplace successfully replaced the line of text.

< 0

EzBatchReplace was unable to replace the line of text.

Comments

- n EzBatchReplace parses szNewString and determines the key of the string. It then searches the default batch file for a line that contains the same key. The function replaces the last line found with the same key.
- n Some common keys in a batch file are PATH, COMSPEC, TEMP, Smartdrv.exe, Win.com, and Share.exe.
- n The EzBatchReplace function may fail if the default batch file is hidden or read-only.
- n EzBatchReplace does not make a backup copy of the file it modifies.

{button ,JI('LANGREF.HLP>Examples','EzBatchReplace_example')} Example

{button ,AL('EzBatchAddString,EzBatchAddPath',0,','')} See also

{button ,PI('','comment')} Feedback



EzBatchReplace example

```
/*-----*\
 *
 * This example illustrates the use of the EzBatchReplace function.
 *
 * EzBatchReplace is called to replace one of the target batch file.
 *
 * NOTE: In order for this script to run properly, it is advised to set the
 *       EXAMPLE_BAT constant to a valid batch file on the target system.
 *
 *-----*/

#define EXAMPLE_BAT "EXAMPLE\\EXAMPLE.BAT"

    STRING szBatchFile, szNewString;

program

    szBatchFile = EXAMPLE_BAT;
    BatchSetFileName(szBatchFile);

    szNewString = "COMSPEC = C:\\DOS\\COMMAND.COM";
/*-----*\
 *
 * EzBatchReplace replaces the line COMSPEC = C:\\COMMAND.COM with
 * COMSPEC = C:\\DOS\\COMMAND.COM. The key it searches for is COMSPEC.
 *
 *-----*/
    if (EzBatchReplace(szNewString) < 0) then

        MessageBox("First call to EzBatchReplace failed.", SEVERE);
    endif;

    szNewString = "SET PATH = C:\\DOS;C:\\MYAPP";
/*-----*\
 *
 * EzBatchReplace replaces the line SET PATH = C:\\DOS with
 * SET PATH = C:\\DOS;C:\\MYAPP. The key it searches for is PATH.
 *
 *-----*/
    if (EzBatchReplace(szNewString) < 0) then

        MessageBox("Second call to EzBatchReplace failed.", SEVERE);
    endif;

    szNewString = "SMARTDRV.EXE /P 1024 /C 512";
/*-----*\
 *
 * The third example replaces the SMARTDRIVE statement in the file.
 * The key is SMARTDRV.EXE.
 *
 *-----*/
    if (EzBatchReplace(szNewString) < 0) then

        MessageBox("Third call to EzBatchReplace failed.", SEVERE);
    endif;

endprogram

// Source file: Is5fn064.rul
```

{button ,PI("",`comment')} Feedback



Advanced batch file functions

Advanced batch file functions differ from Ez batch file functions in that they provide greater flexibility and more control over batch files. Use these functions when you need to make more extensive or complex changes to a batch file.



To edit a batch file with these advanced functions, you must first load the file into memory by calling `BatchFileLoad`. When modifications to the batch file are complete, you must then save the file by calling `BatchFileSave`.

When Setup initializes, it selects the target system's startup batch file (`Autoexec.bat`) as the default batch file; unless changed by a call to `BatchSetFileName`, this is the file that `BatchFileLoad` reads into memory if no other filename is specified. To determine the fully qualified name of the default batch file, call `BatchGetFileName`.

BatchAdd

Adds an environment variable to a batch file.

BatchDeleteEx

Deletes a line in the batch file.

BatchFileLoad

Loads a batch file into memory for editing with advanced batch functions.

BatchFileSave

Saves a batch file that has been loaded with `BatchFileLoad`.

BatchFind

Finds items in a batch file.

BatchGetFileName

Retrieves the fully qualified filename of the default batch file.

BatchMoveEx

Moves an item within a batch file.

BatchSetFileName

Specifies a batch file to be the default batch file.

Related Function

SdShowFileMods

Creates a dialog box that displays proposed file changes and offers options on how to proceed.

{button ,PI("", `comment')} [Feedback](#)



BatchAdd

Syntax

BatchAdd (szKey, szValue, szRefKey, nOptions);

Description

The BatchAdd function inserts a SET command or other DOS command into a batch file that has been loaded into memory with [BatchFileLoad](#). The parameter nOptions enables you to add the new command as the first or last statement in the file, replace an existing statement with the new command, or specify that the new command be added before or after an existing statement.

BatchAdd automatically adds the DOS keyword SET to the beginning of the statement to be inserted unless you OR the constant COMMAND with the value you pass in Options. If you do not explicitly specify REPLACE in nOptions, the specified statement is added even if a duplicate line exists in the batch file.

Before calling BatchAdd, you must call BatchFileLoad to load the file to be modified into memory. After you modify the file, call [BatchFileSave](#) to save it to disk.



Do not mix the Ez batch file functions with the advanced batch file functions. After calling BatchFileLoad, you may not use Ez batch file functions until you have called BatchFileSave to save the file.

Parameters

szKey

Specifies the keyword to add to the batch file. PATH, TEMP, and MYENV are examples of valid keys for this parameter.

szValue

Specifies the value of the key being added to the batch file. This string must be no longer than 512 bytes; passing a string longer than 512 bytes will cause a setup error. To add a longer string, use the [FileGrep](#) and [FileInsertLine](#) functions.



Batch files do not support long path names completely. If you are using this function to add a line that contains a long path name, call [LongPathToShortPath](#) to convert the long path to its short path equivalent before adding it to the string to be placed in the batch file.

szRefKey

Specifies the reference key relative to which you are adding szKey in the batch file.

nOptions

Specifies where in the file to insert the line; pass one of the following predefined constants in this parameter:

BEFORE

The statement is added before the first line that contains szRefKey. If szRefKey is a null string (""), the statement is added as the first line of the file.

AFTER

The statement is added after the last line that contains szRefKey. If szRefKey is a null string (""), the statement is added as the last line of the file.

REPLACE

The statement replaces an existing line in the file. If multiple lines with same key exist, only the last line is replaced. If szKey does not exist in the file, a new line will be added after szRefKey. If szRefKey is a null string (""), the new line is added as the last line of the file.

When the statement to be added is not a SET command, pass a null string ("") in szKey, pass the complete command in szValue, and OR the constant COMMAND with one of the other option constants, as shown below:

```
BatchAdd ("", "PAUSE", "", COMMAND | AFTER);
```

Return values

0

BatchAdd successfully added a SET statement or other command to the batch file.

< 0

BatchAdd was unable to add the SET statement or other command to the batch file.

Comments

An InstallScript reference key is either an environment variable, DOS command, or a program filename. Environment variables are keywords such as PATH, COMSPEC, LIB, or other predefined or user-defined identifier. The value of an environment variable is established by using the DOS SET command. Statements that appear in a batch file must be either DOS commands, program names (with or without command-line parameters), or comments. Refer to your operating system manual for a detailed definition of commands and environment variables.

{button ,JI('LANGREF.HLP>Examples','BatchAdd_example')} [Example](#)

{button ,AL('EzBatchAddString;BatchDeleteEx;BatchFileLoad;BatchFileSave;BatchFind;BatchGetFileName;BatchMoveEx;BatchSetFileName',0,'')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



BatchAdd example

```
/*-----*\
 *
 * This example illustrates the usage of the BatchAdd function.
 *
 * Remember that the file must be loaded using the BatchFileLoad function.
 * This file must also be saved as a backup copy of the original file using
 * the BatchFileSave function.
 *
 * In the first call to BatchAdd, a path statement is added to the batch file.
 *
 * In the second call to BatchAdd, an EXENV statement is added to the loaded
 * batch file.
 *
 * The third call to BatchAdd adds a command statement in the loaded batch
 * file.
 *
 * If any of the calls to BatchAdd fail, the setup will exit.
 *
 * NOTE: In order for this script to run properly, it is advised to set the
 *       preprocessor constants to valid filenames and paths.
 *-----*/

#define EXAMPLE_BAT "EXAMPLE\\EXAMPLE.BAT"
#define EXAMPLE_BAK "EXAMPLE.BAK"

    STRING szPath;

program

    // Load the target batch file into memory.
    // A null string ("") would load the bootup AUTOEXEC.BAT file by default.
    BatchFileLoad (EXAMPLE_BAT);

    szPath = "C:\\EXAPP\\BIN";

/*-----*\
 *
 * The following adds the line SET PATH = C:\\EXAPP\\BIN after the PATH
 * statement in the batch file, even if a duplicate line exists.
 *-----*/
    if (BatchAdd ("PATH", szPath, "PATH", AFTER) < 0) then

        MessageBox ("First call to BatchAdd failed", WARNING);
        abort;
    endif;

    szPath = "C:\\OTHERAPP\\BIN";

/*-----*\
 *
 * The following adds the line SET EXENV = C:\\OTHERAPP\\BIN. If the
 * environment variable EXENV already exists in the batch file, the last EXENV
 * statement is replaced.
 *-----*/
    if (BatchAdd ("EXENV", szPath, "EXENV", REPLACE) < 0) then

        MessageBox ("Second call to BatchAdd failed", WARNING);
```

```
abort;
endif;

/*-----*\
*
* The following adds the command SHARE.EXE before WIN in the example batch
* file on the target system.
*
/*-----*/
if (BatchAdd ("", "SHARE.EXE", "WIN", BEFORE | COMMAND) < 0) then
    MessageBox ("Third call to BatchAdd failed", WARNING);
    abort;
endif;

BatchFileSave (EXAMPLE_BAK);

PrintfBox (INFORMATION, "BatchAdd", "Successful.\n\nThe target batch" +
    "file has been altered. The original was saved as " +
    EXAMPLE_BAK + ".");

endprogram

// Source file: Is5fn007.rul
```

{button ,PI("",`comment')} Feedback



BatchDeleteEx

Syntax

BatchDeleteEx (szKey, nOptions);

Description

The BatchDeleteEx function deletes lines in a batch file that contain the value specified in szKey. Before calling BatchDeleteEx, you must call [BatchFileLoad](#) to load the file to be modified into memory. After you modify the file, call [BatchFileSave](#) to save it to disk.



Do not mix the Ez batch file functions with the advanced batch file functions. After calling BatchFileLoad, you may not use Ez batch file functions until you have called BatchFileSave to save the file.

Parameters

szKey

Specifies the reference keyword that identifies the line or lines to be deleted.

nOptions

Indicates whether szKey contains a command statement or an environment variable. Pass one of the following predefined constants in this parameter:

0

Specifies that szKey is an environment variable, which is either a predefined identifier (such as PATH, COMSPEC, and LIB), or a user-defined identifier. The value of an environment variable is established by the DOS SET command.

COMMAND

Specifies that szKey is either a DOS command or a program name.

Return values

This function always returns 0.

{button ,JI('LANGREF.HLP>Examples','BatchDeleteEx_example')} [Example](#)

{button ,AL('EzBatchAddString;BatchAdd;BatchFileLoad;BatchFind;BatchFileSave;BatchGetFileName;BatchMoveEx;BatchSetFileName',0,'')} See also

{button ,PI('','comment')} [Feedback](#)



BatchDeleteEx example

```
/*-----*\
*
* This example illustrates the use of the BatchDeleteEx function.
*
* It is necessary to load the file using the BatchFileLoad function, and
* save a backup copy of the original using the BatchFileSave function.
*
* The first call to BatchDeleteEx deletes all PATH statements in the batch
* file.
*
* The second call to BatchDeleteEx deletes all line(s) with MYAPP.EXE, such
* as C:\MYAPP\BIN\MYAPP.EXE and D:\APP\MYAPP.EXE.
*
* If any of the calls to BatchDeleteEx fail, the setup will exit.
*
* NOTE: In order for this script to run properly, it is advised to set the
*       preprocessor constants to valid filenames and paths.
*
\*-----*/

#define TARGET_BATCH "EXAMPLE\EXAMPLE.BAT"
#define BACKUP_BATCH "EXAMPLE.BAK"

program

    // Load the batch file to be edited.
    // A null string ("") would load the bootup AUTOEXEC.BAT file by default.
    BatchFileLoad(TARGET_BATCH);

/*-----*\
*
* The following deletes all lines with the word PATH within the loaded batch
* file.
*
\*-----*/
    if (BatchDeleteEx("PATH", 0) < 0) then

        MessageBox("First call to BatchDeleteEx failed", SEVERE);
        abort;
    endif;

/*-----*\
*
* The following deletes all lines with the word MYAPP.EXE within the loaded
* batch file.
*
\*-----*/
    if (BatchDeleteEx("MYAPP.EXE", COMMAND) < 0) then

        MessageBox("Second call to BatchDeleteEx failed", SEVERE);
        abort;
    endif;

    BatchFileSave (BACKUP_BATCH);

    sprintfBox(INFORMATION, "BatchDeleteEx", "Successful.\n\nThe target " +
        "batch file has been altered. The original was saved as " +
        BACKUP_BATCH + ".");
```

endprogram

// Source file: Is5fn008.rul

{button ,PI("",`comment')} Feedback



BatchFileLoad

Syntax

BatchFileLoad (szBatchFile);

Description

The BatchFileLoad function loads a copy of the specified batch file into memory so that other advanced batch file functions can be called to operate on the file. Specify the name of the batch file you want to edit in szBatchFile or pass a null string ("") in szBatchFile to edit the default batch file, which is set initially by InstallShield to the bootup Autoexec.bat file used by the system.

Before using any of the advanced batch file functions, you must call BatchFileLoad to load the file to be modified into memory. After you modify the file, call BatchFileSave to save it to disk. To obtain the fully qualified filename of the batch file that will be used by default in the setup script, call BatchGetFileName. To specify a different batch file to be used by default in the setup script, call BatchSetFileName.



Do not mix the Ez batch file functions with the advanced batch file functions. After calling BatchFileLoad, you may not use Ez batch file functions until you have called BatchFileSave to save the file.

Parameters

szBatchFile

Specifies the fully qualified name of the batch file you are loading into memory. When you pass a null string (""), InstallShield loads the default batch file. Once you specify a filename in this parameter, it is considered the default filename. After calling this function, you can use all the advanced batch file functions to manipulate this file.

Return values

0

BatchFileLoad successfully loaded the specified batch file into memory.

< 0

BatchFileLoad was unable to load the file into memory.

{button ,JI('LANGREF.HLP>Examples', 'BatchFileLoad_example')} Example

{button ,AL('EzBatchAddString;BatchAdd;BatchDeleteEx;BatchFind;BatchGetFileName;BatchMoveEx;BatchFileSave;BatchSetFileName', 0, ',')} See also

{button ,PI('', 'comment')} Feedback



BatchFileLoad example

```
/*-----*\
 *
 * This example illustrates the use of the BatchFileLoad and the BatchFileSave
 * function.
 *
 * The first call to BatchFileLoad loads a batch file to be edited into
 * memory. The original file is then saved as a backup file using
 * BatchFileSave. This backup copy will have a .bak extension.
 *
 * The second call to BatchFileLoad loads a different batch file into memory.
 * The original file is then saved as a backup copy using BatchFileSave. This
 * backup copy will have a numbered extension, preventing replacing of files.
 *
 * NOTE: In order for this script to run properly, the preprocessor constants
 *       should be set to directories and filenames that exist on the target
 *       system.
 *-----*/

#define TARGET_BATCH "EXAMPLE\\EXAMPLE.BAT"
#define TARGET_BATCH2 "EXAMPLE\\EXAMPLE2.BAT"
#define BACKUP_BATCH "EXAMPLE.BAK"
#define BACKUP_BATCH2 "EXAMPLE2.*"

    STRING szMsg;

program

/*-----*\
 *
 * Load the target batch file to be edited.
 * A null string ("") here would load the bootup AUTOEXEC.BAT file by default.
 *-----*/
    if (BatchFileLoad(TARGET_BATCH) < 0) then

        MessageBox("First call to BatchFileLoad failed.", SEVERE);
    else

        MessageBox("First call to BatchFileLoad Successful.", INFORMATION);
    endif;

    // Make changes to the batch file here.

/*-----*\
 *
 * Save the original batch file with a .BAK extension.
 *-----*/
    if (BatchFileSave(BACKUP_BATCH) < 0) then

        MessageBox("First call to BatchFileSave failed.", SEVERE);
    else

        sprintfBox(INFORMATION, "BatchFileSave", "First call to " +
            "BatchFileSave successful,\n\nThe target batch file has " +
            "been altered. The original was saved as " + BACKUP_BATCH +
            ".");
    endif;
endprogram
```

```

endif;

/*-----*\
*
* Load a different file to be edited.
*
/*-----*/
if (BatchFileLoad(TARGET_BATCH2) < 0) then

    MessageBox("Second call to BatchFileLoad failed.", SEVERE);
else

    MessageBox("Second call to BatchFileLoad Successful.", INFORMATION);
endif;

// Make changes to the batch file here...

/*-----*\
*
* Save the original batch file with a numbered extension.
*
/*-----*/
if (BatchFileSave(BACKUP_BATCH2) < 0) then

    MessageBox("Second call to BatchFileSave failed.", SEVERE);
else

    sprintfBox(INFORMATION, "BatchFileSave", "Second call to " +
        "BatchFileSave successful.\n\nThe specified batch file " +
        "has been altered. The original was saved as a numbered" +
        "backup copy.");

endif;

endprogram

// Source file: Is5fn009.rul

```

{button ,PI("",`comment')} Feedback



BatchFileSave

Syntax

BatchFileSave (szBackupFile);

Description

The BatchFileSave function saves to disk a batch file that has been loaded into memory with the function [BatchFileLoad](#). The file is saved under its original name. If a filename is specified in szBackupFile, the original file is renamed with that filename before the edited file is written to disk. If szBackupFile contains a null string (""), the original file is replaced with the modified file. If you do not call BatchFileSave when you are finished modifying a batch file with [advanced batch file functions](#), all modifications will be lost.



Do not mix the Ez batch file functions with the advanced batch file functions. After calling BatchFileLoad, you may not use Ez batch file functions until you have called BatchFileSave to save the file.

Parameters

szBackupFile

Specifies the new filename you are giving to the original file you loaded into memory. This parameter can contain only the filename. Do not include the drive and path with the filename. If this parameter specifies a null string ("") or the original name of the file, no backup file is created.

If szBackupFile has the format "Filename.*", InstallShield attempts to save the file with the extension 001. If a file with the resulting name already exists, InstallShield increments the value of the extension by 1 until a unique file is created. The name of the backup file is stored in the system variable INFOFILENAME.

Return values

0

BatchFileSave successfully saved the batch file in memory to disk.

< 0

BatchFileSave was unable to save the batch file to disk.

{button ,JI('LANGREF.HLP>Examples', 'BatchFileSave_example')} [Example](#)

{button ,AL('EzBatchAddString;BatchAdd;BatchDeleteEx;BatchFileLoad;BatchFind;BatchGetFileName;BatchMoveEx;BatchSetFileName',0,'')} See also

{button ,PI('','comment')} [Feedback](#)



BatchFileSave example

```
/*-----*\
 *
 * This example illustrates the use of the BatchFileLoad and the BatchFileSave
 * function.
 *
 * The first call to BatchFileLoad loads a batch file to be edited into
 * memory. The original file is then saved as a backup file using
 * BatchFileSave. This backup copy will have a .bak extension.
 *
 * The second call to BatchFileLoad loads a different batch file into memory.
 * The original file is then saved as a backup copy using BatchFileSave. This
 * backup copy will have a numbered extension, preventing replacing of files.
 *
 * NOTE: In order for this script to run properly, the preprocessor constants
 *       should be set to directories and filenames that exist on the target
 *       system.
 *-----*/

#define TARGET_BATCH "EXAMPLE\\EXAMPLE.BAT"
#define TARGET_BATCH2 "EXAMPLE\\EXAMPLE2.BAT"
#define BACKUP_BATCH "EXAMPLE.BAK"
#define BACKUP_BATCH2 "EXAMPLE2.*"

    STRING szMsg;

program

/*-----*\
 *
 * Load the target batch file to be edited.
 * A null string ("") here would load the bootup AUTOEXEC.BAT file by default.
 *-----*/
    if (BatchFileLoad(TARGET_BATCH) < 0) then

        MessageBox("First call to BatchFileLoad failed.", SEVERE);
    else

        MessageBox("First call to BatchFileLoad Successful.", INFORMATION);
    endif;

    // Make changes to the batch file here.

/*-----*\
 *
 * Save the original batch file with a .BAK extension.
 *-----*/
    if (BatchFileSave(BACKUP_BATCH) < 0) then

        MessageBox("First call to BatchFileSave failed.", SEVERE);
    else

        sprintfBox(INFORMATION, "BatchFileSave", "First call to " +
            "BatchFileSave successful,\n\nThe target batch file has " +
            "been altered. The original was saved as " + BACKUP_BATCH +
            ".");
    endif;
endprogram
```

```

endif;

/*-----*\
*
* Load a different file to be edited.
*
/*-----*/
if (BatchFileLoad(TARGET_BATCH2) < 0) then

    MessageBox("Second call to BatchFileLoad failed.", SEVERE);
else

    MessageBox("Second call to BatchFileLoad Successful.", INFORMATION);
endif;

// Make changes to the batch file here...

/*-----*\
*
* Save the original batch file with a numbered extension.
*
/*-----*/
if (BatchFileSave(BACKUP_BATCH2) < 0) then

    MessageBox("Second call to BatchFileSave failed.", SEVERE);
else

    sprintfBox(INFORMATION, "BatchFileSave", "Second call to " +
        "BatchFileSave successful.\n\nThe specified batch file " +
        "has been altered. The original was saved as a numbered" +
        "backup copy.");

endif;

endprogram

// Source file: Is5fn009.rul

```

{button ,PI("",`comment')} [Feedback](#)



BatchFind

Syntax

BatchFind (szRefKey, svResult, nOptions);

Description

The BatchFind function searches a batch file for one or more occurrences of the reference key specified in szRefKey. If you specify the constant RESTART in nOptions, the first occurrence of the reference key is returned. To find the next occurrence of szRefKey, call this function repeatedly with nOptions set to CONTINUE.

Before calling BatchFind, you must call [BatchFileLoad](#) to load the file to be modified into memory. After you modify the file, call [BatchFileSave](#) to save it to disk.



Do not mix the Ez batch file functions with the advanced batch file functions. After calling BatchFileLoad, you may not use Ez batch file functions until you have called BatchFileSave to save the file.

Parameters

szRefKey

Specifies the reference key to search for. The reference key can be an environment variable, a DOS command, or a program name. If the reference key is a filename and you do not specify a file extension, the function returns all reference keys with the base filename. For example, if you specify Win.com, the search looks for this reference key only. If you specify Win, the reference keys Win.exe, Win.dll, Win.sys, and so on will be returned if they exist in the batch file.

svResult

Specifies the value of the reference key that was found in the batch file.

nOptions

Specifies where to start the search; pass one of the following predefined constants in this parameter:

CONTINUE

Starts the search from the current position in the batch file.

RESTART

Starts the search from the beginning of the batch file.

When the reference key you are searching for is a DOS command or program name (not an environment variable), OR the constant COMMAND with CONTINUE or RESTART, as shown below:

```
BatchFind ("SCAN.EXE", svResult, COMMAND | RESTART);
```

Return values

0

BatchFind successfully found the value of szRefKey and returned it in svResult.

< 0

BatchFind was unable to find the value of szRefKey and return it in svResult.

{button ,JI('LANGREF.HLP>Examples',`BatchFind_example`)} [Example](#)

{button ,AL(' EzBatchAddString;BatchAdd;BatchDeleteEx;BatchFileLoad;BatchFileSave;BatchGetFileName;BatchMoveEx;BatchSetFileName',0,`,`)} See also

{button ,PI('`,`comment`')} [Feedback](#)



BatchFind example

```
/*-----*\
*
* This example illustrates the BatchFind function.
*
* It is necessary to load the batch file using the BatchFileLoad function and
* save it when finished making modifications using the BatchFileSave
* function.
*
* The first call to BatchFind checks to see if SHARE.EXE exists. A message
* informs the user if the SHARE.EXE reference key is found or not.
*
* The second call to BatchFind searches for the reference key PATH. Each
* time it finds an entry for the PATH reference key, it displays the entry in
* a message box.
*
* NOTE: In order for this script to function properly, the TARGET_BATCH
* constant must be set to a file that exists on the target system.
*
\*-----*/

#define TARGET_BATCH "EXAMPLE\\EXAMPLE.BAT"

    STRING svResult;
    NUMBER nResult;

program

    // Load the target batch file to be edited.
    // A null string ("") would load the bootup AUTOEXEC.BAT file by default.
    BatchFileLoad(TARGET_BATCH);

/*-----*\
*
* Check to see if the SHARE.EXE reference key exists. COMMAND indicates the
* key to search for is a command statement.
*
\*-----*/
    nResult = BatchFind("SHARE.EXE", svResult, COMMAND);

    if (nResult < 0) then

        MessageBox ("SHARE.EXE reference key not found.", WARNING);
    else

        MessageBox ("SHARE.EXE reference key found.", INFORMATION);
    endif;

/*-----*\
*
* Display every PATH reference key entry. RESTART indicates to search from
* the beginning of the file.
*
\*-----*/
    nResult = BatchFind("PATH", svResult, RESTART);

    while (nResult = 0)

        MessageBox(svResult, INFORMATION);
```

```
/*-----*\
*
* Continue searching for every path reference key entry. CONTINUE indicates
* to search from the current location in the file.
*
\*-----*/
    nResult = BatchFind("PATH", svResult, CONTINUE);
    endwhile;

endprogram

// Source file: Is5fn011.rul
```

{button ,PI("",`comment')} Feedback



BatchGetFileName

Syntax

BatchGetFileName (svFileName);

Description

The BatchGetFileName function retrieves the fully qualified name of the default batch file, which is set initially by InstallShield to the bootup Autoexec.bat file used by the system. To specify a different batch file to be used by default in the setup script, call [BatchSetFileName](#).



Do not mix the Ez batch file functions with the advanced batch file functions. After calling [BatchFileLoad](#), you may not use Ez batch file functions until you have called [BatchFileSave](#) to save the file.

Parameters

svFileName

Returns the fully qualified name of the default batch file in svFileName.

Return values

0

BatchGetFileName successfully retrieved the path and filename of the default batch file.

< 0

BatchGetFileName was unable to retrieve the path and filename of the default batch file.

{button ,JI('LANGREF.HLP>Examples','BatchGetFileName_example')} [Example](#)

{button ,AL('EzBatchAddString;BatchAdd;BatchDeleteEx;BatchFileLoad;BatchFileSave;BatchFind;BatchMoveEx;BatchSetFileName;GetLine;ListCreate;OpenFileMode;ParsePath',0,'')} See also

{button ,PI('','comment')} [Feedback](#)



BatchGetFileName example

```
/*-----*\
 *
 * This example illustrates the use of BatchGetFileName and BatchSetFileName
 * functions.
 *
 *-----*/

#define NEW_BATCH "EXAMPLE\\EXAMPLE.BAT"

    STRING svFilename;

program

/*-----*\
 *
 * BatchGetFileName assigns the default batch file's fully qualified path
 * and filename to svFilename.
 *
 *-----*/
    if (BatchGetFileName(svFilename) < 0) then

        MessageBox("First call to BatchGetFileName failed.", SEVERE);
    else

        // Display the filename.
        sprintfBox(INFORMATION, "BatchGetFileName", "First call to " +
            "BatchGetFileName successful.\n\nBatchGetFileName " +
            "returned: %s.", svFilename);
    endif;

/*-----*\
 *
 * BatchSetFileName sets the default batch file to the batch file NEW_BATCH.
 *
 *-----*/
    if (BatchSetFileName(NEW_BATCH) < 0) then

        MessageBox("BatchSetFileName failed.", SEVERE);
    else

        MessageBox("BatchSetFileName succeeded.", INFORMATION);
    endif;

/*-----*\
 *
 * BatchGetFileName assigns the new default batch file's fully qualified path
 * and filename to svFilename.
 *
 *-----*/
    if (BatchGetFileName(svFilename) < 0) then

        MessageBox("Second call to BatchGetFileName failed.", SEVERE);
    else

        // Display the filename.
        sprintfBox(INFORMATION, "BatchGetFileName", "Second call to " +
            "BatchGetFileName successful.\n\nBatchGetFileName " +
            "returned: %s.", svFilename);
    endif;
```

endprogram

// Source file: Is5fn012.rul

{button ,PI("",`comment')} Feedback



BatchMoveEx

Syntax

```
BatchMoveEx (szMove, szRefKey, nOptions, nMoveOption);
```

Description

The BatchMoveEx function moves the line specified by szMove from one location to another in a batch file. The parameter nOptions specifies whether to position the line at the beginning or end of the batch file, or before or after the line specified by szRefKey. Before calling BatchMoveEx, you must call [BatchFileLoad](#) to load the file to be modified into memory. After you modify the file, call [BatchFileSave](#) to save it to disk.



Do not mix the Ez batch file functions with the advanced batch file functions. After calling BatchFileLoad, you may not use Ez batch file functions until you have called BatchFileSave to save the file.

Parameters

szMove

Specifies the reference key that identifies the line to be moved.

szRefKey

Specifies the key that identifies the reference line used to position the line being moved. If szRefKey is a null string (""), the line specified by szMove is moved to the beginning or end of the file, depending on the value of nOptions.

nOptions

Specifies where to move the line; pass one of the following predefined constants in this parameter:

BEFORE

The line specified by szMove is moved before the line containing the reference key in szRefKey. If szRefKey is a null string (""), the line specified by szMove is moved to the beginning of the file.

AFTER

The line specified by szMove is moved after the line containing the reference key in szRefKey. If szRefKey is a null string (""), the line specified by szMove is moved to the end of the file.

When the reference key you are searching for is a DOS command or program name (not an environment variable), OR the constant COMMAND with BEFORE or AFTER, as shown below:

```
BatchMoveEx ("PATH", "SCAN.EXE", BEFORE | COMMAND, 0);
```

nMoveOption

Specifies whether szMove is a command or an environment variable. Pass one of the following predefined constants in this parameter:

0

Specifies that szMove is an environment variable.

COMMAND

Specifies that szMove is a command.

Return values

0

BatchMoveEx successfully moved the specified line in the batch file.

< 0

BatchMoveEx was unable to move the line in the batch file.

{button ,JI('LANGREF.HLP>Examples','BatchMoveEx_example')} [Example](#)

{button ,AL(`EzBatchAddString;BatchAdd;BatchDeleteEx;BatchFileLoad;BatchFileSave;BatchFind;BatchGetFileName;BatchSetFileName',0,`,`')} See also

{button ,PI("",`comment')} [Feedback](#)



BatchMoveEx example

```
/*-----*\
 *
 * This example illustrates the use of the BatchMoveEx function.
 *
 * It is necessary to load the file using the BatchFileLoad function and save
 * the file when finished making modifications using the BatchFileSave
 * function.
 *
 * The first example moves the PATH statement to the last line in the file.
 *
 * The second example moves the SHARE.EXE command before the WIN statement.
 *
 * NOTE: In order for this script to run properly, the TARGET_BATCH and
 *       BACKUP_BATCH constants must be set to an existing directory and file,
 *       respectively, on the target system.
 *
\*-----*/

#define TARGET_BATCH "EXAMPLE\\EXAMPLE.BAT"
#define BACKUP_BATCH "EXAMPLE.BAK"

program

    // Load the target batch file to be edited.
    // A null string ("") would load the bootup AUTOEXEC.BAT file by default.
    BatchFileLoad(TARGET_BATCH);

/*-----*\
 *
 * The following moves the PATH statement to the end of the file.
 *
\*-----*/
    if (BatchMoveEx("PATH", "", AFTER, 0) < 0) then

        MessageBox("First call to BatchMoveEx failed.", SEVERE);
    else

        MessageBox("First call to BatchMoveEx successful.", INFORMATION);
    endif;

/*-----*\
 *
 * The following moves the SHARE.EXE command before the WIN statement.
 *
\*-----*/
    if (BatchMoveEx("SHARE.EXE", "WIN", BEFORE|COMMAND, COMMAND) < 0) then

        MessageBox("Second call to BatchMoveEx failed.", SEVERE);
    else

        MessageBox("Second call to BatchMoveEx successful.", INFORMATION);
    endif;

    BatchFileSave(BACKUP_BATCH);

endprogram

// Source file: Is5fn013.rul
```

{button ,PI("", `comment`)} Feedback



BatchSetFileName

Syntax

BatchSetFileName (szBatchFile);

Description

The BatchSetFileName function specifies the name of the batch file to be used by [Ez batch file functions](#) and by [BatchFileLoad](#) when it is called with a null string ("") as its parameter. In InstallScript, this file is referred to as the [default batch file](#). During setup initialization, the default batch file is set to the bootup Autoexec.bat file used by the system.

It is important to note the following facts about BatchSetFileName:

- n It does verify that the specified file exists.
- n It does not load the file into memory.



Do not mix the Ez batch file functions with the advanced batch file functions. After calling BatchFileLoad, you may not use Ez batch file functions until you have called [BatchFileSave](#) to save the file.

Parameters

szBatchFile

Specifies the fully qualified filename of the batch file to be used by default in the setup script.

Return values

0

BatchSetFileName successfully set the specified file as the default batch file.

< 0

BatchSetFileName was unable to set the file as the default batch file.

Comments

BatchSetFileName simply assigns the name of the default batch file. The function will succeed even if the filename is invalid or the specified file does not exist. An invalid filename will cause subsequent Ez batch file and advanced batch file functions to fail.

{button ,JI('LANGREF.HLP>Examples',`BatchSetFileName_example`)} [Example](#)

{button ,AL(`EzBatchAddString;BatchAdd;BatchDeleteEx;BatchFileLoad;BatchFileSave;BatchFind;BatchGetFileName;BatchMoveEx`,0,`,`)} See also

{button ,PI("",`comment`)} [Feedback](#)



BatchSetFileName example

```
/*-----*\
 *
 * This example illustrates the use of BatchGetFileName and BatchSetFileName
 * functions.
 *
 *-----*/

#define NEW_BATCH "EXAMPLE\\EXAMPLE.BAT"

    STRING svFilename;

program

/*-----*\
 *
 * BatchGetFileName assigns the default batch file's fully qualified path
 * and filename to svFilename.
 *
 *-----*/
    if (BatchGetFileName(svFilename) < 0) then

        MessageBox("First call to BatchGetFileName failed.", SEVERE);
    else

        // Display the filename.
        sprintfBox(INFORMATION, "BatchGetFileName", "First call to " +
            "BatchGetFileName successful.\n\nBatchGetFileName " +
            "returned: %s.", svFilename);
    endif;

/*-----*\
 *
 * BatchSetFileName sets the default batch file to the batch file NEW_BATCH.
 *
 *-----*/
    if (BatchSetFileName(NEW_BATCH) < 0) then

        MessageBox("BatchSetFileName failed.", SEVERE);
    else

        MessageBox("BatchSetFileName succeeded.", INFORMATION);
    endif;

/*-----*\
 *
 * BatchGetFileName assigns the new default batch file's fully qualified path
 * and filename to svFilename.
 *
 *-----*/
    if (BatchGetFileName(svFilename) < 0) then

        MessageBox("Second call to BatchGetFileName failed.", SEVERE);
    else

        // Display the filename.
        sprintfBox(INFORMATION, "BatchGetFileName", "Second call to " +
            "BatchGetFileName successful.\n\nBatchGetFileName " +
            "returned: %s.", svFilename);
    endif;
```

endprogram

// Source file: Is5fn012.rul

{button ,PI("",`comment')} [Feedback](#)



Built-in dialog box functions

The functions below create simple dialog boxes, such as Yes/No dialog boxes and message boxes. Several functions allow you to easily display various types of common dialog boxes.



Built-in dialog boxes that have a Cancel button do not return a CANCEL value when it is selected. Instead, the currently defined exit handler is called.

AskDestPath

Presents a dialog box that requests destination path information.

AskOptions

Presents a dialog box that prompts the end user to select options by using check boxes or radio buttons.

AskPath

Presents a dialog box that prompts the end user to enter a path.

AskText

Presents a dialog box that prompts the end user to enter text.

AskYesNo

Presents a dialog box that prompts the end user to respond to a question by clicking on a Yes or No button.

ComponentDialog

Presents a dialog box that allows the end user to select components and specify a destination location.

EnterDisk

Presents a dialog box that prompts the end user for a specific disk.

MessageBox

Presents a message in a dialog box.

RebootDialog

Presents a dialog box that enables the end user to choose to restart Windows or reboot the computer.

SelectDir

Presents a dialog box that allows the end user to select a folder. SelectDir creates the folder if it does not exist.

SelectFolder

Presents a dialog box that allows the end user to select a folder from a list of program folders.

SetupType

Presents a dialog box that allows the end user to select a typical, compact, or custom setup.

SprintfBox

Returns a formatted string composed of one or more character, numeric, or string values.

Welcome

Presents a dialog box that displays welcome information.

{button ,AL(^Modifying built-in dialog boxes;Exit conditions;Caching dialogs',0,'')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



AskDestPath

Syntax

AskDestPath (szTitle, szMsg, svDir, nReserved);

Description

The AskDestPath function displays a dialog box that allows the end user to specify a destination folder for the files to be installed by the setup. The dialog box also includes a Browse button that allows the end user to select an existing folder on the system. Note that the folder selected by the end user must be writable; non-writable folders will not be accepted. If you want the end user to be able to select folders that are not writable, call the [AskPath](#) function instead.



[View sample dialog](#)

To open the Choose Folder dialog box from the Choose Destination Location dialog box, the end user must click on the Browse . . . button. The Choose Folder dialog box displays a list of all available folders. The end user can select an existing folder or enter a new folder name. If the end user enters the name of a folder that does not exist, a message box appears that allows the end user to create the folder.

Parameters

szTitle

Specifies the title of the dialog box. To display the default title ("Choose Destination Location"), pass a null string ("") in this parameter.

szMsg

Specifies the message to display. To pass multiple lines of static text in this parameter, use the newline escape sequence (\n) to insert a line break. To display the default instructions for this dialog box, pass a null string ("") in this parameter.

svDir

Specifies the default path to display when the dialog box is opened; returns the path to the folder selected by the end user.

nReserved

The value of this parameter must be 0 (zero).

Return values

BACK

Indicates that the Back button was selected.

NEXT

Indicates that the Next button was selected.

Comments

- n If the default folder specified by svDir does not already exist on the end user's system, it will not be created unless the end user presses the Browse button and follows the steps to create it from the Choose Folder dialog box. Therefore, whenever you specify a default folder that you intend to use before calling [ComponentMoveData](#) (which will create the folder if necessary), you must call [ExistsDir](#) when AskDestPath returns in order to determine whether that folder exists. If it does not exist, call [CreateDir](#) to create it on the end user's system.
- n Setups that will run in silent mode should create the new folder if it does not exist before calling AskDestPath. Doing so will ensure that the confirmation dialog is not displayed. Without this step, two response files would be required to handle the two possible conditions.

{button ,JI('LANGREF.HLP>Examples',`AskDestPath_example')}

[Example](#)

{button ,AL(^ AskOptions;AskYesNo;Welcome;Modifying built-in dialog boxes;Exit conditions;Caching dialogs',0,',';')}

See also

{button ,PI("",`comment')} Feedback



AskDestPath example

```
/*-----*\
 *
 * This example illustrates the use of the AskDestPath dialog box function.
 *
 *-----*/

#define DEF_DIR "C:\\EXAMPLE"

    STRING  szTitle, szMsg, svDir;
    NUMBER  nReturn;

program

    // Provide a default path for the installation.
    svDir   = DEF_DIR;

    szMsg   = "Please enter the destination path.\nTo accept the default, " +
              "select > Next";

    szTitle = "AskDestPath Example";

/*-----*\
 *
 * The following displays the AskDestPath dialog box, prompting the user for
 * a destination directory.  svDir is set to the directory.
 *
 *-----*/
    nReturn = AskDestPath(szTitle, szMsg, svDir, 0);

    if (nReturn < 0) then

        MessageBox("AskDestPath failed", SEVERE);
    else

        sprintfBox(INFORMATION, szTitle, "AskDestPath set svDir to: %s", svDir);
    endif;

endprogram

// Source file: Is5fn189.rul
```

{button ,PI("",`comment')} [Feedback](#)



AskOptions

Syntax

AskOptions (nValue, szMsg, szText1, nvCheck1 ,szText2, nvCheck2[, szTextn, nvCheckn] [,..., ...]);

Description

The AskOptions function formats and displays a dialog box that prompts the end user to select one or more options. The default title for this dialog box is "Select Components." To change the contents of the title bar, call [SetDialogTitle](#) before calling AskOptions. The dialog box will display up to nine selection controls, either check boxes or radio buttons, depending on the value of nValue.



[View sample dialog](#)

Parameters

nValue

Specifies the type of controls to display. Pass one of the following predefined constants in this parameter:

EXCLUSIVE

Specifies radio buttons, which allow the end user to select only one option.

NONEXCLUSIVE

Specifies check boxes, which allow the end user to select more than one option.

szMsg

Specifies the message to display in the dialog box. You can use this message to describe the options and/or ask the user to choose one or more options. If the message is too long for one line, use newline escape sequences (\n) to insert a line break.

szText1

Specifies a text label of up to 47 characters to display next to the first check box or radio button. To create an accelerator key, insert an ampersand (&) before the character you want to designate for that purpose. The character will be displayed with an underline to indicate its function. For example, to make Alt+C the accelerator key for Custom, type &Custom. To make Alt+S the accelerator key for Custom type Cu&stom.

nvCheck1

Specifies the initial status of the first check box or radio button when the dialog box is opened; returns the status of the first check box or radio button when the dialog box is closed. The following constants are passed and returned in this parameter:

TRUE

The first check box or radio button is selected.

FALSE

The first check box or radio button is not selected.

szText2

Specifies a text label of up to 47 characters to display next to the second check box/radio button. Create an accelerator in the same manner as you did for szText1.

nvCheck2

Specifies the initial status of the second check box or radio button when the dialog box is opened; returns the status of the second check box or radio button when the dialog box is closed. The following constants are passed and returned in this parameter:

TRUE

The second check box or radio button is selected.

FALSE

The second check box or radio button is not selected.

Up to seven additional options can be defined. Each additional option is indicated by a pair of parameters: a

string parameter that defines a label and a number variable that indicates the state of the option when AskOptions returns. To set the initial state of an option, assign TRUE or FALSE to the number variable before calling AskOptions.



If nValue is EXCLUSIVE and the initial state of more than one option is set to TRUE, AskOptions will preselect the first option in the parameter list that is set to TRUE.

Return values

BACK

Indicates that the Back button was selected. The states of the controls are returned in the individual nvCheck variables.

NEXT

Indicates that the Next button was selected. The states of the controls are returned in the individual nvCheck variables.

```
{button ,JI('LANGREF.HLP>Examples',`AskOptions_example')}
```

Example

```
{button ,AL(`AskText;AskYesNo;SetDialogTitleModifying built-in dialog boxes;Exit conditions;Caching dialogs',0,`,`)}  
See also
```

```
{button ,PI("",`comment')} Feedback
```



AskOptions example

```
/*-----*\
 *
 * This example illustrates the use of the AskOptions dialog box function.
 *
 * AskOptions is displayed twice. The first time it is displayed with check
 * boxes and the second time with radio buttons. The example shows the
 * maximum number of options allowed: nine. Each option text field (szText1
 * and so on) has a limit of 47 characters.
 *
 \*-----*/

STRING szMsg, szText1, szText2, szText3, szText4, szText5, szText6;
STRING szText7, szText8, szText9;

NUMBER nValue, nvCheck1, nvCheck2, nvCheck3, nvCheck4, nvCheck5, nvCheck6;
NUMBER nvCheck7, nvCheck8, nvCheck9, nReturn;

program

    szMsg = "Select from the options below.";
    szText1 = "Option 1";
    szText2 = "Option 2";
    szText3 = "Option 3";
    szText4 = "Option 4";
    szText5 = "Option 5";
    szText6 = "Option 6";
    szText7 = "Option 7";
    szText8 = "Option 8";
    szText9 = "Option 9";

    nvCheck1 = TRUE;
    nvCheck2 = FALSE;
    nvCheck3 = FALSE;
    nvCheck4 = FALSE;
    nvCheck5 = FALSE;
    nvCheck6 = FALSE;
    nvCheck7 = FALSE;
    nvCheck8 = FALSE;
    nvCheck9 = FALSE;

    // Display the check box (NONEXCLUSIVE) dialog box.
    nValue = NONEXCLUSIVE;
    AskOptions(nValue, szMsg,
               szText1, nvCheck1,
               szText2, nvCheck2,
               szText3, nvCheck3,
               szText4, nvCheck4,
               szText5, nvCheck5,
               szText6, nvCheck6,
               szText7, nvCheck7,
               szText8, nvCheck8,
               szText9, nvCheck9);

    // Display the Radio button (EXCLUSIVE) dialog box.
    nValue = EXCLUSIVE;
    AskOptions(nValue, szMsg,
               szText1, nvCheck1,
               szText2, nvCheck2,
               szText3, nvCheck3,
```

```
szText4, nvCheck4,  
szText5, nvCheck5,  
szText6, nvCheck6,  
szText7, nvCheck7,  
szText8, nvCheck8,  
szText9, nvCheck9);
```

```
endprogram
```

```
// Source file: Is5fn190.rul
```

```
{button ,PI("",`comment')} Feedback
```



AskPath

Syntax

AskPath (szMsg, szDefPath, svResultPath);

Description

The AskPath function displays a dialog box that prompts the end user to enter the path to a destination location. The dialog box contains a single-line edit field in which you can display a default path. The end user has three options:

- n Accept the default path.
- n Edit the default path.
- n Display the Choose Folder dialog box to select a folder.

The default title for the dialog is Choose Destination Location. To change the title, call [SetDialogTitle](#) before calling AskPath.



[View sample dialog](#)

Parameters

szMsg

Specifies the message to display in this dialog box. To display the default instructions for this dialog box, pass a null string ("") in this parameter.

szDefPath

Specifies the default path to display in the edit field. The end user can modify this string.

svResultPath

Returns the resulting path name, regardless of whether the user accepts the default path, modifies it, or selects an alternate path from the Choose Folder dialog box. AskPath adds a backslash to the end of the path before placing it into svResultPath. If necessary, that backslash can be removed by calling [StrRemoveLastSlash](#) after AskPath returns.

Return values

NEXT

Indicates that the end user selected the Next button.

BACK

Indicates that the end user selected the Back button.

Comments

- n The edit field displayed in the dialog box scrolls to accommodate long strings.
- n Because the number of characters that can be entered into the edit field is not limited, you should declare the variable passed in svResultPath without an explicit size or with the largest possible size (512 bytes for 16-bit setups, 1024 bytes for 32-bit setups). If the string variable is not large enough to store the text entered by the user, the string is truncated and an error message is displayed. Also note that since this function appends a backslash and a NULL terminator to the end of the string, the size of the string must be at least two characters longer than the path entered by the user.
- n This function will accept the name of a folder that exists but is not writable. To limit the end user's selection to writable folders, call the [AskDestPath](#) function instead.

{button ,JI('LANGREF.HLP>Examples','AskPath_example')} [Example](#)

{button ,AL('AskOptions;AskText;AskYesNo;SetDialogTitle;Modifying built-in dialog boxes;Exit conditions;Caching dialogs',0,','')} [See also](#)

{button ,PI("",`comment')} Feedback



AskPath example

```
/*-----*\
 *
 * This example illustrates the use of the AskPath dialog box.
 *
 *-----*/

    STRING szMsg, szDefPath, svResultPath[101];

program

    // Back button is not required for the initial dialog box.
    Disable(BACKBUTTON);

    szDefPath = "C:\\EXAMPLE\\TARGET";

    szMsg = "Please enter the installation location.\n" +
           "(You may enter up to 100 characters.)";

/*-----*\
 *
 * The following displays the AskPath dialog box.
 *
 *-----*/
    if (AskPath(szMsg, szDefPath, svResultPath) = NEXT) then

        TARGETDIR = svResultPath;
    endif;

    Enable(BACKBUTTON);

    // Display the target directory.
    sprintfBox(INFORMATION, "AskPath", "Successful.\n\nThe Target directory " +
              "is: " + TARGETDIR);

endprogram

// Source file: IS5FN006.rul
```

{button ,PI("",`comment')} [Feedback](#)



AskText

Syntax

AskText (szQuestion, szDefault, svResult);

Description

The AskText function displays a dialog box that contains one static text field and one edit box. Specify default text for the static text field in the parameter szQuestion; specify default text for the edit box in the parameter szDefault. The default title of this dialog box is Enter Information. To change the contents of the title bar, call [SetDialogTitle](#) before calling AskText.



[View sample dialog](#)

Parameters

szQuestion

Specifies the question or statement to display. If the length of the string in this parameter exceeds the width of the static text field, one or more line breaks will be inserted into the string so that it displays on multiple lines in the dialog box. If you prefer, you can format the string manually by inserting one or more \n escape sequences into it. This parameter does not have a default value.

szDefault

Specifies the default text for the edit box.

svResult

Returns the text entered by the end user when the Next button is used to close the dialog box. If the user presses the Back button, the value of svResult will be invalid. In that case, it may be necessary to reset this variable before redisplaying the dialog box.

Return values

BACK

Indicates that the Back button was pressed.

NEXT

Indicates that the Next button was pressed.

Comments

- n The edit box displayed by this dialog box can accept strings of up to 512 bytes in 16-bit setups and 1024 bytes in 32-bit setups. The edit field will scroll if necessary to accommodate a long string. If the string exceeds the maximum length, it will be truncated.
- n The string variable that you pass in svResult must be large enough to accommodate the maximum number of characters that can be accepted by the edit box. Use the autosize method to declare that variable, or declare it as 512 byte string in a 16-bit setup or a 1024 byte string in a 32-bit setup.

{button ,JI('LANGREF.HLP>Examples','AskText_example')} [Example](#)

{button ,AL('AskOptions;AskYesNo;SetDialogTitle;String_size_and_autosize;Modifying built-in dialog boxes;Exit conditions;Caching dialogs',0,'')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



AskText example

```
/*-----*\
 *
 * This example illustrates the usage of the AskText dialog box function.
 *
 \*-----*/

    STRING  szMsg, szDefault, svResult, szTitle;

program

/*-----*\
 *
 * AskText is called to ask the user to enter the company name.
 *
 \*-----*/

    szMsg      = "Please enter your company name.\n\n(For example purposes " +
                "only.)";
    szDefault = "InstallShield Corporation";
    AskText(szMsg, szDefault, svResult);

    // Display what AskText returned.
    szTitle = "AskText Example";
    sprintfBox(INFORMATION, szTitle, "Your company name: %s", svResult);

endprogram

// Source file: Is5fn191.rul
```

{button ,PI("",`comment')} [Feedback](#)



AskYesNo

Syntax

AskYesNo (szQuestion, nDefault);

Description

The AskYesNo function presents a message box that displays a question the end user can answer by clicking on a Yes or No button. The AskYesNo message contains four items:

- n Question mark icon
- n Question text
- n Yes button
- n No button

The default title is Question. To change the contents of the title bar, call [SetDialogTitle](#) before calling AskYesNo.



[View sample dialog](#)

The AskYesNo message box is created by a direct call to the corresponding Windows API function, which displays a system modal dialog box. (Once displayed, a modal dialog box retains focus until it is closed by the user.)

Because this dialog box is displayed by Windows, the text in the buttons cannot be changed by the setup. That text—'Yes' and 'No' in the English version—will be displayed by Windows in the language appropriate to the version of Windows on which the setup is being run; no manual localization of this text is required. If you need to display a more flexible dialog box, call a Windows API function directly or use a custom dialog box.

Parameters

szQuestion

Specifies the question to display in the message box. If the message is too large to fit on one line, embed newline escape characters (\n) in the message to insert line breaks.

nDefault

Specifies the button that is selected by default. Pass one of the following predefined constants in this parameter:

YES

The Yes button is highlighted when the dialog box opens.

NO

The No button is highlighted when the dialog box opens.

Return values

YES

Indicates that the user selected the Yes button.

NO

Indicates that the user selected the No button.

{button ,JI('LANGREF.HLP>Examples','AskYesNo_example')} [Example](#)

{button ,AL('AskOptions;AskPath;AskText;SetDialogTitle;Modifying built-in dialog boxes;Exit conditions;Caching dialogs',0,','')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



AskYesNo example

```
/*-----*\
 *
 * This example illustrates the use of the AskYesNo dialog box function.
 *
 \*-----*/

#define PROGRAM "C:\\EXAMPLE\\NOTEPAD.EXE"
#define PARAM   "C:\\EXAMPLE\\README.TXT"

    STRING szCommand, szCmdLine;

program

    szCommand = PROGRAM;
    szCmdLine = PARAM;

/*-----*\
 *
 * Display the AskYesNo dialog box. The default is set to Yes.
 *
 \*-----*/

    if (AskYesNo("Installation complete. Would you like to read the Readme " +
                "file now?", YES) = YES) then
        LaunchApp(szCommand, szCmdLine);
    endif;

endprogram

// Source file: Is5fn192.rul
```

{button ,PI("",`comment')} [Feedback](#)



EnterDisk

Syntax

EnterDisk (szMsg, szTagFile);

Description

The EnterDisk function displays a dialog box that prompts the end user to insert the next disk. The default title is Setup Needs The Next Disk. To change the title, call [SetDialogTitle](#) before calling EnterDisk. The system variable SRCDIR contains the default path, which is displayed in the dialog box. The end user can modify the default path and change the value of SRCDIR by typing a new path and clicking OK.

EnterDisk recognizes the correct disk by searching the disk for the tag file specified by szTagFile. If the disk does not contain the tag file, an error message prompts the user to enter the correct disk.



[View sample dialog](#)

Parameters

szMsg

Specifies the message that prompts the user to insert the proper disk. To display the default instructions for this dialog box, pass a null string ("") in this parameter.

szTagFile

Specifies the name of the tag file. EnterDisk searches for this file on the inserted disk. If the file is not found, a message is displayed that asks the user to insert the correct disk. If you pass a null string ("") in this parameter, the function does not search for any file; it assumes that the correct diskette is installed.

Return values

OK

Indicates that the user selected the OK button.

< 0

Indicates that an unspecified error has occurred.

{button ,JI('LANGREF.HLP>Examples','EnterDisk_example')} [Example](#)

{button ,AL('SetDialogTitle;Modifying built-in dialog boxes;Exit conditions;Caching dialogs',0,'')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



EnterDisk example

```
/*-----*\
*
* This example illustrates the uses of the EnterDisk function.
*
* EnterDisk is called to prompt the user to insert a disk or to specify a
* path. EnterDisk then searches for the EXAMPLE.TXT file as a tag file to
* indicate if the path is the correct source.
*
\*-----*/

#define SOURCE_DIR "EXAMPLE"

    STRING szMsg, szTagFile;

program

    SRCDIR = SOURCE_DIR;

/*-----*\
*
* The following displays the EnterDisk dialog box.
*
\*-----*/
    szMsg      = "Please insert disk 2";
    szTagFile = "EXAMPLE.TXT";
    EnterDisk(szMsg, szTagFile);

endprogram

// Source file: Is5fn060.rul
```

{button ,PI("",`comment')} [Feedback](#)



MessageBox

Syntax

```
MessageBox (szMsg, nType);
```

Description

The MessageBox function presents a dialog box that contains a message and an icon that indicates the nature of the message: information, warning, or severe. The default title depends on the value of nType, which also indicates the icon type. For example, if you pass INFORMATION in nType, the title "Information" will appear in the title bar. To change the contents of the message box title bar, call [SetDialogTitle](#) before calling MessageBox.



[View sample dialog](#)

Parameters

szMsg

Specifies the message to display. InstallShield does not automatically break the text of the message into separate lines to fit in the message box. If the message is too long for one line, insert a line break by embedding newline escape characters (\n) at appropriate places in the string.

nType

Specifies the type of message box to create and the type of icon to display in the message box. Pass one of the following predefined constants in this parameter (Windows 95 icons are shown):



INFORMATION



WARNING



SEVERE

Return values

The return value is not significant unless you are using standard Microsoft Windows message box styles. If you are using these styles, the return value is the same as the return value from the MessageBox API functions.

Comments

This function uses the Microsoft Windows API MessageBox. The operating environment generates the text for the OK button in the local language (i.e., the language that the operating system is running under). You cannot manually change the text in this button.

The operating environment, not InstallShield, determines the size and location of the message box. Under Windows 3.1 you cannot use SetFont to change the font of the MessageBox.

Advanced developers who are familiar with the Windows can specify any style of message box by using the message box style constants in nType. For more information, see the description for the MessageBox APIs in the programming manual for the operating environment.

The return value of this function is identical to the return value from the MessageBox or APIs. Before you can use style constants with this function, such as MB_OK, MBOK, IDYES, IDABORT, and MBICONASTERISK, you must define them in the script.

{button ,JI('LANGREF.HLP>Examples','MessageBox_example')} [Example](#)

{button ,AL('MessageBeep;SetDialogTitle;Using message boxes as debugging tools;Modifying built-in dialog boxes;Exit conditions;Caching dialogs',0,'')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



MessageBox example

```
/*-----*\
 *
 * This example illustrates the MessageBox function.
 *
 \*-----*/

    STRING szMsg;
    NUMBER nType;

program

/*-----*\
 *
 * The first call to MessageBox displays an INFORMATION-style message box.
 * The second displays a WARNING-style message box, and the third example
 * displays a SEVERE-style box.
 *
 \*-----*/

    szMsg = "This will install Example Program.";
    nType = INFORMATION;
    MessageBox(szMsg, nType );

    szMsg = "Installing this version will replace previous one.";
    nType = WARNING;
    MessageBox(szMsg, nType );

    szMsg = "Cannot install this installation on floppy drives!";
    nType = SEVERE;
    MessageBox(szMsg, nType );

endprogram

// Source file: Is5fn232.rul
```

{button ,PI("",`comment')} [Feedback](#)



RebootDialog

Syntax

RebootDialog (szTitle, szMsg, nDefChoice);

Description

The RebootDialog function displays a dialog box that allows the end user to reboot the computer or restart Windows. If the end user selects either of these options, InstallShield exits the setup script and attempts to carry out the selected operation. If the end user chooses not to reboot the machine or restart Windows, execution returns to the setup script. If the reboot operation fails, the Reboot dialog box continues to display.



[View sample dialog](#)

Parameters

szTitle

Specifies the title of the dialog box. To display the default title ("Restarting Windows"), pass a null string ("") in this parameter.

szMsg

Specifies the message to display in the dialog box. To display the default instructions for this dialog box, pass a null string ("") in this parameter.

nDefChoice

Specifies the default radio button selection. Pass one of the following predefined constants in this parameter:

SYS_BOOTWIN

Makes restarting Windows ("Yes, I want to restart Windows now.") the default radio button selection. This radio button is displayed only on 16-bit systems. On 32-bit systems, SYS_BOOTWIN is mapped to SYS_BOOTMACHINE, which becomes the default radio button.

SYS_BOOTMACHINE

Makes rebooting the machine ("Yes, I want to restart my computer now.") the default radio button selection.

0

Makes the choice to ignore the above options ("No, I will restart my computer later.") the default radio button selection.

Return values

0

Indicates that the user selected the "No, I will restart my computer later." radio button.

Comments

- n RebootDialog can return only a value of 0 ("No, I will restart my computer later."), since any other selection will reboot the machine or the system. Execution returns to the RebootDialog dialog box if the function fails to reboot the system or restart Windows.
- n The first radio button ("Yes, I want to restart Windows now.") is hidden on 32-bit systems because it does not apply.
- n When you call a function with the SHAREDFILE or LOCKEDFILE option and locked .dll or .exe files are encountered, updated versions of the locked files are copied to the target system and the system variable BATCH_INSTALL is set to TRUE. RebootDialog automatically commits the locked files for update when Windows or the system is restarted, unless the user selects the "No, I will restart my computer later." option. In that case, if BATCH_INSTALL is TRUE, you must call the [CommitSharedFiles](#) function before calling RebootDialog so that locked files are committed for update.
- n A preferable alternative to the RebootDialog function is [SdFinishReboot](#), which has a better look and feel than the RebootDialog dialog box, and which never requires you to call CommitSharedFiles.

- n Since InstallShield will make every attempt not to restart Windows or the system when other instances of InstallShield are running, you must make sure all other instances of InstallShield are shut down before calling RebootDialog. In addition, your message to the user should request that they ensure all other applications are shut down before restarting Windows or the system.

{button ,JI(`LANGREF.HLP>Examples`,`RebootDialog_example')}} [Example](#)

{button ,AL(`CommitSharedFiles;SdFinishReboot;Modifying built-in dialog boxes;Exit conditions;Caching dialogs`,0,`,`')}} [See also](#)

{button ,PI(``,`comment`')}} [Feedback](#)



RebootDialog example

```
/*-----*\
 *
 * This example illustrates the usage of the RebootDialog dialog box function.
 *
 * NOTE: If selected, RebootDialog will reboot the local system.
 *
 \*-----*/

    STRING szTitle, szMsg;
    NUMBER nDefChoice;

program

    szTitle    = "RebootDialog Example";
    szMsg      = "";
    nDefChoice = 0;

/*-----*\
 *
 * The RebootDialog dialog box is displayed asking the user if the system
 * should be rebooted.
 *
 \*-----*/

    RebootDialog(szTitle, szMsg, nDefChoice);

endprogram

// Source file: Is5fn236.rul
```

{button ,PI("",`comment')} [Feedback](#)



SelectDir

Syntax

SelectDir (szTitle, szMsg, svDir, bCreate);

Description

The SelectDir function displays a dialog box that allows the end user to specify the folder into which the application will be installed. The end user can enter a fully qualified folder name or select an existing folder from a list. If the end user enters an invalid folder name or an unqualified folder name, a message box is displayed to prompt the end user to enter a valid name. The fully qualified name of the selected folder is returned in svDir.

If the specified folder does not exist and the parameter bCreate is TRUE, a message box appears asking whether or not to create the folder. If the end user clicks Yes, SelectDir automatically creates the specified folder. If the parameter bCreate is set to FALSE and a non-existent folder is selected, the end user is not informed, and SelectDir does not create it. In this case, it is up to you to handle the selection contained in svDir.



SelectDir is called automatically when the end user clicks the Browse button in the dialog boxes presented by [AskDestPath](#), [SdAskDestPath](#) and other InstallShield functions that obtain a folder name.

Parameters

szTitle

Specifies the title of this dialog box. To display the default title ("Choose Folder"), pass a null string ("") in this parameter.

szMsg

Specifies the message you want to display in this dialog box. To display the default instructions for this dialog box, pass a null string ("") in this parameter.

svDir

Specifies the name of a folder that will appear as the default selection. Returns the fully qualified name of the folder selected by the end user.

bCreate

Specifies whether or not you want InstallShield to create the specified folder if it does not already exist. Pass one of the following predefined constants in this parameter:

TRUE

Indicates that the folder should be created if it does not exist.

FALSE

Indicates that the folder should not be created if it does not exist.

Return values

CANCEL

Indicates that the user clicked the Cancel button.

1

Indicates that the function successfully displayed the dialog box. If specified, the function successfully created the new folder.

< 0

Indicates that the function was unable display the dialog box and/or was unable to create the folder.

{button ,JI('LANGREF.HLP>Examples','SelectDir_example')} [Example](#)

{button ,AL('AskDestPath;SdAskDestPath;Modifying built-in dialog boxes;Exit conditions;Caching dialogs',0,',')}

See also

{button ,PI("",`comment')} Feedback



SelectDir example

```
/*-----*\
 *
 * This example illustrates the use of the SelectDir dialog box.
 *
 * The first call to SelectDir prompts the user for a destination directory.
 * If the chosen directory does not exist, SelectDir asks if it should be
 * created.
 *
 * The second call to SelectDir prompts the user for a destination directory
 * again. If the chosen directory does not exist, a message is displayed to
 * choose again.
 *
\*-----*/

    STRING  szTitle, szMsg, svDir;
    BOOL    bCreate;
    NUMBER  nResponse;

program

    szTitle = "SelectDir Example 1";
    szMsg   = "Please choose the directory to install Example App.";
    svDir   = "EXAMPLE\\TARGET";
    bCreate = TRUE;

/*-----*\
 *
 * Prompt user to choose a directory. If the directory does not exist, the
 * user will be asked if it should be created.
 *
\*-----*/
    if (SelectDir(szTitle, szMsg, svDir, bCreate) < 0) then

        MessageBox("SelectDir failed.", SEVERE);
    else

        sprintfBox(INFORMATION, szTitle, "svDir was set to %s.", svDir);
    endif;

selection:

    szTitle = "SelectDir Example 2";
    szMsg   = "Please choose the directory to install Example App 2.0.";
    svDir   = TARGETDIR;
    bCreate = FALSE;

/*-----*\
 *
 * Prompt user to choose a directory. If the directory does not exist, the
 * user will be asked to choose a different directory.
 *
\*-----*/
    if (SelectDir(szTitle, szMsg, svDir, bCreate) < 0) then

        MessageBox("SelectDir failed.", SEVERE);
    else

        if (ExistsDir(svDir) = NOTEXISTS) then
```

```
        szMsg = "Directory %s does not exist!\nPlease choose another " +
                "directory.";
        sprintfBox (WARNING, szTitle, szMsg, svDir);

        goto selection;
    endif;

    sprintfBox(INFORMATION, szTitle, "svDir was set to %s.", svDir);
endif;

endprogram

// Source file: Is5fn161.rul
```

{button ,PI("",`comment')} Feedback



SetupType

Syntax

SetupType (szTitle, szMsg, szReserved, nType, nReserved);

Description

The SetupType function displays a dialog box that enables the end user to select one of the three standard setup types: Typical, Compact, or Custom. These setup options are displayed with standard description text. If you want to add other setup types or change the displayed setup type names or descriptions, call [SdSetupTypeEx](#) instead.



[View sample dialog](#)



If the end user returns to the Setup Type dialog box after using a component dialog box to select and deselect components associated with the selected setup type, those choices will be lost. This occurs because the SetupType function automatically resets the default component selections for the selected setup type each time it is called.

Parameters

szTitle

Specifies the title of this dialog. To display the default title ("Setup Type"), pass a null string ("") in this parameter.

szMsg

Specifies the message you want to display at the top of the dialog box. To display the default instructions for this dialog box, pass a null string ("") in this parameter.

szReserved

Pass a null string ("") in this parameter—no other value is allowed.

nType

Specifies the default setup type when the dialog box is opened. Pass one of the following predefined constants in this parameter:

TYPICAL

Defines the default setup type as Typical.

COMPACT

Defines the default setup type as Compact.

CUSTOM

Defines the default setup type as Custom.

nReserved

Pass zero in this parameter. No other value is allowed.

Return values

TYPICAL

Indicates that the Typical setup type was selected.

COMPACT

Indicates that the Compact setup type was selected.

CUSTOM

Indicates that the Custom setup type was selected.

BACK

Indicates that the Back button was selected.

{button ,JI('LANGREF.HLP>Examples','SetupType_example')} Example

{button ,AL('SdSetupType;SdSetupTypeEx;ComponentSetupTypeSet;Getting setup type selection from the end user;Modifying built-in dialog boxes;Exit conditions;Caching dialogs',0,','')} See also

{button ,PI('','comment')} Feedback



SetupType example

```
/*-----*\
 *
 * This example illustrates the use of SetupType.
 *
 * Comments: To run this example script, create a project (or insert into
 *           a project) with several components and subcomponents with
 *           file groups containing files. This example includes setup of
 *           uninstallation functionality.
 *
 *-----*/

#include "sddialog.h"

// Specify your component name here. These are the names you gave to your
// components in the IDE. A NULL ("") string specifies base components.
#define ASKDESTTITLE "Choose Destination Location"
#define ASKDESTMSG "Choose a destination location for the application."
#define SETUPTYPE_TITLE "Choose Setup Type"
#define SETUPTYPE_MSG "Select a setup type."
#define COMPONENT ""
#define SDCMPDLG_TITLE "Component Selection"
#define SDCMPDLG_MSG "Select components to install and destination location."
#define APPBASE_PATH "Your Company\\Word Processor"
#define COMPANY_NAME "Your Company"
#define PRODUCT_NAME "Word Processor"
#define PRODUCT_VERSION "1.0"
#define PRODUCT_KEY "Word Processor"
#define DEINSTALL_KEY "Word Processor"
#define UNINSTALL_NAME "Word Processor"

prototype HandleComponentError(NUMBER);

STRING svLogFile;
NUMBER nvDisk, nResult;

program
    // Disable Back button, which is not needed.
    Disable( BACKBUTTON );

    // Get the destination location.
    TARGETDIR = PROGRAMFILES ^ APPBASE_PATH;
    AskDestPath( ASKDESTTITLE, ASKDESTMSG, TARGETDIR, 0 );

    // Get setup type AND target location with SdSetupType.
    TARGETDIR = PROGRAMFILES ^ APPBASE_PATH;
    nResult = SetupType(SETUPTYPE_TITLE, SETUPTYPE_MSG, "", TYPICAL, 0);

    // If Custom setup type chosen, let user select components
    // and set change location if desired.
    if (nResult = CUSTOM) then
        SdComponentDialogAdv( SDCMPDLG_TITLE, SDCMPDLG_MSG,
            TARGETDIR, COMPONENT);
    endif;

    // Set up uninstallation.
    InstallationInfo( COMPANY_NAME, PRODUCT_NAME,
        PRODUCT_VERSION, PRODUCT_KEY );
    svLogFile = "Uninst.isu";
    DeinstallStart( TARGETDIR, svLogFile, DEINSTALL_KEY, 0 );
endprogram
```

```

    RegDBSetItem( REGDB_UNINSTALL_NAME, UNINSTALL_NAME );

    // Transfer files based on component selection. Handle errors.
    Enable( STATUSDLG );
    Enable( INDVFILESTATUS );
    StatusUpdate( ON, 100 );
    nResult = ComponentMoveData( MEDIA, nvDisk, 0 );
    HandleComponentError( nResult );

    Disable( INDVFILESTATUS );
    Disable( STATUSDLG );

endprogram

/*-----*\
*
* Function: HandleComponentError
*
* Purpose: This function evaluates the value returned by a Component...
*          function and if the value is less than zero, displays the error
*          number and aborts the setup.
*
*\-----*/
function HandleComponentError( nResult )

    NUMBER  nvError;
    STRING  svMedia, svComponent, svFileGroup, svFile;

begin
    if(nResult < 0) then
        ComponentError( svMedia, svComponent, svFileGroup, svFile, nvError );
        sprintfBox( INFORMATION, "Data Transfer Error Information",
            "ComponentError returned the following data transfer error.\n" +
            "Setup will now abort.\n\n" +
            "Media Name: %s\nComponent: %s\nFile Group: %s\n" +
            "File: %s\nError Number: %ld",
            svMedia, svComponent, svFileGroup, svFile, nvError );
        abort;
    endif;
end;

#include "sddialog.rul"

//Source file: Is5fn258.rul

```

{button ,PI("",`comment')} [Feedback](#)



SprintfBox

Syntax

SprintfBox (nType, szTitle, szFormat [,arg] [...]);

Description

The SprintfBox function presents a message box containing one of three icons, a title, and a formatted message. The message can contain variables that are formatted according to commands you enter.



[View sample dialog](#)



SprintfBox is similar to [MessageBox](#), but SprintfBox permits much more flexible control over displayed items.

Parameters

nType

Specifies the type of icon to display in the message box. Pass one of the following predefined constants in this parameter (Windows 95 icons are shown):



INFORMATION



WARNING



SEVERE

szTitle

Specifies the title of the message box. To display the default title ("Error"), pass a null string ("") in this parameter.

szFormat

Specifies a string that includes a [format specifier](#) for each argument to be included in the message.

arg

Specifies up to ten arguments to be included in the message. There must be one argument for each format specifier in the message; the type of each argument must match the type of its respective format specifier. SprintfBox will fail under the following conditions:

- n More than ten format specifiers and arguments are specified.
- n The number of arguments does not match the number of format specifications.
- n A variable does not match the type of its respective format specifier.

Return values

Unless you are using native Windows message box styles (described below), the return value is insignificant.

Comments

This function uses the Microsoft Windows API MessageBox to create the message box. The OK button for the message box contains text generated by the operating environment. The operating environment, not InstallShield, determines the size and location of the message box.

Advanced developers familiar with the Windows APIs can specify any style of message box by using the native message box style constants in the parameter nType. See the description for the MessageBox or WinMessageBox API in the programming manual for your operating environment. If you are using any of the native message box styles, the InstallShield SprintfBox function will return the return value from the Windows API.

Therefore, you must use the Windows API return values in your script.

For example, if you pass YES|NO|CANCEL as the first parameter, the sprintfBox message box will have Yes, No, and Cancel buttons. The respective button return values, as defined by the Windows API MessageBox, are 6 (IDYES), 7 (IDNO), and 2 (IDCANCEL). You must use the appropriate constant values in your script, either as number literals, or as constants defined as the number literals in the declare block of your script.

Advanced developers can use MB_STYLE directly as the first parameter of the sprintfBox function to replace the constants SEVERE, WARNING, or INFORMATION. The value of MB_STYLE is listed in the Windows.h file. You can either enter the value directly in the parameter nType, or you can use the #define preprocessor directive to define the constants associated with the value.

{button ,JI('LANGREF.HLP>Examples','sprintfBox_example')} [Example](#)

{button ,AL('MessageBox;sprintf;Using message boxes as debugging tools;Modifying built-in dialog boxes;Exit conditions;Caching dialogs',0,'')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



SprintfBox example

```
/*-----*\
 *
 * This example illustrates the usage of the SprintfBox function.
 *
 \*-----*/

    STRING  szString, szTitle, szFormat;
    NUMBER  nNumber, nType;
    CHAR    cChar;

program

    szString = "This is a sample text string";
    nNumber  = 420;
    cChar    = 'P';

    szTitle  = "SprintfBox Example";
    szFormat = "String: %s";

/*-----*\
 *
 * Display SprintfBox with three different icon types.
 *
 \*-----*/

    SprintfBox(INFORMATION, szTitle, szFormat, szString);

    szFormat = "Number: %d";
    SprintfBox(WARNING, szTitle, szFormat, nNumber);

    szFormat = "Character: %c";
    SprintfBox(SEVERE, szTitle, szFormat, cChar);

endprogram

// Source file: Is5fn172.rul
```

{button ,PI(,"",`comment')} Feedback



Welcome

Syntax

Welcome (szTitle, nReserved);

Description

The Welcome function displays a dialog box that welcomes the end user.



[View sample dialog](#)

Parameters

szTitle

Specifies the title of this dialog box. To display the default title ("Welcome"), pass a null string ("") in this parameter.

nReserved

Pass 0 in this parameter.

Return values

BACK

Indicates that the end user selected BACK button.

NEXT

Indicates that the end user selected NEXT button.

< 0

Indicates that Welcome failed to display the dialog box.

Comments

You must call [InstallationInfo](#) before calling Welcome so that InstallShield can insert the product name into the first paragraph of message text in the Welcome dialog box. If you do not pass a product name using InstallationInfo, InstallShield cannot insert the product name and will insert extra spaces instead.

{button ,JI('LANGREF.HLP>Examples','Welcome_example')} [Example](#)

{button ,AL('SdBitmap;SdWelcome;Modifying built-in dialog boxes;Exit conditions;Caching dialogs',0,','')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



Welcome example

```
/*-----*\
 *
 * This example illustrates the use of the Welcome dialog box function.
 *
 \*-----*/

#define COMPANY "ExampleCompany"
#define PRODUCT "ExampleProduct"
#define VER     "5.0"
#define KEY     "EXAMPLE.EXE"
#define DEFPATH "C:\\EXAMPLE"

    STRING szCompany, szProduct, szVersion, szProductKey, szDefLogPath, szKey;
    STRING svLogFile;

program

    // Call InstallationInfo to have Welcome display the product name.
    InstallationInfo(COMPANY, PRODUCT, VER, KEY);

    // Call DeinstallStart to log registry changes.
    DeinstallStart(DEFPATH, svLogFile, KEY, 0);

/*-----*\
 *
 * Display Welcome dialog box.
 *
 \*-----*/

    if (Welcome ("Welcome Dialog Box Example", 0) < 0) then
        MessageBox("Welcome dialog box failed.", SEVERE);
    endif;

endprogram

// Source file: Is5fn273.rul
```

{button ,PI("",`comment')} [Feedback](#)



Component functions

The functions below allow you to control file media and to create and process script-created components.

ComponentAddItem

Adds a new component to a script-created component set.

ComponentCompareSizeRequired

Determines if enough free disk space exists for the selected components.

ComponentDialog

Presents a dialog box that allows the end user to select components and specify a destination location.

ComponentError

Returns additional error information when a Component function fails.

ComponentFileEnum

Enumerates the files associated with a component.

ComponentFileInfo

Retrieves all available information about a file inside a file media library.

ComponentFilterLanguage

Enables and disables filtering based on language.

ComponentFilterOS

Enables and disables filtering based on operating system (OS).

ComponentGetData

Retrieves information about a component.

ComponentGetItemSize

Determines the size of a specified component.

ComponentIsItemSelected

Determines if the specified component has been selected by the end user.

ComponentListItems

Creates a list of the components in a file media library or a script-created component set.

ComponentMoveData

Transfers and decompresses files associated with selected components in the file media library.

ComponentSelectItem

Selects or deselects components.

ComponentSetData

Sets properties and data for the specified component.

ComponentSetTarget

Specifies a user-defined variable to place in a component's <TARGETDIR> field.

ComponentSetupTypeEnum

Enumerates the setup types associated with the specified file media library.

ComponentSetupTypeGetData

Retrieves data associated with a specified setup type that has been created in the InstallShield IDE.

ComponentSetupTypeSet

Selects all components associated with the specified setup type.

ComponentTotalSize

Calculates the total size, in bytes, of selected components and subcomponents.

ComponentValidate

Validates the password of the entire file media library or a specified component in the file library media.

SdSetupType

Displays a dialog box that enables the end user to select one of the three standard setup types: Typical, Compact, or Custom.

SdSetupTypeEx

Displays a dialog box that enables the end user to select standard or custom setup types.

{button ,PI("",`comment')} Feedback



ComponentAddItem

Syntax

ComponentAddItem (szMedia, szComponent, nDataSize, bSelected);

Description

The ComponentAddItem function creates a component in a script-created component set. szMedia is the media name used to specify the script-created component set. If the media specified in szMedia doesn't exist, it will be created. You call ComponentAddItem only once for each component you are adding to a given script-created component set. You can create multiple script-created component sets, each with a unique name (szMedia parameter).

To display a single level of components for selection, use [ComponentDialog](#), [SdComponentDialog](#), or [SdComponentDialogAdv](#). Use [SdComponentDialog2](#) or [SdComponentMult](#) to display components and their subcomponents.



This function cannot be used with file media libraries.

Parameters

szMedia

Specifies the media name of a script-created component set. If it does not exist, ComponentAddItem will create it.

szComponent

Specifies the name of the component to be added. Do not use a null string (""). For more information about specifying components and subcomponents in function calls, click [here](#).

nDataSize

Specifies the size in bytes of the data the component represents. If the component is a series of files, enter the total uncompressed size of all the files.

bSelected

Specifies the default selection setting of the component. Pass one of the following predefined constants in this parameter:

TRUE

Indicates that the component is selected by default. If TRUE is passed to select a subcomponent whose parent component is deselected, then the subcomponent will also be deselected, despite passing TRUE in the bSelected parameter.

FALSE

Indicates that the component is deselected by default.



Default selection settings are cleared when the user selects a component or subcomponent displayed in a dialog box. If the user deselects a component, all of its subcomponents will be deselected. If the user deselects all of a component's subcomponents, the component will be deselected.

Return values

0

ComponentAddItem successfully added the item to the component or subcomponent.

< 0

ComponentAddItem was unable to add the item to the component or subcomponent. Call [ComponentError](#) for additional information.

Comments

You can use `ComponentAddItem` and script-created components to allow users to select from options other than component options. To do so, first call `ComponentAddItem` to create a script-created component set containing the options you want. Then display those options for selection by calling [SdAskOptions](#) or [SdAskOptionsList](#) with the script-created component name in the parameter `szComponent`. Finally, determine which options were selected by calling [ComponentIsItemSelected](#).

{button ,JI('LANGREF.HLP>Examples',`ComponentAddItem_example')} [Example](#)

{button ,AL(`Script created component set vs. file media library;Specifying components in function calls;ComponentDialog;ComponentGetItemSize;ComponentIsItemSelected;ComponentListItems;ComponentSelectedItem;SdAskOptions;SdAskOptionsList;SdComponentDialog;SdComponentDialog2;SdComponentDialogAdv;SdComponentMult,0,`,`')} [See also](#)

{button ,PI('`,`comment')} [Feedback](#)



ComponentAddItem example

```
/*-----*\
*
* This example demonstrates ComponentAddItem, ComponentDialog,
* ComponentSetData, and ComponentIsItemSelected.
*
* Comments: This example uses ComponentAddItem to make script-created
* components. First, the IDE-created file media library
* components are displayed. Then, ComponentAddItem adds three
* components to the script-created components set. Each
* component is visible and selected by default. After
* script-created component selection, the selections are
* shown in messages boxes.
*
* For this example to work properly, create a file media
* library with several components. Also create some file
* groups, insert files into them, and assign the file groups
* to the components.
*\-----*/

// Define a script media name and script media component names.
#define SCRIPTMEDIANAME "ScriptCreatedComponents"
#define SCRIPTMEDIA_COMP1 "Script-created Program Files Component"
#define SCRIPTMEDIA_COMP2 "Script-created Graphics Files Component"
#define SCRIPTMEDIA_COMP3 "Script-created Help Files Component"

// Define strings used in various function calls. In a real setup, you might
// instead define these strings in your string tables for localization.
#define COMPDLGTITLE1 "File Media Component Selection"
#define COMPDLGTITLE2 "Script-created Component Selection"
#define COMPDLGPROMPT "Choose any component to install."
#define WASSELECTED " was selected."
#define NONESELECTED "No component was selected."

// Define a destination location for data transfer.
#define DESTDIR "C:\\Temp"

prototype HandleComponentError( NUMBER );

STRING svDir, szComponent, szData;
NUMBER nResult, nData;
BOOL bCompSelected;

program
    // Disable Back button, which is not needed.
    Disable( BACKBUTTON );

    // Initialize a variable indicating if a component has been selected.
    bCompSelected = FALSE;

    // Set the default target location.
    svDir = DESTDIR;

    // Initialize szComponent to "" to refer to all top-level components.
    szComponent = "";

    // Call ComponentDialog to show file media components.
    ComponentDialog( COMPDLGTITLE1, COMPDLGPROMPT, svDir, szComponent );
```

```

// Set the MEDIA system variable to a media name for your
// script-created components.
MEDIA = SCRIPTMEDIANAME;

// Add components, including the default selection status and
// component dialog display icon, to the script-created component set.
// If the script-created component media does not exist, it is created.
nResult = ComponentAddItem( MEDIA, SCRIPTMEDIA_COMP1, 40000, TRUE );
HandleComponentError( nResult );
ComponentSetData( MEDIA, SCRIPTMEDIA_COMP1,
                  COMPONENT_FIELD_VISIBLE, TRUE, szData );

nResult = ComponentAddItem( MEDIA, SCRIPTMEDIA_COMP2, 132000, TRUE );
HandleComponentError( nResult );
ComponentSetData( MEDIA, SCRIPTMEDIA_COMP2,
                  COMPONENT_FIELD_VISIBLE, TRUE, szData );

nResult = ComponentAddItem( MEDIA, SCRIPTMEDIA_COMP3, 132000, TRUE );
HandleComponentError( nResult );
ComponentSetData( MEDIA, SCRIPTMEDIA_COMP3,
                  COMPONENT_FIELD_VISIBLE, TRUE, szData );

// Call ComponentDialog to show the script-created components you
// added using ComponentAddItem.
ComponentDialog( COMPDLGTITLE2, COMPDLGPROMPT, svDir, szComponent );

// Determine which, if any, script-created components were selected
// and display that information. In a real setup, instead of showing
// message boxes, you would do file transfer or any other operation
// your setup required based on this information.
if ( ComponentIsItemSelected( MEDIA, SCRIPTMEDIA_COMP1 ) ) then
    MessageBox( SCRIPTMEDIA_COMP1 + WASSELECTED, INFORMATION );
    bCompSelected = TRUE;
endif;

if ( ComponentIsItemSelected( MEDIA, SCRIPTMEDIA_COMP2 ) ) then
    MessageBox( SCRIPTMEDIA_COMP2 + WASSELECTED, INFORMATION );
    bCompSelected = TRUE;
endif;

if ( ComponentIsItemSelected( MEDIA, SCRIPTMEDIA_COMP3 ) ) then
    MessageBox( SCRIPTMEDIA_COMP3 + WASSELECTED, INFORMATION );
    bCompSelected = TRUE;
endif;

if ( !bCompSelected ) then
    MessageBox( NONESELECTED, INFORMATION );
endif;

endprogram

/*-----*\
*
* Function: HandleComponentError
*
* Purpose: This function evaluates the value returned by a Component...
*          function and if the value is less than zero, displays the error
*          number and aborts the setup.
*
*\-----*/
function HandleComponentError( nResult )

```

```
NUMBER nvError;
STRING svMedia, svComponent, svFileGroup, svFile;

begin
  if(nResult < 0) then
    ComponentError( svMedia, svComponent, svFileGroup, svFile, nvError );
    sprintfBox( INFORMATION, "Data Transfer Error Information",
      "ComponentError returned the following data transfer error.\n" +
      "Setup will now abort.\n\n" +
      "Media Name: %s\nComponent: %s\nFile Group: %s\n" +
      "File: %s\nError Number: %ld",
      svMedia, svComponent, svFileGroup, svFile, nvError );
    abort;
  endif;
end;

// Source file: Is5fn622.rul
```

{button ,PI("",`comment')} [Feedback](#)



ComponentCompareSizeRequired

Syntax

ComponentCompareSizeRequired (szMedia, svTarget, nvSize);

Description

The ComponentCompareSizeRequired function determines whether the target folder contains enough free space for the selected components specified by szMedia, which must be a [file media library](#). If the target folder does not have sufficient free space, the fully qualified folder name is returned in svTarget and the amount of required free space is returned in nvSize.

Note that the parameter svTarget is used only to return a folder name. You cannot use the parameter to specify a destination folder. ComponentCompareSizeRequired checks the drive indicated by the [Destination folder](#) that you specified on the Properties sheet for each component. If szMedia file media library with components to be installed in the General Application Destination, you must assign a destination path to TARGETDIR before calling ComponentCompareSizeRequired. You can obtain a destination path from an end user by calling [AskDestPath](#) or a component dialog.

If your setup will specify a destination folder for a component in a file library at run time, it must do so by calling [ComponentSetTarget](#) before checking for free space with ComponentCompareSizeRequired.

Parameters

szMedia

Specifies the file media library. If this parameter contains the name of a script-created media, the function will fail.

svTarget

Returns a null string ("") if there is sufficient free space available on the target drive. Returns the target path if there is *not* sufficient free space on the target drive.

nvSize

Returns 0 if there is sufficient free space available on the target drive. Returns the size required if there is *not* sufficient free space on the target drive.

Return values

0

ComponentCompareSizeRequired was successful.

< 0

ComponentCompareSizeRequired failed. Call [ComponentError](#) for additional information.

{button ,JI('LANGREF.HLP>Examples','ComponentCompareSizeRequired_example')} [Example](#)

{button ,AL('Script created component set vs. file media library;Specifying components in function calls;Componentfunctions',0,'')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



ComponentCompareSizeRequired example

```
/*-----*\
*
* This example demonstrates ComponentAddItem, ComponentSetData,
* ComponentCompareSizeRequired, and SdComponentMult.
*
* Comments: Make sure you define DESTDIR as a drive with little space
*           available, such as a floppy drive. Remember to put a
*           diskette into the floppy drive before running the script.
*
*-----*/

#include "sddialog.h"

#define COMP1          "CAD Prog. Files"
#define COMP1SIZE      25000000
#define COMP1DESC      "CAD program EXEs and DLLs."
#define COMP2          "CAD Templates"
#define COMP2SIZE      18000000
#define COMP2DESC      "CAD template files."
#define SUBCOMP1       "Industrial"
#define SUBCOMP1SIZE   7000000
#define SUBCOMP1DESC   "CAD industrial engineering template files."
#define SUBCOMP2       "Civil"
#define SUBCOMP2SIZE   5000000
#define SUBCOMP2DESC   "CAD civil engineering template files."
#define SUBCOMP3       "Mechanical"
#define SUBCOMP3SIZE   6000000
#define SUBCOMP3DESC   "CAD mechanical engineering template files."
#define DESTDIR        "a:\\"
#define SDCOMPTITLE    "Displaying Script-created Components"
#define SDCOMPMSG      "Make sure all components are selected."
#define COMPSIZEERRMSG "Make sure target drive is accessible."

STRING svDir, svTarget;
NUMBER nvSize, nData;

program
    // Disable the Back button, which is not needed.
    Disable( BACKBUTTON );

    // Make a script-created component set including subcomponents
    // and give it a media name of "Run-time CAD".
    MEDIA = "Run-time CAD";
    ComponentAddItem( MEDIA, COMP1, COMP1SIZE, TRUE );
        ComponentSetData ( MEDIA, COMP1, COMPONENT_FIELD_DESCRIPTION,
            nData, COMP1DESC );
    ComponentAddItem( MEDIA, COMP2, COMP1SIZE, TRUE );
        ComponentSetData ( MEDIA, COMP2, COMPONENT_FIELD_DESCRIPTION,
            nData, COMP2DESC );
    ComponentAddItem( MEDIA, COMP2 + "\\\" + SUBCOMP1, SUBCOMP1SIZE, TRUE );
        ComponentSetData ( MEDIA, COMP2 + "\\\" + SUBCOMP1,
            COMPONENT_FIELD_DESCRIPTION, nData, SUBCOMP1DESC );
    ComponentAddItem( MEDIA, COMP2 + "\\\" + SUBCOMP2, SUBCOMP2SIZE, TRUE );
        ComponentSetData ( MEDIA, COMP2 + "\\\" + SUBCOMP2,
            COMPONENT_FIELD_DESCRIPTION, nData, SUBCOMP2DESC );
    ComponentAddItem( MEDIA, COMP2 + "\\\" + SUBCOMP3, SUBCOMP3SIZE, TRUE );
        ComponentSetData ( MEDIA, COMP2 + "\\\" + SUBCOMP3,
            COMPONENT_FIELD_DESCRIPTION, nData, SUBCOMP3DESC );
```

```
// Display the script-created components and subcomponents.
SdComponentMult( SDCOMPTITLE, SDCOMPMSG, svDir, "" );

// Set TARGETDIR to DESTDIR, which, if you define as a floppy drive
// (with a diskette in it), will cause ComponentCompareSizeRequired
// to display its 'not enough space' message.
nvSize = 0;
TARGETDIR = DESTDIR;
if ( ComponentCompareSizeRequired(MEDIA, svTarget, nvSize) < 0 ) then
    MessageBox( COMPSIZEERRMSG, SEVERE );
endif;

endprogram

#include "sddialog.rul"

// Source file: Is5fn598.rul
```

{button ,PI("",`comment')} Feedback



ComponentDialog

Syntax

ComponentDialog (szTitle, szMsg, svDir, szComponent);

Description

The ComponentDialog function displays a dialog box that allows the end user to select one or more items from a list of components in the current media. The user can also select a destination location.



If your setup does not use a setup type dialog, you *must* call [ComponentSetupTypeSet](#) to specify a setup type that has been defined in the IDE Setup Types pane before calling ComponentDialog.

The name of the current media is stored in the system variable MEDIA. During setup initialization, InstallShield assigns to MEDIA the default media name ("DATA"), which is associated with your [file media library](#) (Data1.cab). To display script-created components, follow these steps:

1. Save the current value of MEDIA.
2. Assign to MEDIA the name of the script-created media.
3. Call ComponentDialog to get end-user selections.
4. Assign to MEDIA the value saved in step 1.



The value of the system variable MEDIA is set to 'DATA' when during setup initialization. If you change the value of this variable to refer to a script-created component set, you must change the value back to 'DATA' before calling [ComponentMoveData](#), [CreateShellObjects](#), or [CreateRegistrySet](#).



[View sample dialog](#)

Clicking the Browse button launches the Choose Folder dialog box, which displays a list of existing folders. The end user can select an existing folder or enter a new folder name. ComponentDialog returns the name of the selected folder in svDir.

If the user enters a folder that does not exist, a message box appears asking if the end user wants to create the folder. If yes, InstallShield creates the specified folder.

Parameters

szTitle

Specifies the dialog box title. To display the default title ("Select Components"), pass a null string ("") in this parameter.

szMsg

Specifies the message to display in the dialog box. For example, the message might read "Please select one or more of the following components to install on your system." To display the default instructions for this dialog box, pass a null string ("") in this parameter.

svDir

Specifies the default destination location. Returns the folder selected by the end user. Note that the location returned in svDir does not affect file transfer unless you assign it to TARGETDIR or call [ComponentSetTarget](#) to associate it with a user-defined variable.

szComponent

Specifies the component whose subcomponents are to be displayed for selection. Pass a null string ("") in this parameter to display all top-level components. For more information about specifying components and subcomponents in function calls, click [here](#).

Return values

BACK

Indicates that the end user selected BACK button.

NEXT

Indicates that the end user selected NEXT button.

< 0

Unable to display ComponentDialog dialog box. Call [ComponentError](#) for additional information.

Comments

- n If the default folder specified by svDir does not already exist on the end user's system, it will not be created unless the end user presses the Browse button and follows the steps to create it from the Choose Folder dialog box. Therefore, whenever you specify a default folder that you intend to use before calling [ComponentMoveData](#) (which will create the folder if necessary), you must call [ExistsDir](#) when ComponentDialog returns in order to determine whether that folder exists. If it does not exist, call [CreateDir](#) to create it on the end user's system.
- n Setups that will run in silent mode should create the new folder if it does not exist before calling ComponentDialog. Doing so will ensure that the confirmation dialog is not displayed. Without this step, two response files would be required to handle the two possible conditions.

{button ,JI('LANGREF.HLP>Examples','ComponentDialog_example')} [Example](#)

{button ,AL('Script created component set vs. file media library;Specifying components in function calls;Displaying icons in component dialogs;ComponentAddItem;ComponentGetItemSize;ComponentIsItemSelected;ComponentListItems;ComponentSelectItem;SdComponentDialog;SdComponentDialogAdv;ComponentSetupTypeSet',0,'')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



ComponentDialog example

```
/*-----*\
*
* This example demonstrates ComponentAddItem, ComponentDialog,
* ComponentSetData, and ComponentIsItemSelected.
*
* Comments: This example uses ComponentAddItem to make script-created
* components. First, the IDE-created file media library
* components are displayed. Then, ComponentAddItem adds three
* components to the script-created components set. Each
* component is visible and selected by default. After
* script-created component selection, the selections are
* shown in messages boxes.
*
* For this example to work properly, create a file media
* library with several components. Also create some file
* groups, insert files into them, and assign the file groups
* to the components.
*
*\-----*/

// Define a script media name and script media component names.
#define SCRIPTMEDIANAME "ScriptCreatedComponents"
#define SCRIPTMEDIA_COMP1 "Script-created Program Files Component"
#define SCRIPTMEDIA_COMP2 "Script-created Graphics Files Component"
#define SCRIPTMEDIA_COMP3 "Script-created Help Files Component"

// Define strings used in various function calls. In a real setup, you might
// instead define these strings in your string tables for localization.
#define COMPDLGTITLE1 "File Media Component Selection"
#define COMPDLGTITLE2 "Script-created Component Selection"
#define COMPDLGPROMPT "Choose any component to install."
#define WASSELECTED " was selected."
#define NONSELECTED "No component was selected."

// Define a destination location for data transfer.
#define DESTDIR "C:\\Temp"

prototype HandleComponentError( NUMBER );

STRING svDir, szComponent, szData;
NUMBER nResult, nData;
BOOL bCompSelected;

program
    // Disable Back button, which is not needed.
    Disable( BACKBUTTON );

    // Initialize a variable indicating if a component has been selected.
    bCompSelected = FALSE;

    // Set the default target location.
    svDir = DESTDIR;

    // Initialize szComponent to "" to refer to all top-level components.
    szComponent = "";

    // Call ComponentDialog to show file media components.
    ComponentDialog( COMPDLGTITLE1, COMPDLGPROMPT, svDir, szComponent );
```

```

// Set the MEDIA system variable to a media name for your
// script-created components.
MEDIA = SCRIPTMEDIANAME;

// Add components, including the default selection status and
// component dialog display icon, to the script-created component set.
// If the script-created component media does not exist, it is created.
nResult = ComponentAddItem( MEDIA, SCRIPTMEDIA_COMP1, 40000, TRUE );
HandleComponentError( nResult );
ComponentSetData( MEDIA, SCRIPTMEDIA_COMP1,
                  COMPONENT_FIELD_VISIBLE, TRUE, szData );

nResult = ComponentAddItem( MEDIA, SCRIPTMEDIA_COMP2, 132000, TRUE );
HandleComponentError( nResult );
ComponentSetData( MEDIA, SCRIPTMEDIA_COMP2,
                  COMPONENT_FIELD_VISIBLE, TRUE, szData );

nResult = ComponentAddItem( MEDIA, SCRIPTMEDIA_COMP3, 132000, TRUE );
HandleComponentError( nResult );
ComponentSetData( MEDIA, SCRIPTMEDIA_COMP3,
                  COMPONENT_FIELD_VISIBLE, TRUE, szData );

// Call ComponentDialog to show the script-created components you
// added using ComponentAddItem.
ComponentDialog( COMPDLGTITLE2, COMPDLGPROMPT, svDir, szComponent );

// Determine which, if any, script-created components were selected
// and display that information. In a real setup, instead of showing
// message boxes, you would do file transfer or any other operation
// your setup required based on this information.
if ( ComponentIsItemSelected( MEDIA, SCRIPTMEDIA_COMP1 ) ) then
    MessageBox( SCRIPTMEDIA_COMP1 + WASSELECTED, INFORMATION );
    bCompSelected = TRUE;
endif;

if ( ComponentIsItemSelected( MEDIA, SCRIPTMEDIA_COMP2 ) ) then
    MessageBox( SCRIPTMEDIA_COMP2 + WASSELECTED, INFORMATION );
    bCompSelected = TRUE;
endif;

if ( ComponentIsItemSelected( MEDIA, SCRIPTMEDIA_COMP3 ) ) then
    MessageBox( SCRIPTMEDIA_COMP3 + WASSELECTED, INFORMATION );
    bCompSelected = TRUE;
endif;

if ( !bCompSelected ) then
    MessageBox( NONESELECTED, INFORMATION );
endif;

endprogram

/*-----*\
*
* Function: HandleComponentError
*
* Purpose: This function evaluates the value returned by a Component...
*          function and if the value is less than zero, displays the error
*          number and aborts the setup.
*
*-----*/
function HandleComponentError( nResult )

```

```
NUMBER nvError;
STRING svMedia, svComponent, svFileGroup, svFile;

begin
  if(nResult < 0) then
    ComponentError( svMedia, svComponent, svFileGroup, svFile, nvError );
    sprintfBox( INFORMATION, "Data Transfer Error Information",
      "ComponentError returned the following data transfer error.\n" +
      "Setup will now abort.\n\n" +
      "Media Name: %s\nComponent: %s\nFile Group: %s\n" +
      "File: %s\nError Number: %ld",
      svMedia, svComponent, svFileGroup, svFile, nvError );
    abort;
  endif;
end;

// Source file: Is5fn622.rul
```

{button ,PI("",`comment')} [Feedback](#)



ComponentError

Syntax

ComponentError (svMedia, svComponent, svFileGroup, svFile, nvError);

Description

The ComponentError function obtains additional error information when a Component function returns a value less than zero. The following code fragment shows a typical implementation of ComponentError:

```

nResult = ComponentMoveData (szMedia, nvDisk, nReserved );
if (nResult < 0) then
    ComponentError (svMedia, svComponent, svFileGroup, svFile, nvError);
    sprintfBox (INFORMATION, "ComponentMoveData Error Information",
        "ComponentMoveData had the following error:\n\n" +
        "Media Name: %s\nComponent: %s\nFile Group: %s\n" +
        "File: %s\nError Number: %ld (refer to the online help)",
        svMedia, svComponent, svFileGroup, svFile, nvError);
endif;

```

The ComponentError function should be called only after another Component function returns a value less than zero. The ComponentError function may return invalid error codes if called when another Component function has not returned a value less than zero.



ComponentError does not work with the following functions:

- n [ComponentSetupTypeEnum](#)
- n [ComponentSetupTypeSet](#)

Parameters

svMedia

Returns the name of the media if relevant to the error indicated by nvError.

svComponent

Returns the name of the component if relevant to the error indicated by nvError.

svFileGroup

Returns the name of the file group if relevant to the error indicated by nvError.

svFile

Returns the name of the file if relevant to the error indicated by nvError.

nvError

Returns a code that identifies the type of error that occurred in a previous call to a component-related function. These error codes are described below.

Error Codes

ComponentError handles component function error codes ranging from -100 to -499. The error code descriptions below indicate which parameters contain error information.

Error Code	Parameters Containing Data	Descriptions
-101	svMedia, svComponent, nvError	ComponentAddItem was unable to add a component to the script-created component set.
-102	svMedia, svComponent, nvError	Specified component already exists.
-103	svMedia,	ComponentSelectItem tried to select a

	svComponent, nvError	component that was grayed out.
-104	svMedia, svComponent, nvError	Specified component name is not valid.
-105	svMedia, svComponent, nvError	Specified component cannot be found in the media.
-106	svMedia, svComponent, svFileGroup, svFile, nvError	Unable to decompress a file.
-107	svMedia, nvError	Disk ID specified in call to <u>ComponentMoveData</u> is not valid.
-108	svMedia, svFile, nvError	Target disk does not have enough space. svFile will contain target path.
-109	svMedia, svFile, nvError	<u>EnterDisk</u> function call failed. svFile will contain tag file name (name of next library).
-112	svMedia, svComponent, svFileGroup, svFile, nvError	Specified file cannot be found. If svComponent and svFileGroup are empty, svFile contains the name of the <u>file media library</u> that cannot be found. If svComponent and svFileGroup are not empty, then svFile contains the name of the file (inside the file media library) that cannot be found.
-113	svMedia, svComponent, svFileGroup, svFile, nvError	Specified file cannot be opened as read-only. If svComponent and svFileGroup are empty, svFile contains the name of the file that cannot be opened as read-only. If svComponent and svFileGroup are not empty, then svFile contains the name of the file (inside the file media library) that cannot be opened as read-only.
-114	svMedia, svComponent, svFileGroup, svFile, nvError	Specified file cannot be opened as read/write. If svComponent and svFileGroup are empty, svFile contains the name of the file that cannot be opened as read/write. If svComponent and svFileGroup are not empty, then svFile contains the name of the file (inside the file media library) that cannot be opened as read/write.
-115	svMedia, svComponent, svFileGroup, svFile, nvError	Specified file cannot be opened as write. If svComponent and svFileGroup are empty, svFile contains the name of the file that cannot be opened as write. If svComponent and svFileGroup are not empty, then svFile contains the name of the file (inside the file media library) that cannot be opened as write.
-116	svMedia, svComponent, nvError	File specification made in <u>ComponentFileInfo</u> is not valid.
-117	svMedia, svComponent, svFileGroup, svFile, nvError	Cannot read the specified file. If svComponent and svFileGroup are empty, svFile contains the name of the file that cannot be read. If svComponent and

		svFileGroup are not empty, then svFile contains the name of the file (inside the file media library) that cannot be read.
-118	svMedia, nvError	Attempted operation not allowed with script media.
-119	svMedia, svComponent, svFileGroup, svFile, nvError	Unable to self-register a file in ComponentMoveData. Make sure that the self-registering file self-registers properly.
-120	svMedia, svComponent, svFileGroup, svFile, nvError	Unable to update a shared file in ComponentMoveData.
-121	svMedia, svComponent, svFileGroup, svFile, nvError	Unable to write to a file. If svComponent and svFileGroup are empty, svFile contains the name of the file that cannot be written to. If svComponent and svFileGroup are not empty, then svFile contains the name of the file (inside the file media library) that cannot be written to.
-123	svMedia, svComponent, svFileGroup, nvError	Unable to find a file group.
-125	svMedia, svComponent, nvError	The list specified in call to <u>ComponentFileEnum</u> is not valid.
-126	svMedia, nvError	Attempted operation not allowed with file media library.
-127	svMedia, svFile, nvError	Media is already initialized. svFile contains the file media library name.
-128	svMedia, svFile, nvError	Specified file media library was not generated by the InstallShield Media Build Wizard.
-129	nvError	Specified media name was a null string ("").
-132	svMedia, nvError	Specified media cannot be found.
-133	svMedia, nvError	An error occurred with the specified media. Must reset the media by calling ComponentMoveData with first parameter set to a null string ("") and remaining parameters set to values used in previous call to ComponentMoveData.
-136	svMedia, svFile, nvError	Unable to allocate memory. svFile contains the string for which memory was to be allocated, if applicable.
-137	svMedia, nvError	Specified option is not valid.
-139	svMedia, svComponent, svFile, nvError	Specified password does not match the password stored in the specified file media library or the component. svFile will contain the specified password.
-140	svMedia, svComponent, svFile, nvError	Tried to get password using <u>ComponentGetData</u> , which is not allowed!

-141	svMedia, svComponent, svFile, nvError	Specified password cannot be found. The specified media or component does not have a password.
-142	svMedia, svComponent, nvError	The media or the component password was not validated. Make sure you validate the media or component password using ComponentValidate .
-145	svMedia, svComponent, svFileGroup, nvError	Target path for the component or file group cannot be found.
-147	nvError	Invalid value passed to a component-related function.



Error codes less than or equal to -500 (-501, -502, and so on) are internal error codes. nvError contains the error code, but all other ComponentError parameters empty. Contact Technical Support when one of these error codes is returned.

Return values

0

ComponentError was successful.

< 0

ComponentError failed.

{button ,JI('LANGREF.HLP>Examples','ComponentError_example')} [Example](#)

{button ,AL('Script created component set vs. file media library;Specifying components in function calls;Componentfunctions',0,'')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



ComponentError example

```
/*-----*\
*
* This example demonstrates SdSetupTypeEx, SdComponentDialog,
* ComponentIsItemSelected, ComponentGetData, ComponentValidate,
* ComponentMoveData, ComponentError, and PlaceWindow.
*
* Comments: To run this example script, create a project with the
*           following components (c), subcomponents (sc), and file
*           groups (fg):
*
*           (c) Program Files
*               (fg) Program DLLS
*               (fg) Program EXEs
*           (c) Example Files
*               (sc) Small Documents
*                   (fg) Small Document Examples
*               (sc) Books
*                   (fg) Book Examples
*               (sc) Graphics
*                   (fg) Graphic Examples
*           (c) Help Files
*               (fg) Help Files
*           (c) Utilities
*               (sc) Grammar Checker
*                   (fg) Grammar Checker
*               (sc) Art Studio
*                   (fg) Art Studio
*           (c) Evaluation Copy
*               (fg) Evaluation Copy
*               (fg) Help Files
*
* Insert "dummy" files into the file groups. Make sure you define the
* correct filename for the main EXE (MAIN_EXE, below) that you insert into
* the Program EXEs file group. Run the setup with and without a password
* assigned to the Program Files component (remember to rebuild your media
* each time).
*
* You can also create billboards (name them Bbrd1.bmp, Bbrd2.bmp, and
* Bbrd3.bmp) and add them to the project in the Setup Files pane under
* the Language Independent folder.
*
* This example script installs files, adds an icon to the Start Programs
* menu, and provides uninstallation functionality.
*
\*-----*/

#include "sddialog.h"

// Define strings. In a real setup you would define these in your string tables
// and precede each constant with @ to use them in your script.
#define COMP_SELECT_TITLE           "Select Components"
#define COMP_SELECT_MSG             "Select components and subcomponents to
install."
#define COMP_PROGRAMFILES_DISPLAYTEXT "Program Files"
#define PASSWORD_PROMPT             "Please enter the password."
#define PASSWORD_ERRMSG             "Password incorrect. Please enter again."
#define TITLE_MAIN                  "Word Processor"
#define TITLE_CAPTIONBAR            "Word Processor Setup"
#define APPBASE_PATH                "Your Company\\Word Processor"
```

```

#define COMPANY_NAME           "Your Company"
#define PRODUCT_NAME          "Word Processor"
#define PRODUCT_VERSION       "1.0"
#define PRODUCT_KEY           "Word Processor"
#define DEINSTALL_KEY         "Word Processor"
#define UNINSTALL_NAME        "Word Processor"
#define ADDINGICON             "Adding program icon to the Start Programs
menu..."
#define PROGRAMDIR             "Program"
#define DEFAULT_FOLDER_NAME    ""
#define APP_NAME               "Word Processor"
#define COMPLETE_MSG           "Setup is complete. You can run Word
Processor from the Start Programs menu."
#define MAIN_EXE               "WRITE.EXE"
#define SETUPTYPE_TITLE        "Setup Type Selection"
#define SETUPTYPE_MSG          "Please select a setup type."
#define SETUPTYPE_CUSTOM       "Custom"

```

```

// Global variable declarations.
STRING svData, svLogFile, szProgram, szComponent, svResult, svSetupType, svDir;
BOOL    bInitStepsDone, bPwdValid;
NUMBER  nvData, nvDisk, nResult;

```

```

// Function declarations.
prototype SetUpFileTransfer();
prototype HandleComponentError(NUMBER);
prototype FinishSetup();

```

```

program
    SetUpFileTransfer();

    // Get the setup type.
    Disable( BACKBUTTON );
    svDir = TARGETDIR;
    SdSetupTypeEx( SETUPTYPE_TITLE, SETUPTYPE_MSG, "", svSetupType, 0 );
    // If user selected Custom setup type, display component selection dialog.
    if ( svSetupType = SETUPTYPE_CUSTOM ) then
        SdComponentDialog( COMP_SELECT_TITLE, COMP_SELECT_MSG, svDir, "" );
    endif;
    Enable( BACKBUTTON );

    // If the Program Files component is selected and there is a password
    // associated with it, the user must input the password and
    // it must be properly validated.
    nResult = FALSE;
    nvData = FALSE;
    nResult = ComponentIsItemSelected( MEDIA, COMP_PROGRAMFILES_DISPLAYTEXT );
    ComponentGetData( MEDIA, COMP_PROGRAMFILES_DISPLAYTEXT,
        COMPONENT_FIELD_PASSWORD, nvData, svData);
    if ( nResult && nvData ) then
        bPwdValid = FALSE;
        Disable( BACKBUTTON ); // Back button not needed or supported here.
        while ( !bPwdValid )
            AskText( PASSWORD_PROMPT, "", svResult );
            nResult = ComponentValidate( MEDIA, COMP_PROGRAMFILES_DISPLAYTEXT,
                svResult );

            if ( nResult = 0 ) then
                bPwdValid = TRUE;
            else
                MessageBox( PASSWORD_ERRMSG, SEVERE );
            endif;
        endwhile;
    endif;
endprogram

```

```

    Enable( BACKBUTTON ); // Restore Back button's default status.
endif;

// Set up the progress indicator, including locations for
// progress indicator, information gauges, and billboards.
PlaceWindow( FEEDBACK, LOWER_LEFT, LOWER_LEFT, LOWER_LEFT );
PlaceWindow( STATUSDLG, CENTERED, LOWER_RIGHT, LOWER_RIGHT );
PlaceWindow( BILLBOARD, CENTERED, CENTERED, CENTERED );
Enable( STATUSDLG );
Enable( INDVFILESTATUS );

// Indicate the final percentage the progress bar is to show when the
// following file transfer operation is complete.
StatusUpdate( ON, 95 );

// Transfer files to the target system. ComponentMoveData will prompt
// for next disk in a floppy disk installation.
nResult=ComponentMoveData( MEDIA, nvDisk, 0);

// To see the ComponentError function in action in the following call
// to the HandleComponentError user-defined function, remark out the
// while...endwhile loop above in which ComponentValidate is called,
// make sure a password is associated with the Program Files component
// in the IDE, rebuild the media, and run the setup.
HandleComponentError( nResult );

FinishSetup();
endprogram

/*-----*\
*
* Function: SetupFileTransfer()
*
* Purpose: This function sets up file transfer. The main reason for
*          abstracting this process into this function is to make it
*          easy to see the function calls this sample script demonstrates.
*
\*-----*/
function SetUpFileTransfer()
begin
    // Set up the installation screen.
    Enable( FULLWINDOWMODE );
    SetTitle( TITLE_MAIN, 24, WHITE );
    SetTitle( TITLE_CAPTIONBAR, 0, BACKGROUNDCAPTION );
    Enable( BACKGROUND );

    // Welcome the user, check that the system meets minimum requirements,
    // and verify the destination location.
    bInitStepsDone = FALSE;
    while (!bInitStepsDone)
        Disable( BACKBUTTON );
        Welcome( "", 0 );
        Enable( BACKBUTTON );

        TARGETDIR = PROGRAMFILES ^ APPBASE_PATH;
        if (AskDestPath( "", "", TARGETDIR, 0 ) != BACK) then
            bInitStepsDone = TRUE;
        endif;
    endwhile;

    // Set installation information required for registry entries and for
    // the following call to DeinstallStart.

```

```

InstallationInfo( COMPANY_NAME, PRODUCT_NAME,
                 PRODUCT_VERSION, PRODUCT_KEY );

// Initialize the uninstallation log file, including registry entry.
svLogFile = "Uninst.isu";
DeinstallStart( TARGETDIR, svLogFile, DEINSTALL_KEY, 0 );
RegDBSetItem( REGDB_UNINSTALL_NAME, UNINSTALL_NAME );
end;

/*-----*\
*
* Function:  HandleComponentError
*
* Purpose:  This function evaluates the value returned by a Component...
*           function and if the value is less than zero, displays the error
*           number and aborts the setup.
*
*-----*/
function HandleComponentError( nResult )

    NUMBER  nvError;
    STRING  svMedia, svComponent, svFileGroup, svFile;

begin
    if(nResult < 0) then
        ComponentError( svMedia, svComponent, svFileGroup, svFile, nvError );
        sprintfBox( INFORMATION, "Data Transfer Error Information",
                  "ComponentError returned the following data transfer error.\n" +
                  "Setup will now abort.\n\n" +
                  "Media Name: %s\nComponent: %s\nFile Group: %s\n" +
                  "File: %s\nError Number: %ld",
                  svMedia, svComponent, svFileGroup, svFile, nvError );
        abort;
    endif;
end;

/*-----*\
*
* Function:  FinishSetup()
*
* Purpose:  This function finishes the setup. The main reason for
*           abstracting this process into this function is to make it
*           easy to see the function calls this sample script demonstrates.
*
*-----*/
function FinishSetup()

begin
    // Indicate the final percentage the progress bar is to show when the
    // following file transfer operation is complete.
    StatusUpdate(ON, 99);

    // Increment progress bar to 99% for creation of Start Programs menu icon.
    SetStatusWindow (96 , ADDINGICON );

    // Add the APP_NAME icon to the DEFAULT_FOLDER_NAME folder.
    szProgram = TARGETDIR ^ PROGRAMDIR ^ MAIN_EXE;
    LongPathToQuote( szProgram, TRUE );
    AddFolderIcon( DEFAULT_FOLDER_NAME, APP_NAME, szProgram,
                  TARGETDIR ^ PROGRAMDIR,
                  "", 0, "", REPLACE );

    Delay( 1 );

```

```
// Disable the progress indicator and its settings.
Disable ( INDVFILESTATUS );
Disable( STATUSDLG );

// Announce setup complete and offer to view Readme file.
MessageBox( COMPLETE_MSG, INFORMATION );
end;

#define SD_SINGLE_DIALOGS 1
#define SD_SETUPTYPEEX 1
#define SD_COMPONENTDIALOG 1

#include "sddialog.rul"

// Source file: Is5fn603.rul
```

{button ,PI("",`comment')} [Feedback](#)



ComponentFileEnum

Syntax

ComponentFileEnum (szMedia, szComponent, szQuery, listFiles, nOption);

Description

The ComponentFileEnum function builds a list of the files in a file group associated with the specified component.

Parameters

szMedia

Specifies the name of the media.

szComponent

Specifies the name of the component to be enumerated. Do not pass a null string ("") in this parameter; ComponentFileEnum requires a specific component name. For more information about specifying components and subcomponents in function calls, click [here](#).

szQuery

Specifies a file group name and a file specification. Delimit the group name and the file specification with double backslashes and enclose the expression in double quotation marks. The filename part of the file specification may include wildcard characters. The following example specifies all of the files in the file group
Graphic Examples:

```
"Graphic Examples\*.**"
```

listFiles

Returns a string list with all files that match szQuery. Use [ListCreate](#) to create the string list.

nOption

Indicates whether or not to include files in subfolders. Pass one of the following predefined constants in this parameter:

INCLUDE_SUBDIR

Indicates that files in subfolders are to be included.

NO_SUBDIR

Indicates that files in subfolders are *not* to be included.

Return values

0

ComponentFileEnum was successful.

< 0

ComponentFileEnum failed. Call [ComponentError](#) for additional information.

{button ,JI('LANGREF.HLP>Examples','ComponentFileEnum_example')} [Example](#)

{button ,AL(' Script created component set vs. file media library;Specifying components in function calls;Componentfunctions',0,'')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



ComponentFileEnum example

```
/*-----*\
 *
 * Description: Demonstrates ComponentFileEnum.
 *
 * Comments: To run this example script, create a project with the
 *           following components (c), subcomponents (sc), and file
 *           groups (fg):
 *
 *           (c) Program Files
 *               (fg) Program DLLS
 *               (fg) Program EXEs
 *           (c) Example Files
 *               (sc) Graphics
 *                   (fg) Graphic Examples
 *
 *           Be sure to assign files to the file groups so there is
 *           something to enumerate.
 *
 *-----*/
#include "sddialog.h"

#define COMP1      "Program Files"
#define COMP2      "Example Files"
#define SUBCOMP1   "Graphics"
#define EXECGROUP  "Program Executable Files"
#define GRAPHGROUP "Graphic Examples"
#define SDSHOWTITLE "ComponentFileEnum Results"
#define SDSHOWMSG1  COMP1 + " enumerated files:"
#define SDSHOWMSG2  COMP2 + " enumerated files:"

prototype HandleComponentError( NUMBER );

NUMBER nResult;
LIST   listList1, listList2;

program

    listList1 = ListCreate(STRINGLIST);
    listList2 = ListCreate(STRINGLIST);

    // Be sure to associate the correct File group with the
    // appropriate component name.
    nResult = ComponentFileEnum( MEDIA, COMP1, EXECGROUP + "\\*.*",
                                listList1, INCLUDE_SUBDIR );
    HandleComponentError( nResult );

    nResult = ComponentFileEnum( MEDIA, COMP2 + "\\\" + SUBCOMP1,
                                GRAPHGROUP+"\\*.*", listList2, INCLUDE_SUBDIR );
    HandleComponentError( nResult );

    SdShowInfoList( SDSHOWTITLE, SDSHOWMSG1, listList1);
    SdShowInfoList( SDSHOWTITLE, SDSHOWMSG2, listList2);

    ListDestroy(listList1);
    ListDestroy(listList2);

endprogram

/*-----*\
```

```

*
* Function: HandleComponentError
*
* Purpose: This function evaluates the value returned by a Component...
*          function and if the value is less than zero, displays the error
*          number and aborts the setup.
*
\*-----*/
function HandleComponentError( nResult )

    NUMBER  nvError;
    STRING  svMedia, svComponent, svFileGroup, svFile;

begin
    if(nResult < 0) then
        ComponentError( svMedia, svComponent, svFileGroup, svFile, nvError );
        sprintfBox( INFORMATION, "Data Transfer Error Information",
            "ComponentError returned the following data transfer error.\n" +
            "Setup will now abort.\n\n" +
            "Media Name: %s\nComponent: %s\nFile Group: %s\n" +
            "File: %s\nError Number: %ld",
            svMedia, svComponent, svFileGroup, svFile, nvError );
        abort;
    endif;
end;

#include "sddialog.rul"

// Source file: Is5fn601.rul

```

{button ,PI("",`comment')} [Feedback](#)



ComponentFileInfo

Syntax

ComponentFileInfo (szMedia, szComponent, szFile, nInfo, nvResult, svResult);

Description

The ComponentFileInfo function retrieves information on file in the [file media library](#) that is referenced by szMedia.



This function cannot be used with script-created components.

Parameters

szMedia

Specifies the name of the media.

szComponent

Specifies the component containing the file group in which the file is found. For more information about specifying components and subcomponents in function calls, click [here](#).

szFile

Specifies the file group name and the file name, with or without a path. Delimit tokens in the szFile expression with double backslashes and enclose the expression in double quotation marks.

If there are two tokens in szFile, the first is the file group name and the second is the file name:

"file group\file name"

If there are more than two tokens in szFile, the first is the file group name, the last is the file name, and the middle tokens form the path:

"file group\path\file name"

InstallShield lets you add entire file and folder structures to your file groups by dragging and dropping them into the Files window. When you want to refer to a file nested inside a file and folder structure in a file group, you must specify its path location under the Links folder in that file group.

For example:

szFile	Meaning
"Shared Files\ls5.dll"	Shared Files is a file group and ls5.dll is a file name.
"Shared Files\dev\myapp\dlls\ls5.dll"	Shared Files is a filegroup, \dev\myapp\dlls is a path, and ls5.dll is a file name.

nInfo

Specifies the type of information to be retrieved:

COMPONENT_INFO_LANGUAGE (file group only)

Language setting for the specified file group. Since this applies to a file group only, szFile must be a single token expression identifying a file group.

COMPONENT_INFO_OS (file group only)

OS setting for the specified file group. Since this applies to a file group only, szFile must be a single token expression identifying a file group.

COMPONENT_INFO_ORIGSIZE

Original size of the specified file.

COMPONENT_INFO_COMPsize

Compressed file size for the specified file in the file media library.

COMPONENT_INFO_DATE

Date value for the specified file in the format mm-dd-yy.

COMPONENT_INFO_DATE_EX

Date value for the specified file in the 2000 compliant format mm-dd-yyyy.

COMPONENT_INFO_TIME

Time value for the specified file in the format hh:mm, using 24-hour time.

COMPONENT_INFO_ATTRIBUTE

Attribute values for the specified file.

COMPONENT_INFO_VERSIONMS

The major version number(s) in string form.

COMPONENT_INFO_VERSIONLS

The minor version number(s) in string form.

COMPONENT_INFO_VERSIONSTR

The entire version number in string form.

nvResult

Returns a number value when nInfo is one of the following: COMPONENT_INFO_ORIGSIZE, COMPONENT_INFO_COMPsize, or COMPONENT_INFO_ATTRIBUTE.

When nInfo is COMPONENT_INFO_LANGUAGE or COMPONENT_INFO_OS, the parameter position normally occupied by nvResult must instead specify a LIST variable to store the language or operating system IDs. You must call [ListCreate](#) with the constant NUMBERLIST to create the list before you call ComponentFileInfo to determine which languages or operating systems are specified by the component:

```
listID = ListCreate(NUMBERLIST);
ComponentFileInfo(szMedia, szComponent, szFile, COMPONENT_INFO_LANGUAGE,
listID, svResult);
```

When nInfo is COMPONENT_INFO_LANGUAGE, ComponentFileInfo adds [language IDs](#) to the list; when nInfo is COMPONENT_INFO_OS, ComponentFileInfo adds [operating system IDs](#) to the list.



When nInfo is COMPONENT_INFO_ATTRIBUTE, nvResult return with a numeric value that represents the [file attributes](#). To determine the file attribute settings, use file attribute constants in bitwise OR operations with nvResult. For an example of how to test nvResult, see the [example script](#).

svResult

Returns the date, time or, version when nInfo specifies a corresponding information type.

Return values

0

ComponentFileInfo was successful.

< 0

ComponentFileInfo failed. Call [ComponentError](#) for additional information.

{button ,JI('LANGREF.HLP>Examples',`ComponentFileInfo_example`)} [Example](#)

{button ,AL(`Script created component set vs. file media library;Specifying components in function calls;Componentfunctions',0,`,`)} [See also](#)

{button ,PI('`,`comment')} [Feedback](#)



ComponentFileInfo example

```
/*-----*\
 *
 * Description: Demonstrates ComponentFileInfo.
 *
 * Comments: To run this example script, create a project with the
 *           following components (c), subcomponents (sc), and file
 *           groups (fg):
 *
 *           (c) Example Files
 *           (sc) Graphics
 *           (fg) Graphic Examples
 *
 *           Be sure to assign at least one file to the Graphic Examples
 *           file group so there is something to enumerate. Remember
 *           also to specify the filename in the #define FILE line.
 *
 *-----*/
#include "sddialog.h"

#define COMP      "Example Files"
#define SUBCOMP   "Graphics"
#define FILEGRP   "Graphic Examples"
#define FILE      "comdlg32.dll"
#define SDSHOWTITLE "ComponentFileInfo Results"
#define SDSHOWMSG  FILE + " information:"

prototype HandleComponentError( NUMBER );

NUMBER nReturn, nvResult;
STRING svResult;
LIST listID;

program

    listID = ListCreate( STRINGLIST );

    // Call ComponentFileInfo repeatedly to retrieve various pieces of information
    // about FILE. Store each result in a string list for display.
    nReturn = ComponentFileInfo( MEDIA, COMP + "\\\" + SUBCOMP,
                                FILEGRP + "\\\" + FILE,
                                COMPONENT_INFO_ORIGSIZE , nvResult, svResult );
    HandleComponentError( nvResult );
    Sprintf( svResult, "%d", nvResult );
    ListAddString( listID, "The Original size of the file is      " +
                    svResult, AFTER );

    nReturn = ComponentFileInfo( MEDIA, COMP + "\\\" + SUBCOMP,
                                FILEGRP + "\\\" + FILE,
                                COMPONENT_INFO_TIME , nvResult, svResult );
    HandleComponentError( nvResult );
    ListAddString( listID, "The modified time for the file is    " +
                    svResult, AFTER );

    nReturn = ComponentFileInfo( MEDIA, COMP + "\\\" + SUBCOMP,
                                FILEGRP + "\\\" + FILE,
                                COMPONENT_INFO_DATE , nvResult, svResult );
    HandleComponentError( nvResult );
    ListAddString( listID, "The modified date for the file is    " +
                    svResult, AFTER );
```

```

nReturn = ComponentFileInfo( MEDIA, COMP + "\\\" + SUBCOMP,
                             FILEGRP + "\\\" + FILE,
                             COMPONENT_INFO_ATTRIBUTE , nvResult, svResult );
HandleComponentError( nReturn );
// Prepare a string listing the attributes. If normal (no attributes),
// set svResult to "normal".

if ( nvResult = FILE_ATTR_NORMAL ) then
    svResult = "normal";
// If attributes are set, concatenate them into a string.
else
    if ( FILE_ATTR_ARCHIVED & nvResult ) then
        svResult = "archived,";
    endif;

    if ( FILE_ATTR_HIDDEN & nvResult ) then
        svResult = svResult + "hidden,";
    endif;

    if ( FILE_ATTR_READONLY & nvResult ) then
        svResult = svResult + "read-only,";
    endif;

    if ( FILE_ATTR_SYSTEM & nvResult ) then
        svResult = svResult + "system,";
    endif;

    if ( FILE_ATTR_DIRECTORY & nvResult ) then
        svResult = svResult + "directory,";
    endif;
endif;
ListAddString( listID, "The attribute for the file is          " +
                svResult, AFTER );

nReturn = ComponentFileInfo( MEDIA, COMP + "\\\" + SUBCOMP,
                             FILEGRP + "\\\" + FILE,
                             COMPONENT_INFO_VERSIONMS , nvResult, svResult );
HandleComponentError( nReturn );
ListAddString( listID, "The upper 32-bit version value        " +
                svResult, AFTER );

nReturn = ComponentFileInfo( MEDIA, COMP + "\\\" + SUBCOMP,
                             FILEGRP + "\\\" + FILE,
                             COMPONENT_INFO_VERSIONLS , nvResult, svResult );
HandleComponentError( nReturn );
ListAddString( listID, "The lower 32-bit version value is     " +
                svResult, AFTER );

nReturn = ComponentFileInfo( MEDIA, COMP + "\\\" + SUBCOMP,
                             FILEGRP + "\\\" + FILE,
                             COMPONENT_INFO_VERSIONSTR , nvResult, svResult );
HandleComponentError( nReturn );
ListAddString( listID, "The version for the file is          " +
                svResult, AFTER );

// Display the results.
SdShowInfoList( SDSHOWTITLE, SDSHOWMSG, listID );

ListDestroy(listID);

endprogram

```

```

/*-----*\
*
* Function:  HandleComponentError
*
* Purpose:  This function evaluates the value returned by a Component...
*           function and if the value is less than zero, displays the error
*           number and aborts the setup.
*
\*-----*/
function HandleComponentError( nvResult )

    NUMBER  nvError;
    STRING  svMedia, svComponent, svFileGroup, svFile;

begin
    if(nvResult < 0) then
        ComponentError( svMedia, svComponent, svFileGroup, svFile, nvError );
        sprintfBox( INFORMATION, "Data Transfer Error Information",
            "ComponentError returned the following data transfer error.\n" +
            "Setup will now abort.\n\n" +
            "Media Name: %s\nComponent: %s\nFile Group: %s\n" +
            "File: %s\nError Number: %ld",
            svMedia, svComponent, svFileGroup, svFile, nvError );
        abort;
    endif;
end;

#include "sddialog.rul"

// Source file: Is5fn612.rul

```

{button ,PI("",`comment')} [Feedback](#)



ComponentGetData

Syntax

ComponentGetData (szMedia, szComponent, nInfo, nvResult, svResult);

Description

The ComponentGetData function retrieves information about a component. When szMedia refers to a script-created component set, some information cannot be retrieved.

Parameters

szMedia

Specifies the name of the media.

szComponent

Specifies the name of the component whose information is to be retrieved. For more information about specifying components and subcomponents in function calls, click [here](#).

nInfo

Specifies the type of information to retrieve. Pass one of the following predefined constants in this parameter:

COMPONENT_FIELD_DESCRIPTION

The description displayed when the component is selected in a component selection dialog. For a component in a [file media library](#), this is the value stored in the Description field of the Component Properties window.

COMPONENT_FIELD_FILENEED

Defines how critical the component files are for the installation. One of the following values will be set in nvResult:

COMPONENT_VALUE_CRITICAL

This component contains critical files.

COMPONENT_VALUE_HIGHLYRECOMMENDED

This component is highly recommended.

COMPONENT_VALUE_STANDARD

This may be included or not included.

COMPONENT_FIELD_FTPLOCATION

The contents of the FTP Location properties specified in the Components pane.

COMPONENT_FIELD_HTTPLOCATION

The contents of the HTTP Location properties specified in the Components pane.

COMPONENT_FIELD_STATUS (not for script-created components)

The status text displayed in the progress indicator during file transfer. This is the value stored in the Status Text field of the Component Properties window.

COMPONENT_FIELD_VISIBLE

Determines whether or not the component is visible in a component selection dialog. For a component in a file media library, this is the value stored in the Visible field of the Component Properties window. One of the following values will be set in nvResult:

TRUE

This component is visible.

FALSE

This component is not visible.

COMPONENT_FIELD_OVERWRITE (not for script-created components)

How or whether to overwrite a file of the same name on the target system. This is the value in the Overwrite field of the Component Properties window. One or more of the following values will be set in nvResult:

COMPONENT_VALUE_ALWAYSoverwrite
COMPONENT_VALUE_SAMEORNEWDATE
COMPONENT_VALUE_NEWERDATE
COMPONENT_VALUE_OLDERDATE
COMPONENT_VALUE_SAMEORNEWERVERSION
COMPONENT_VALUE_NEWERVERSION
COMPONENT_VALUE_OLDERVERSION
COMPONENT_VALUE_NEVERoverwrite

You can test for each of the above values in `nvResult` using a logical AND (&) operation, as shown below:

```
if( COMPONENT_VALUE_NEWERDATE & nvResult) then
    // COMPONENT_VALUE_NEWERDATE was set
endif;

if( COMPONENT_VALUE_SAMEORNEWERVERSION & nvResult) then
    // COMPONENT_VALUE_SAMEORNEWERVERSION was set
endif;
```

COMPONENT_FIELD_DESTINATION (not for script-created components)

The destination location for the files associated with the component. This is the value stored in the Destination field of the Component Properties window.

COMPONENT_FIELD_PASSWORD (not for script-created components)

Whether or not there is a password associated with the component (in the Password field of the Component Properties window. One of the following values will be set in `nvResult`:

TRUE

There is a password for this component. If the component is password protected, you must get the correct password from the end user and validate it with `ComponentValidate` before calling `ComponentMoveData` to transfer the files in the file media library.

FALSE

There is no password for this component.

COMPONENT_FIELD_SELECTED

Determines whether or not the component is selected:

TRUE

This component is selected.

FALSE

This component is not selected.

COMPONENT_FIELD_SIZE (not for file media)

Total original file size for the specified component. You can also use `ComponentGetItemSize` to determine the size of a component (not including subcomponents). Use `ComponentTotalSize` to determine the total size of all selected components and subcomponents.

COMPONENT_FIELD_MISC

Miscellaneous text. This field can be very useful at run time, since you can use it to flag or identify components using any information you wish.

COMPONENT_FIELD_DISPLAYNAME

The component name displayed in the component selection dialogs. For components in file media libraries, this is the value in the Display Name field in the Component Properties window.

COMPONENT_FIELD_CDROM_FOLDER (not for script-created components)

For CD-ROM build type only. The location of the uncompressed files associated with the specified component when "Data as files" is checked in the Media Build Wizard's Disk Type panel.

nvResult

Returns a number value when `nInfo` produces that type of result.

svResult

Returns a string value when `nInfo` produces that type of result.

Return values

0

ComponentGetData was successful.

< 0

ComponentGetData failed. Call [ComponentError](#) for additional information.

{button ,JI('LANGREF.HLP>Examples','ComponentGetData_example')} [Example](#)

{button ,AL('Script created component set vs. file media library;Specifying components in function calls;Componentfunctions',0,'')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



ComponentGetData example

```
/*-----*\
 *
 * This example demonstrates ComponentSetupTypeGetData, ComponentGetData,
 * ComponentSetData, SdComponentDialog2, and ComponentSelectedItem.
 *
 * Comments: To run this example script, create a project with the
 *           following components (c), subcomponents (sc), and file
 *           groups (fg):
 *
 *           (c) Program Files
 *               (fg) Program DLLS
 *               (fg) Program EXEs
 *           (c) Example Files
 *               (sc) Small Documents
 *                   (fg) Small Document Examples
 *               (sc) Books
 *                   (fg) Book Examples
 *               (sc) Graphics
 *                   (fg) Graphic Examples
 *           (c) Help Files
 *               (fg) Help Files
 *           (c) Utilities
 *               (sc) Grammar Checker
 *                   (fg) Grammar Checker
 *               (sc) Art Studio
 *                   (fg) Art Studio
 *
 * Be sure to enter descriptions into the Description fields of the component
 * properties sheets for the Program Files and Example Files components and
 * their subcomponents.
 *
\*-----*/

#include "sddialog.h"

// Define strings. In a real setup you would define these in your string tables
// and precede each constant with @ to use them in your script.
#define COMP_SELECT_TITLE           "Select Components"
#define COMP_SELECT_MSG1           "IMPORTANT! Note the various component "
#define COMP_SELECT_MSG2           "and subcomponent names, descriptions, "
#define COMP_SELECT_MSG3           "and selection settings."
#define COMP_SELECT_MSG4           "IMPORTANT! Note the CHANGED component "
#define COMP_SELECT_MSG5           "and subcomponent name, description, "
#define COMP_SELECT_MSG6           "and selection settings."
#define COMP_PROGRAMFILES_DISPLAYNAME "Program Files"
#define COMP_EXAMPLEFILES_DISPLAYNAME "Example Files"
#define COMP_SMALLDOCUMENTS_DISPLAYNAME "Small Documents"
#define COMP_BOOKS_DISPLAYNAME      "Books"
#define COMP_GRAPHICS_DISPLAYNAME   "Graphics"
#define SETUP_TYPE                  "Typical"

// Global variable declarations.
STRING  svInfo, szInfo, szComponent;
NUMBER  nvInfo, nInfo, nResult;

program

    // Get the description field data for the SETUP_TYPE setup type.
```

```

ComponentSetupTypeGetData( MEDIA, SETUP_TYPE, SETUPTYPE_INFO_DESCRIPTION,
                           nvInfo, svInfo );
sprintfBox( INFORMATION, "ComponentSetupTypeGetData demo",
           "ComponentSetupTypeGetData got the following " +
           "value from the " + SETUP_TYPE + " description field:\n\n%s",
           svInfo );

// Get the description field data for the COMP_PROGRAMFILES_DISPLAYNAME
// component using ComponentGetData.
szComponent = COMP_PROGRAMFILES_DISPLAYNAME;
nResult = ComponentGetData( MEDIA, szComponent, COMPONENT_FIELD_DESCRIPTION,
                           nvInfo, svInfo );
sprintfBox( INFORMATION, "ComponentGetData demo",
           "ComponentGetData got the following value " +
           "from the " + COMP_PROGRAMFILES_DISPLAYNAME +
           " description field:\n\n%s", svInfo );

// Show the original description field values in the component selection dialog.
Disable( BACKBUTTON); // Back button not needed or handled here.
SdComponentDialog2( COMP_SELECT_TITLE, COMP_SELECT_MSG1 + COMP_SELECT_MSG2 +
                   COMP_SELECT_MSG3, TARGETDIR, "" );

// Change the displayed names for the Program Files component and the
// Example Files component and subcomponents.
szInfo = "CHANGED Component Name!";
nResult = ComponentSetData( MEDIA, szComponent, COMPONENT_FIELD_DISPLAYNAME,
                           nInfo, szInfo );
szComponent = COMP_EXAMPLEFILES_DISPLAYNAME;
nResult = ComponentSetData( MEDIA, szComponent, COMPONENT_FIELD_DISPLAYNAME,
                           nInfo, szInfo );
szComponent = COMP_EXAMPLEFILES_DISPLAYNAME + "\\\" +
              COMP_SMALLDOCUMENTS_DISPLAYNAME;
nResult = ComponentSetData( MEDIA, szComponent, COMPONENT_FIELD_DISPLAYNAME,
                           nInfo, szInfo );
szComponent = COMP_EXAMPLEFILES_DISPLAYNAME + "\\\" +
              COMP_BOOKS_DISPLAYNAME;
nResult = ComponentSetData( MEDIA, szComponent, COMPONENT_FIELD_DISPLAYNAME,
                           nInfo, szInfo );
szComponent = COMP_EXAMPLEFILES_DISPLAYNAME + "\\\" +
              COMP_GRAPHICS_DISPLAYNAME;
nResult = ComponentSetData( MEDIA, szComponent, COMPONENT_FIELD_DISPLAYNAME,
                           nInfo, szInfo );

// Change the descriptions displayed for the Program Files component
// and the Example Files component and subcomponents.
szComponent = COMP_PROGRAMFILES_DISPLAYNAME;
szInfo = "CHANGED description field value!";
nResult = ComponentSetData( MEDIA, szComponent, COMPONENT_FIELD_DESCRIPTION,
                           nInfo, szInfo );
szComponent = COMP_EXAMPLEFILES_DISPLAYNAME;
nResult = ComponentSetData( MEDIA, szComponent, COMPONENT_FIELD_DESCRIPTION,
                           nInfo, szInfo );
szComponent = COMP_EXAMPLEFILES_DISPLAYNAME + "\\\" +
              COMP_SMALLDOCUMENTS_DISPLAYNAME;
nResult = ComponentSetData( MEDIA, szComponent, COMPONENT_FIELD_DESCRIPTION,
                           nInfo, szInfo );
szComponent = COMP_EXAMPLEFILES_DISPLAYNAME + "\\\" +
              COMP_BOOKS_DISPLAYNAME;
nResult = ComponentSetData( MEDIA, szComponent, COMPONENT_FIELD_DESCRIPTION,
                           nInfo, szInfo );
szComponent = COMP_EXAMPLEFILES_DISPLAYNAME + "\\\" +
              COMP_GRAPHICS_DISPLAYNAME;
nResult = ComponentSetData( MEDIA, szComponent, COMPONENT_FIELD_DESCRIPTION,

```

```
        nInfo, szInfo );

// Deselect the Program Files and Example Files components (and all their
// subcomponents, by extension).
ComponentSelectItem( MEDIA, COMP_PROGRAMFILES_DISPLAYNAME, FALSE );
ComponentSelectItem( MEDIA, COMP_EXAMPLEFILES_DISPLAYNAME, FALSE );

// Display the components again, noting the changed names, descriptions,
// and selection settings.
SdComponentDialog2( COMP_SELECT_TITLE, COMP_SELECT_MSG4 + COMP_SELECT_MSG5 +
                    COMP_SELECT_MSG6, TARGETDIR, "" );
endprogram

#include "sddialog.rul"

// Source file: IS5FN623.rul
```

{button ,PI("",`comment')} [Feedback](#)



ComponentGetItemSize

Syntax

ComponentGetItemSize (szMedia, szComponent, nvSize);

Description

The ComponentGetItemSize function retrieves the size in bytes of a specified component. The sizes of subcomponents are not included.

Parameters

szMedia

Specifies the name of the media.

szComponent

Specifies the name of the component whose size is to be retrieved. For more information about specifying components and subcomponents in function calls, click [here](#).

nvSize

Returns the size in bytes of the specified component. You can also use ComponentGetData to get the total original file size for a specified component. Use ComponentTotalSize to determine the total size of all selected components and subcomponents.

Return values

0

ComponentGetItemSize was successful.

< 0

ComponentGetItemSize failed. Call [ComponentError](#) for additional information.

{button ,JI('LANGREF.HLP>Examples','ComponentGetItemSize_example')} [Example](#)

{button ,AL(' Script created component set vs. file media library;Specifying components in function calls;ComponentAddItem;ComponentDialog;ComponentIsItemSelected;ComponentListItems;ComponentSelectItem ;SdComponentDialog;SdComponentDialogAdv',0,'')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



ComponentGetItemSize example

```
/*-----*\
 *
 * This example uses ComponentGetItemSize to get the size of a component and
 * a subcomponent. The sizes are displayed in an SdShowInfoList dialog.
 *
 * Comments: To run this example script, create a project (or insert into
 * a project) with several components and/or subcomponents with
 * file groups containing files. Either name one component and
 * one subcomponent as indicated in the #define statements in
 * this example, or change the #define statements to reflect
 * your component names.
 *
 *-----*/

#include "sddialog.h"

#define COMP_NAME1 "Program Files"
#define COMP_NAME2 "Example Files\\Graphics"

NUMBER nvSize;
STRING szString1, szString2;
LIST listInfo;

program

    // Create a string list for display in the SdShowInfoList dialog.
    listInfo = ListCreate(STRINGLIST);

    // Get the size of COMP_NAME1, convert it to a string, and put it
    // into the string list.
    ComponentGetItemSize(MEDIA, COMP_NAME1, nvSize);
    NumToStr(szString1, nvSize);
    ListAddString(listInfo, "The size in bytes of " + COMP_NAME1 +
        " is: " + szString1, AFTER);

    // Get the size of COMP_NAME2, convert it to a string, and put it
    // into the string list.
    ComponentGetItemSize(MEDIA, COMP_NAME2, nvSize);
    NumToStr(szString2, nvSize);
    ListAddString(listInfo, "The size in bytes of " + COMP_NAME2 +
        " is: " + szString2, AFTER);

    // Display the component sizes.
    SdShowInfoList("Results of Calls to ComponentGetItemSize",
        "The component sizes are:", listInfo);

    ListDestroy(listInfo);

endprogram

#include "sddialog.rul"

// Source file: Is5fn602.rul
```



ComponentIsItemSelected

Syntax

ComponentIsItemSelected (szMedia, szComponent);

Description

The ComponentIsItemSelected function determines whether a specific component is selected. The component is usually one that was selected by the end user in one of the component selection dialogs.

Parameters

szMedia

Specifies the name of the media with the component whose selection setting is to be check.

szComponent

Specifies the name of the component to be checked. For more information about specifying components and subcomponents in function calls, click [here](#).

Return values

TRUE

szComponent is selected.

FALSE

szComponent is not selected.

< 0

The function failed to determine if the component was selected. Call [ComponentError](#) for additional information.



You can also use [ComponentGetData](#) to determine if a component is selected.

{button ,JI('LANGREF.HLP>Examples','ComponentIsItemSelected_example')} [Example](#)

{button ,AL('Script created component set vs. file media library;Specifying components in function calls;ComponentAddItem;ComponentDialog;ComponentGetItemSize;ComponentListItems;ComponentSelectItem;SdComponentDialog;SdComponentDialogAdv',0,',')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



ComponentIsItemSelected example

```
/*-----*\
*
* This example demonstrates SdSetupTypeEx, SdComponentDialog,
* ComponentIsItemSelected, ComponentGetData, ComponentValidate,
* ComponentMoveData, ComponentError, and PlaceWindow.
*
* Comments: To run this example script, create a project with the
*           following components (c), subcomponents (sc), and file
*           groups (fg):
*
*           (c) Program Files
*               (fg) Program DLLS
*               (fg) Program EXEs
*           (c) Example Files
*               (sc) Small Documents
*                   (fg) Small Document Examples
*               (sc) Books
*                   (fg) Book Examples
*               (sc) Graphics
*                   (fg) Graphic Examples
*           (c) Help Files
*               (fg) Help Files
*           (c) Utilities
*               (sc) Grammar Checker
*                   (fg) Grammar Checker
*               (sc) Art Studio
*                   (fg) Art Studio
*           (c) Evaluation Copy
*               (fg) Evaluation Copy
*               (fg) Help Files
*
* Insert "dummy" files into the file groups. Make sure you define the
* correct filename for the main EXE (MAIN_EXE, below) that you insert into
* the Program EXEs file group. Run the setup with and without a password
* assigned to the Program Files component (remember to rebuild your media
* each time).
*
* You can also create billboards (name them Bbrd1.bmp, Bbrd2.bmp, and
* Bbrd3.bmp) and add them to the project in the Setup Files pane under
* the Language Independent folder.
*
* This example script installs files, adds an icon to the Start Programs
* menu, and provides uninstallation functionality.
*
\*-----*/

#include "sddialog.h"

// Define strings. In a real setup you would define these in your string tables
// and precede each constant with @ to use them in your script.
#define COMP_SELECT_TITLE           "Select Components"
#define COMP_SELECT_MSG             "Select components and subcomponents to
install."
#define COMP_PROGRAMFILES_DISPLAYTEXT "Program Files"
#define PASSWORD_PROMPT             "Please enter the password."
#define PASSWORD_ERRMSG             "Password incorrect. Please enter again."
#define TITLE_MAIN                  "Word Processor"
#define TITLE_CAPTIONBAR            "Word Processor Setup"
#define APPBASE_PATH                "Your Company\\Word Processor"
```

```

#define COMPANY_NAME           "Your Company"
#define PRODUCT_NAME          "Word Processor"
#define PRODUCT_VERSION       "1.0"
#define PRODUCT_KEY           "Word Processor"
#define DEINSTALL_KEY        "Word Processor"
#define UNINSTALL_NAME       "Word Processor"
#define ADDINGICON            "Adding program icon to the Start Programs
menu..."
#define PROGRAMDIR            "Program"
#define DEFAULT_FOLDER_NAME   ""
#define APP_NAME              "Word Processor"
#define COMPLETE_MSG          "Setup is complete. You can run Word
Processor from the Start Programs menu."
#define MAIN_EXE              "WRITE.EXE"
#define SETUPTYPE_TITLE       "Setup Type Selection"
#define SETUPTYPE_MSG         "Please select a setup type."
#define SETUPTYPE_CUSTOM      "Custom"

```

```

// Global variable declarations.
STRING svData, svLogFile, szProgram, szComponent, svResult, svSetupType, svDir;
BOOL    bInitStepsDone, bPwdValid;
NUMBER  nvData, nvDisk, nResult;

```

```

// Function declarations.
prototype SetUpFileTransfer();
prototype HandleComponentError(NUMBER);
prototype FinishSetup();

```

```

program
    SetUpFileTransfer();

    // Get the setup type.
    Disable( BACKBUTTON );
    svDir = TARGETDIR;
    SdSetupTypeEx( SETUPTYPE_TITLE, SETUPTYPE_MSG, "", svSetupType, 0 );
    // If user selected Custom setup type, display component selection dialog.
    if ( svSetupType = SETUPTYPE_CUSTOM ) then
        SdComponentDialog( COMP_SELECT_TITLE, COMP_SELECT_MSG, svDir, "" );
    endif;
    Enable( BACKBUTTON );

    // If the Program Files component is selected and there is a password
    // associated with it, the user must input the password and
    // it must be properly validated.
    nResult = FALSE;
    nvData = FALSE;
    nResult = ComponentIsItemSelected( MEDIA, COMP_PROGRAMFILES_DISPLAYTEXT );
    ComponentGetData( MEDIA, COMP_PROGRAMFILES_DISPLAYTEXT,
        COMPONENT_FIELD_PASSWORD, nvData, svData);
    if ( nResult && nvData ) then
        bPwdValid = FALSE;
        Disable( BACKBUTTON ); // Back button not needed or supported here.
        while ( !bPwdValid )
            AskText( PASSWORD_PROMPT, "", svResult );
            nResult = ComponentValidate( MEDIA, COMP_PROGRAMFILES_DISPLAYTEXT,
                svResult );

            if ( nResult = 0 ) then
                bPwdValid = TRUE;
            else
                MessageBox( PASSWORD_ERRMSG, SEVERE );
            endif;
        endwhile;
    endif;
endprogram

```

```

    Enable( BACKBUTTON ); // Restore Back button's default status.
endif;

// Set up the progress indicator, including locations for
// progress indicator, information gauges, and billboards.
PlaceWindow( FEEDBACK, LOWER_LEFT, LOWER_LEFT, LOWER_LEFT );
PlaceWindow( STATUSDLG, CENTERED, LOWER_RIGHT, LOWER_RIGHT );
PlaceWindow( BILLBOARD, CENTERED, CENTERED, CENTERED );
Enable( STATUSDLG );
Enable( INDVFILESTATUS );

// Indicate the final percentage the progress bar is to show when the
// following file transfer operation is complete.
StatusUpdate( ON, 95 );

// Transfer files to the target system. ComponentMoveData will prompt
// for next disk in a floppy disk installation.
nResult=ComponentMoveData( MEDIA, nvDisk, 0);

// To see the ComponentError function in action in the following call
// to the HandleComponentError user-defined function, remark out the
// while...endwhile loop above in which ComponentValidate is called,
// make sure a password is associated with the Program Files component
// in the IDE, rebuild the media, and run the setup.
HandleComponentError( nResult );

FinishSetup();
endprogram

/*-----*\
*
* Function: SetupFileTransfer()
*
* Purpose: This function sets up file transfer. The main reason for
*          abstracting this process into this function is to make it
*          easy to see the function calls this sample script demonstrates.
*
*\-----*/
function SetUpFileTransfer()
begin
    // Set up the installation screen.
    Enable( FULLWINDOWMODE );
    SetTitle( TITLE_MAIN, 24, WHITE );
    SetTitle( TITLE_CAPTIONBAR, 0, BACKGROUNDCAPTION );
    Enable( BACKGROUND );

    // Welcome the user, check that the system meets minimum requirements,
    // and verify the destination location.
    bInitStepsDone = FALSE;
    while (!bInitStepsDone)
        Disable( BACKBUTTON );
        Welcome( "", 0 );
        Enable( BACKBUTTON );

        TARGETDIR = PROGRAMFILES ^ APPBASE_PATH;
        if (AskDestPath( "", "", TARGETDIR, 0 ) != BACK) then
            bInitStepsDone = TRUE;
        endif;
    endwhile;

    // Set installation information required for registry entries and for
    // the following call to DeinstallStart.

```

```

InstallationInfo( COMPANY_NAME, PRODUCT_NAME,
                 PRODUCT_VERSION, PRODUCT_KEY );

// Initialize the uninstallation log file, including registry entry.
svLogFile = "Uninst.isu";
DeinstallStart( TARGETDIR, svLogFile, DEINSTALL_KEY, 0 );
RegDBSetItem( REGDB_UNINSTALL_NAME, UNINSTALL_NAME );
end;

/*-----*\
*
* Function:  HandleComponentError
*
* Purpose:  This function evaluates the value returned by a Component...
*           function and if the value is less than zero, displays the error
*           number and aborts the setup.
*
*-----*/
function HandleComponentError( nResult )

    NUMBER  nvError;
    STRING  svMedia, svComponent, svFileGroup, svFile;

begin
    if(nResult < 0) then
        ComponentError( svMedia, svComponent, svFileGroup, svFile, nvError );
        sprintfBox( INFORMATION, "Data Transfer Error Information",
                  "ComponentError returned the following data transfer error.\n" +
                  "Setup will now abort.\n\n" +
                  "Media Name: %s\nComponent: %s\nFile Group: %s\n" +
                  "File: %s\nError Number: %ld",
                  svMedia, svComponent, svFileGroup, svFile, nvError );
        abort;
    endif;
end;

/*-----*\
*
* Function:  FinishSetup()
*
* Purpose:  This function finishes the setup. The main reason for
*           abstracting this process into this function is to make it
*           easy to see the function calls this sample script demonstrates.
*
*-----*/
function FinishSetup()

begin
    // Indicate the final percentage the progress bar is to show when the
    // following file transfer operation is complete.
    StatusUpdate(ON, 99);

    // Increment progress bar to 99% for creation of Start Programs menu icon.
    SetStatusWindow (96 , ADDINGICON );

    // Add the APP_NAME icon to the DEFAULT_FOLDER_NAME folder.
    szProgram = TARGETDIR ^ PROGRAMDIR ^ MAIN_EXE;
    LongPathToQuote( szProgram, TRUE );
    AddFolderIcon( DEFAULT_FOLDER_NAME, APP_NAME, szProgram,
                  TARGETDIR ^ PROGRAMDIR,
                  "", 0, "", REPLACE );

    Delay( 1 );

```

```
// Disable the progress indicator and its settings.
Disable ( INDVFILESTATUS );
Disable( STATUSDLG );

// Announce setup complete and offer to view Readme file.
MessageBox( COMPLETE_MSG, INFORMATION );
end;

#define SD_SINGLE_DIALOGS 1
#define SD_SETUPTYPEEX 1
#define SD_COMPONENTDIALOG 1

#include "sddialog.rul"

// Source file: Is5fn603.rul
```

{button ,PI("",`comment')} [Feedback](#)



ComponentListItems

Syntax

ComponentListItems (szMedia, szComponent, listComponents);

Description

The ComponentListItems function lists all components under szComponent in the [file media library](#) or script-created component set referenced by szMedia. The list of fully qualified subcomponent names is stored in listComponents. If szComponent has no children, listComponents will be an empty list.

Parameters

szMedia

Specifies the name of the media that contains the component whose subcomponents are to be listed.

szComponent

Specifies the component whose subcomponents are to be listed. Pass a null string ("") in this parameter to list all top-level components. For more information about specifying components and subcomponents in function calls, click [here](#).

listComponents

Returns the list of Components. Create this string list with [ListCreate](#) before calling ComponentListItems .

Return values

0

ComponentListItems listed the components.

< 0

ComponentListItems was unable to list the components. Call [ComponentError](#) for additional information.

{button ,JI('LANGREF.HLP>Examples','ComponentListItems_example')}

[Example](#)

{button ,AL(' Script created component set vs. file media library;Specifying components in function calls;ComponentAddItem;ComponentDialog;ComponentGetItemSize;ComponentSelectItem;ComponentIsItemSelected;SdComponentDialog;SdComponentDialogAdv',0,'')} [See also](#)

{button ,PI('',`comment`)} [Feedback](#)



ComponentListItems example

```
/*-----*\
*
* This example demonstrates ComponentListItems, SdComponentMult,
* and ComponentTotalSize.
*
* Comments: To run this example script, create a project with the
*           following components (c), subcomponents (sc), and file
*           groups (fg):
*
*           (c) Program Files
*               (fg) Program DLLS
*               (fg) Program EXEs
*           (c) Example Files
*               (sc) Small Documents
*                   (fg) Small Document Examples
*               (sc) Books
*                   (fg) Book Examples
*               (sc) Graphics
*                   (fg) Graphic Examples
*           (c) Help Files
*               (fg) Help Files
*           (c) Utilities
*               (sc) Grammar Checker
*                   (fg) Grammar Checker
*               (sc) Art Studio
*                   (fg) Art Studio
*
*-----*/

#include "sddialog.h"

#define COMP_SELECT_TITLE      "Select Components"
#define COMP_SELECT_MSG      "Select components and subcomponents to install."
#define COMPTOTSIZEMSG1      "Want to change component selections and see\n"
#define COMPTOTSIZEMSG2      "size change reflected in ComponentTotalSize call?"

// Global variable declarations.
STRING  szDir, svString;
NUMBER  nResult, nDone;
LIST    listCompList, listTemp;

program
    // Disable Back button, which is not needed.
    Disable( BACKBUTTON );

    // Create a string list of all top-level components.
    listCompList = ListCreate( STRINGLIST );
    ComponentListItems( MEDIA, "", listCompList );
    // Display the string list of top-level components.
    SdShowInfoList( "List MEDIA Components", "MEDIA contains " +
        "the following top-level components:", listCompList );

    // Get each top-level component in listCompList, in turn, and
    // list and display all its subcomponents, if any.
    nResult = ListGetFirstString( listCompList, svString );
    while ( nResult != END_OF_LIST )
        listTemp = ListCreate( STRINGLIST );
        ComponentListItems( MEDIA, svString, listTemp );
        SdShowInfoList( "Subcomponent Listing", svString + " contains " +
```

```
        "the following subcomponents:", listTemp );
    ListDestroy( listTemp );
    nResult = ListGetNextString( listCompList, svString );
endwhile;

// Show component selection dialog and total size of all selected
// components. Loop to change selections and see total size change
// reflected in the call to ComponentTotalSize.
nDone = YES;
while ( nDone = YES )
    szDir = TARGETDIR;
    SdComponentMult( COMP_SELECT_TITLE, COMP_SELECT_MSG, szDir, "" );

    nResult = ComponentTotalSize( MEDIA, "", TRUE, TRUE );
    sprintfBox( INFORMATION, "", "Total size of all files " +
        "in SELECTED components:\n\n%d", nResult );
    nDone = AskYesNo( COMPTOTSIZEMSG1 + COMPTOTSIZEMSG2, YES );
endwhile;

ListDestroy( listCompList );

endprogram

#include "sddialog.rul"

// Source file: Is5fn021.rul
```

{button ,PI("",`comment')} [Feedback](#)



ComponentMoveData

Syntax

ComponentMoveData (szMedia, nvDisk, nReserved);

Description

The ComponentMoveData function transfers/decompresses files associated with selected components in the [file media library](#) referenced by szMedia. You can call ComponentMoveData more than once on the same media, but you must reset internal structures before the second and subsequent calls by calling ComponentMoveData with a null string ("") in the first parameter position. (InstallShield automatically initializes the default media and internal structures before your first call to ComponentMoveData.) This function will automatically prompt the user for the next disk when it is needed.

Parameters

szMedia

Specifies the media name of the file media library whose files are to be transferred.

nvDisk

Returns the number of the last disk accessed. You do not need to initialize nvDisk before calling ComponentMoveData.

nReserved

Pass zero in this parameter. No other value is allowed.

Return values

0

ComponentMoveData was successful.

< 0

ComponentMoveData failed. Call [ComponentError](#) for additional information.

{button ,JI('LANGREF.HLP>Examples',`ComponentMoveData_example`)} [Example](#)

{button ,AL(`Script created component set vs. file media library;Specifying components in function calls;Componentfunctions;The minimum function calls necessary to install component selections',0,`,`)} [See also](#)

{button ,PI(``,`comment`)} [Feedback](#)



ComponentMoveData example

```
/*-----*\
*
* This example demonstrates SdSetupTypeEx, SdComponentDialog,
* ComponentIsItemSelected, ComponentGetData, ComponentValidate,
* ComponentMoveData, ComponentError, and PlaceWindow.
*
* Comments: To run this example script, create a project with the
*           following components (c), subcomponents (sc), and file
*           groups (fg):
*
*           (c) Program Files
*               (fg) Program DLLS
*               (fg) Program EXEs
*           (c) Example Files
*               (sc) Small Documents
*                   (fg) Small Document Examples
*               (sc) Books
*                   (fg) Book Examples
*               (sc) Graphics
*                   (fg) Graphic Examples
*           (c) Help Files
*               (fg) Help Files
*           (c) Utilities
*               (sc) Grammar Checker
*                   (fg) Grammar Checker
*               (sc) Art Studio
*                   (fg) Art Studio
*           (c) Evaluation Copy
*               (fg) Evaluation Copy
*               (fg) Help Files
*
* Insert "dummy" files into the file groups. Make sure you define the
* correct filename for the main EXE (MAIN_EXE, below) that you insert into
* the Program EXEs file group. Run the setup with and without a password
* assigned to the Program Files component (remember to rebuild your media
* each time).
*
* You can also create billboards (name them Bbrd1.bmp, Bbrd2.bmp, and
* Bbrd3.bmp) and add them to the project in the Setup Files pane under
* the Language Independent folder.
*
* This example script installs files, adds an icon to the Start Programs
* menu, and provides uninstallation functionality.
*
\*-----*/

#include "sddialog.h"

// Define strings. In a real setup you would define these in your string tables
// and precede each constant with @ to use them in your script.
#define COMP_SELECT_TITLE           "Select Components"
#define COMP_SELECT_MSG             "Select components and subcomponents to
install."
#define COMP_PROGRAMFILES_DISPLAYTEXT "Program Files"
#define PASSWORD_PROMPT             "Please enter the password."
#define PASSWORD_ERRMSG             "Password incorrect. Please enter again."
#define TITLE_MAIN                  "Word Processor"
#define TITLE_CAPTIONBAR            "Word Processor Setup"
#define APPBASE_PATH                "Your Company\\Word Processor"
```

```

#define COMPANY_NAME           "Your Company"
#define PRODUCT_NAME          "Word Processor"
#define PRODUCT_VERSION       "1.0"
#define PRODUCT_KEY           "Word Processor"
#define DEINSTALL_KEY        "Word Processor"
#define UNINSTALL_NAME       "Word Processor"
#define ADDINGICON            "Adding program icon to the Start Programs
menu..."
#define PROGRAMDIR            "Program"
#define DEFAULT_FOLDER_NAME   ""
#define APP_NAME              "Word Processor"
#define COMPLETE_MSG         "Setup is complete. You can run Word
Processor from the Start Programs menu."
#define MAIN_EXE              "WRITE.EXE"
#define SETUPTYPE_TITLE      "Setup Type Selection"
#define SETUPTYPE_MSG        "Please select a setup type."
#define SETUPTYPE_CUSTOM     "Custom"

```

```

// Global variable declarations.
STRING svData, svLogFile, szProgram, szComponent, svResult, svSetupType, svDir;
BOOL   bInitStepsDone, bPwdValid;
NUMBER nvData, nvDisk, nResult;

```

```

// Function declarations.
prototype SetUpFileTransfer();
prototype HandleComponentError(NUMBER);
prototype FinishSetup();

```

```

program
    SetUpFileTransfer();

    // Get the setup type.
    Disable( BACKBUTTON );
    svDir = TARGETDIR;
    SdSetupTypeEx( SETUPTYPE_TITLE, SETUPTYPE_MSG, "", svSetupType, 0 );
    // If user selected Custom setup type, display component selection dialog.
    if ( svSetupType = SETUPTYPE_CUSTOM ) then
        SdComponentDialog( COMP_SELECT_TITLE, COMP_SELECT_MSG, svDir, "" );
    endif;
    Enable( BACKBUTTON );

    // If the Program Files component is selected and there is a password
    // associated with it, the user must input the password and
    // it must be properly validated.
    nResult = FALSE;
    nvData = FALSE;
    nResult = ComponentIsItemSelected( MEDIA, COMP_PROGRAMFILES_DISPLAYTEXT );
    ComponentGetData( MEDIA, COMP_PROGRAMFILES_DISPLAYTEXT,
        COMPONENT_FIELD_PASSWORD, nvData, svData);
    if ( nResult && nvData ) then
        bPwdValid = FALSE;
        Disable( BACKBUTTON ); // Back button not needed or supported here.
        while ( !bPwdValid )
            AskText( PASSWORD_PROMPT, "", svResult );
            nResult = ComponentValidate( MEDIA, COMP_PROGRAMFILES_DISPLAYTEXT,
                svResult );

            if ( nResult = 0 ) then
                bPwdValid = TRUE;
            else
                MessageBox( PASSWORD_ERRMSG, SEVERE );
            endif;
        endwhile;
    endif;
endprogram

```

```

    Enable( BACKBUTTON ); // Restore Back button's default status.
endif;

// Set up the progress indicator, including locations for
// progress indicator, information gauges, and billboards.
PlaceWindow( FEEDBACK, LOWER_LEFT, LOWER_LEFT, LOWER_LEFT );
PlaceWindow( STATUSDLG, CENTERED, LOWER_RIGHT, LOWER_RIGHT );
PlaceWindow( BILLBOARD, CENTERED, CENTERED, CENTERED );
Enable( STATUSDLG );
Enable( INDVFILESTATUS );

// Indicate the final percentage the progress bar is to show when the
// following file transfer operation is complete.
StatusUpdate( ON, 95 );

// Transfer files to the target system. ComponentMoveData will prompt
// for next disk in a floppy disk installation.
nResult=ComponentMoveData( MEDIA, nvDisk, 0);

// To see the ComponentError function in action in the following call
// to the HandleComponentError user-defined function, remark out the
// while...endwhile loop above in which ComponentValidate is called,
// make sure a password is associated with the Program Files component
// in the IDE, rebuild the media, and run the setup.
HandleComponentError( nResult );

FinishSetup();
endprogram

/*-----*\
*
* Function: SetupFileTransfer()
*
* Purpose: This function sets up file transfer. The main reason for
*          abstracting this process into this function is to make it
*          easy to see the function calls this sample script demonstrates.
*
*\-----*/
function SetUpFileTransfer()
begin
    // Set up the installation screen.
    Enable( FULLWINDOWMODE );
    SetTitle( TITLE_MAIN, 24, WHITE );
    SetTitle( TITLE_CAPTIONBAR, 0, BACKGROUNDCAPTION );
    Enable( BACKGROUND );

    // Welcome the user, check that the system meets minimum requirements,
    // and verify the destination location.
    bInitStepsDone = FALSE;
    while (!bInitStepsDone)
        Disable( BACKBUTTON );
        Welcome( "", 0 );
        Enable( BACKBUTTON );

        TARGETDIR = PROGRAMFILES ^ APPBASE_PATH;
        if (AskDestPath( "", "", TARGETDIR, 0 ) != BACK) then
            bInitStepsDone = TRUE;
        endif;
    endwhile;

    // Set installation information required for registry entries and for
    // the following call to DeinstallStart.

```

```

InstallationInfo( COMPANY_NAME, PRODUCT_NAME,
                 PRODUCT_VERSION, PRODUCT_KEY );

// Initialize the uninstallation log file, including registry entry.
svLogFile = "Uninst.isu";
DeinstallStart( TARGETDIR, svLogFile, DEINSTALL_KEY, 0 );
RegDBSetItem( REGDB_UNINSTALL_NAME, UNINSTALL_NAME );
end;

/*-----*\
*
* Function:  HandleComponentError
*
* Purpose:  This function evaluates the value returned by a Component...
*           function and if the value is less than zero, displays the error
*           number and aborts the setup.
*
*-----*/
function HandleComponentError( nResult )

    NUMBER  nvError;
    STRING  svMedia, svComponent, svFileGroup, svFile;

begin
    if(nResult < 0) then
        ComponentError( svMedia, svComponent, svFileGroup, svFile, nvError );
        sprintfBox( INFORMATION, "Data Transfer Error Information",
                  "ComponentError returned the following data transfer error.\n" +
                  "Setup will now abort.\n\n" +
                  "Media Name: %s\nComponent: %s\nFile Group: %s\n" +
                  "File: %s\nError Number: %ld",
                  svMedia, svComponent, svFileGroup, svFile, nvError );
        abort;
    endif;
end;

/*-----*\
*
* Function:  FinishSetup()
*
* Purpose:  This function finishes the setup. The main reason for
*           abstracting this process into this function is to make it
*           easy to see the function calls this sample script demonstrates.
*
*-----*/
function FinishSetup()

begin
    // Indicate the final percentage the progress bar is to show when the
    // following file transfer operation is complete.
    StatusUpdate(ON, 99);

    // Increment progress bar to 99% for creation of Start Programs menu icon.
    SetStatusWindow (96 , ADDINGICON );

    // Add the APP_NAME icon to the DEFAULT_FOLDER_NAME folder.
    szProgram = TARGETDIR ^ PROGRAMDIR ^ MAIN_EXE;
    LongPathToQuote( szProgram, TRUE );
    AddFolderIcon( DEFAULT_FOLDER_NAME, APP_NAME, szProgram,
                  TARGETDIR ^ PROGRAMDIR,
                  "", 0, "", REPLACE );

    Delay( 1 );

```

```
// Disable the progress indicator and its settings.
Disable ( INDVFILESTATUS );
Disable( STATUSDLG );

// Announce setup complete and offer to view Readme file.
MessageBox( COMPLETE_MSG, INFORMATION );
end;

#define SD_SINGLE_DIALOGS 1
#define SD_SETUPTYPEEX 1
#define SD_COMPONENTDIALOG 1

#include "sddialog.rul"

// Source file: Is5fn603.rul
```

{button ,PI("",`comment')} [Feedback](#)



ComponentSelectItem

Syntax

ComponentSelectItem (szMedia, szComponent, bSelect);

Description

The ComponentSelectItem function sets a component's selection status to either selected or unselected. You can use ComponentSelectItem to change selection status before displaying components in component dialogs, and you can use it to change or override selections afterwards, depending on your needs.

Parameters

szMedia

Specifies the media name of the [file media library](#) or script-created component set containing the component whose selection status is to be set.

szComponent

Specifies the component whose selection status is to be set. For more information about specifying components and subcomponents in function calls, click [here](#).

bSelected

Specifies whether the component is selected or deselected. Pass one of the following predefined constants in this parameter:

TRUE

Selects the specified component.

FALSE

Deselects the specified component.

Return values

0

ComponentSelectItem successfully set the item's selection status.

< 0

ComponentSelectItem was unable to set the item's selection status. Call [ComponentError](#) for additional information.

{button ,JI('LANGREF.HLP>Examples','ComponentSelectItem_example')} [Example](#)

{button ,AL('Script created component set vs. file media library;Specifying components in function calls;ComponentAddItem;ComponentDialog;ComponentGetItemSize;ComponentIsItemSelected;ComponentListItems;SdComponentDialog;SdComponentDialogAdv',0,','')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



ComponentSelectItem example

```
/*-----*\
*
* This example demonstrates ComponentSetupTypeGetData, ComponentGetData,
* ComponentSetData, SdComponentDialog2, and ComponentSelectItem.
*
* Comments: To run this example script, create a project with the
*           following components (c), subcomponents (sc), and file
*           groups (fg):
*
*           (c) Program Files
*               (fg) Program DLLS
*               (fg) Program EXEs
*           (c) Example Files
*               (sc) Small Documents
*                   (fg) Small Document Examples
*               (sc) Books
*                   (fg) Book Examples
*               (sc) Graphics
*                   (fg) Graphic Examples
*           (c) Help Files
*               (fg) Help Files
*           (c) Utilities
*               (sc) Grammar Checker
*                   (fg) Grammar Checker
*               (sc) Art Studio
*                   (fg) Art Studio
*
* Be sure to enter descriptions into the Description fields of the component
* properties sheets for the Program Files and Example Files components and
* their subcomponents.
*
\*-----*/

#include "sddialog.h"

// Define strings. In a real setup you would define these in your string tables
// and precede each constant with @ to use them in your script.
#define COMP_SELECT_TITLE           "Select Components"
#define COMP_SELECT_MSG1           "IMPORTANT! Note the various component "
#define COMP_SELECT_MSG2           "and subcomponent names, descriptions, "
#define COMP_SELECT_MSG3           "and selection settings."
#define COMP_SELECT_MSG4           "IMPORTANT! Note the CHANGED component "
#define COMP_SELECT_MSG5           "and subcomponent name, description, "
#define COMP_SELECT_MSG6           "and selection settings."
#define COMP_PROGRAMFILES_DISPLAYNAME "Program Files"
#define COMP_EXAMPLEFILES_DISPLAYNAME "Example Files"
#define COMP_SMALLDOCUMENTS_DISPLAYNAME "Small Documents"
#define COMP_BOOKS_DISPLAYNAME      "Books"
#define COMP_GRAPHICS_DISPLAYNAME   "Graphics"
#define SETUP_TYPE                  "Typical"

// Global variable declarations.
STRING  svInfo, szInfo, szComponent;
NUMBER  nvInfo, nInfo, nResult;

program

    // Get the description field data for the SETUP_TYPE setup type.
```

```

ComponentSetupTypeGetData( MEDIA, SETUP_TYPE, SETUPTYPE_INFO_DESCRIPTION,
                           nvInfo, svInfo );
sprintfBox( INFORMATION, "ComponentSetupTypeGetData demo",
            "ComponentSetupTypeGetData got the following " +
            "value from the " + SETUP_TYPE + " description field:\n\n%s",
            svInfo );

// Get the description field data for the COMP_PROGRAMFILES_DISPLAYNAME
// component using ComponentGetData.
szComponent = COMP_PROGRAMFILES_DISPLAYNAME;
nResult = ComponentGetData( MEDIA, szComponent, COMPONENT_FIELD_DESCRIPTION,
                           nvInfo, svInfo );
sprintfBox( INFORMATION, "ComponentGetData demo",
            "ComponentGetData got the following value " +
            "from the " + COMP_PROGRAMFILES_DISPLAYNAME +
            " description field:\n\n%s", svInfo );

// Show the original description field values in the component selection dialog.
Disable( BACKBUTTON); // Back button not needed or handled here.
SdComponentDialog2( COMP_SELECT_TITLE, COMP_SELECT_MSG1 + COMP_SELECT_MSG2 +
                   COMP_SELECT_MSG3, TARGETDIR, "" );

// Change the displayed names for the Program Files component and the
// Example Files component and subcomponents.
szInfo = "CHANGED Component Name!";
nResult = ComponentSetData( MEDIA, szComponent, COMPONENT_FIELD_DISPLAYNAME,
                           nInfo, szInfo );
szComponent = COMP_EXAMPLEFILES_DISPLAYNAME;
nResult = ComponentSetData( MEDIA, szComponent, COMPONENT_FIELD_DISPLAYNAME,
                           nInfo, szInfo );
szComponent = COMP_EXAMPLEFILES_DISPLAYNAME + "\\\" +
              COMP_SMALLDOCUMENTS_DISPLAYNAME;
nResult = ComponentSetData( MEDIA, szComponent, COMPONENT_FIELD_DISPLAYNAME,
                           nInfo, szInfo );
szComponent = COMP_EXAMPLEFILES_DISPLAYNAME + "\\\" +
              COMP_BOOKS_DISPLAYNAME;
nResult = ComponentSetData( MEDIA, szComponent, COMPONENT_FIELD_DISPLAYNAME,
                           nInfo, szInfo );
szComponent = COMP_EXAMPLEFILES_DISPLAYNAME + "\\\" +
              COMP_GRAPHICS_DISPLAYNAME;
nResult = ComponentSetData( MEDIA, szComponent, COMPONENT_FIELD_DISPLAYNAME,
                           nInfo, szInfo );

// Change the descriptions displayed for the Program Files component
// and the Example Files component and subcomponents.
szComponent = COMP_PROGRAMFILES_DISPLAYNAME;
szInfo = "CHANGED description field value!";
nResult = ComponentSetData( MEDIA, szComponent, COMPONENT_FIELD_DESCRIPTION,
                           nInfo, szInfo );
szComponent = COMP_EXAMPLEFILES_DISPLAYNAME;
nResult = ComponentSetData( MEDIA, szComponent, COMPONENT_FIELD_DESCRIPTION,
                           nInfo, szInfo );
szComponent = COMP_EXAMPLEFILES_DISPLAYNAME + "\\\" +
              COMP_SMALLDOCUMENTS_DISPLAYNAME;
nResult = ComponentSetData( MEDIA, szComponent, COMPONENT_FIELD_DESCRIPTION,
                           nInfo, szInfo );
szComponent = COMP_EXAMPLEFILES_DISPLAYNAME + "\\\" +
              COMP_BOOKS_DISPLAYNAME;
nResult = ComponentSetData( MEDIA, szComponent, COMPONENT_FIELD_DESCRIPTION,
                           nInfo, szInfo );
szComponent = COMP_EXAMPLEFILES_DISPLAYNAME + "\\\" +
              COMP_GRAPHICS_DISPLAYNAME;
nResult = ComponentSetData( MEDIA, szComponent, COMPONENT_FIELD_DESCRIPTION,

```

```
        nInfo, szInfo );

// Deselect the Program Files and Example Files components (and all their
// subcomponents, by extension).
ComponentSelectItem( MEDIA, COMP_PROGRAMFILES_DISPLAYNAME, FALSE );
ComponentSelectItem( MEDIA, COMP_EXAMPLEFILES_DISPLAYNAME, FALSE );

// Display the components again, noting the changed names, descriptions,
// and selection settings.
SdComponentDialog2( COMP_SELECT_TITLE, COMP_SELECT_MSG4 + COMP_SELECT_MSG5 +
                    COMP_SELECT_MSG6, TARGETDIR, "" );
endprogram

#include "sddialog.rul"

// Source file: IS5FN623.rul
```

{button ,PI("",`comment')} [Feedback](#)



ComponentSetData

Syntax

ComponentSetData (szMedia, szComponent, nInfo, nData, szData);

Description

The ComponentSetData function sets properties and data for the specified component. For the most part, the settings correspond to the field settings in the Component Properties window, accessible from the Components pane in the InstallShield IDE. If the media is for script-created components, some fields cannot be set.

Parameters

szMedia

Specifies the name of the media.

szComponent

Specifies the name of the component. For more information about specifying components and subcomponents in function calls, click [here](#).

nInfo

Specifies the type of information to be set. Pass one of the following predefined constants in this parameter:

COMPONENT_FIELD_DESCRIPTION

This text is displayed in the Description field of component selection dialogs.

COMPONENT_FIELD_FTPLOCATION

An FTP location.

COMPONENT_FIELD_HTTPLOCATION

An HTTP location.

COMPONENT_FIELD_STATUS (not for script media)

This text is displayed in the progress indicator during file transfer.

COMPONENT_FIELD_VISIBLE

Indicates whether the component is visible or not. The parameter nData can be one of the following:

TRUE

This component is visible.

FALSE

This component is not visible.

COMPONENT_FIELD_OVERWRITE (not for script media)

Determines how to overwrite files that exist on the target system. The parameter nData can be one or more of the following:

COMPONENT_VALUE_ALWAYSoverwrite

COMPONENT_VALUE_SAMEORNEWDATE

COMPONENT_VALUE_NEWERDATE

COMPONENT_VALUE_OLDERDATE

COMPONENT_VALUE_SAMEORNEWVERSION

COMPONENT_VALUE_NEWERVERSION

COMPONENT_VALUE_OLDERVERSION

COMPONENT_VALUE_NEVEROVERWRITE

COMPONENT_FIELD_DESTINATION (not for script media)

Determines the destination (target) location for the files associated with specified component.

COMPONENT_FIELD_SELECTED

Sets the selection status of the component. This setting has the same effect as ComponentSelectItem. The parameter nData can be one of the following:

TRUE

Select this component item.

FALSE

Deselect this component item.

COMPONENT_FIELD_SIZE (not for file media)

The total original file size for the component.

COMPONENT_FIELD_MISC

Miscellaneous text.

COMPONENT_FIELD_DISPLAYNAME

Determines the name displayed for the component in component selection dialogs.

COMPONENT_FIELD_CDROM_FOLDER (not for script media)

For CD-ROM distribution media builds only. The location of the component's data on the CD.

nData

Specifies a numeric value when the information indicated by nInfo is numeric.

szData

Specifies a string value when the information indicated by nInfo is a string.

Return values

0

ComponentSetData was successful.

< 0

ComponentSetData failed. Call [ComponentError](#) for additional information.

{button ,JI('LANGREF.HLP>Examples','ComponentSetData_example')} [Example](#)

{button ,AL('Script created component set vs. file media library;Specifying components in function calls;Componentfunctions',0,','')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



ComponentSetData example

```
/*-----*\
 *
 * This example demonstrates ComponentAddItem, ComponentSetData,
 * ComponentCompareSizeRequired, and SdComponentMult.
 *
 * Comments: Make sure you define DESTDIR as a drive with little space
 *           available, such as a floppy drive. Remember to put a
 *           diskette into the floppy drive before running the script.
 *
 *-----*/

#include "sddialog.h"

#define COMP1          "CAD Prog. Files"
#define COMP1SIZE      25000000
#define COMP1DESC      "CAD program EXEs and DLLs."
#define COMP2          "CAD Templates"
#define COMP2SIZE      18000000
#define COMP2DESC      "CAD template files."
#define SUBCOMP1       "Industrial"
#define SUBCOMP1SIZE   7000000
#define SUBCOMP1DESC   "CAD industrial engineering template files."
#define SUBCOMP2       "Civil"
#define SUBCOMP2SIZE   5000000
#define SUBCOMP2DESC   "CAD civil engineering template files."
#define SUBCOMP3       "Mechanical"
#define SUBCOMP3SIZE   6000000
#define SUBCOMP3DESC   "CAD mechanical engineering template files."
#define DESTDIR        "a:\\"
#define SDCOMPTITLE    "Displaying Script-created Components"
#define SDCOMPMSG      "Make sure all components are selected."
#define COMPSIZEERRMSG "Make sure target drive is accessible."

STRING svDir, svTarget;
NUMBER nvSize, nData;

program
    // Disable the Back button, which is not needed.
    Disable( BACKBUTTON );

    // Make a script-created component set including subcomponents
    // and give it a media name of "Run-time CAD".
    MEDIA = "Run-time CAD";
    ComponentAddItem( MEDIA, COMP1, COMP1SIZE, TRUE );
        ComponentSetData ( MEDIA, COMP1, COMPONENT_FIELD_DESCRIPTION,
            nData, COMP1DESC );
    ComponentAddItem( MEDIA, COMP2, COMP1SIZE, TRUE );
        ComponentSetData ( MEDIA, COMP2, COMPONENT_FIELD_DESCRIPTION,
            nData, COMP2DESC );
    ComponentAddItem( MEDIA, COMP2 + "\\\" + SUBCOMP1, SUBCOMP1SIZE, TRUE );
        ComponentSetData ( MEDIA, COMP2 + "\\\" + SUBCOMP1,
            COMPONENT_FIELD_DESCRIPTION, nData, SUBCOMP1DESC );
    ComponentAddItem( MEDIA, COMP2 + "\\\" + SUBCOMP2, SUBCOMP2SIZE, TRUE );
        ComponentSetData ( MEDIA, COMP2 + "\\\" + SUBCOMP2,
            COMPONENT_FIELD_DESCRIPTION, nData, SUBCOMP2DESC );
    ComponentAddItem( MEDIA, COMP2 + "\\\" + SUBCOMP3, SUBCOMP3SIZE, TRUE );
        ComponentSetData ( MEDIA, COMP2 + "\\\" + SUBCOMP3,
            COMPONENT_FIELD_DESCRIPTION, nData, SUBCOMP3DESC );
```

```
// Display the script-created components and subcomponents.
SdComponentMult( SDCOMPTITLE, SDCOMPMSG, svDir, "" );

// Set TARGETDIR to DESTDIR, which, if you define as a floppy drive
// (with a diskette in it), will cause ComponentCompareSizeRequired
// to display its 'not enough space' message.
nvSize = 0;
TARGETDIR = DESTDIR;
if ( ComponentCompareSizeRequired(MEDIA, svTarget, nvSize) < 0 ) then
    MessageBox( COMPSIZEERRMSG, SEVERE );
endif;

endprogram

#include "sddialog.rul"

// Source file: Is5fn598.rul
```

{button ,PI("",`comment')} Feedback



ComponentSetTarget

Syntax

ComponentSetTarget (szMedia, szUserVar, szLocation);

Description

The ComponentSetTarget function substitutes a script variable's value (szLocation) for a user-defined variable (szUserVar). User-defined variables are used in the Destination field of the Component Properties window. szLocation may be a complete path (including drive letter and colon) or a partial path, depending on how szUserVar is used. You are responsible for ensuring that szLocation is the correct form of path expression. Call ComponentSetTarget before calling [ComponentMoveData](#).



This function cannot be used with script-created components.

Parameters

szMedia

Specifies the name of the media.

szUserVar

Specifies the user-defined variable. In the InstallShield IDE, user-defined variables take the form <variable name>. Express the user-defined variable as a string, including the brackets. For example:

```
szUserVar = "<szVar>";
```

szLocation

Specifies the path expression to substitute for the user-defined variable.

Return values

0

ComponentSetTarget was successful.

< 0

ComponentSetTarget failed. Call [ComponentError](#) for additional information.

{button ,JI('LANGREF.HLP>Examples',`ComponentSetTarget_example')}} [Example](#)

{button ,AL(`Script created component set vs. file media library;Specifying components in function calls;Select the destination folder for a component;Componentfunctions',0,`,`')}} [See also](#)

{button ,PI("",`comment')}} [Feedback](#)



ComponentSetTarget example

```
/*-----*\
*
* This example demonstrates ComponentSetTarget. Includes calls to
* SdComponentMult, ComponentIsItemSelected, and ComponentMoveData.
*
* Comments: This example uses a script-defined variable to allow
* you to specify a variable component target location
* at run time.
*
* To run this example, create a project with at least two
* components: Program Files and Example Files. Create two
* file groups, add files to each, and assign one file group
* to each component. Open the Destination field properties
* sheet of the Program Files component. Create a folder named
* Program under the General Application Destination folder.
* With the new Program folder selected, click OK. The Program
* Files component's Destination field should contain
* <TARGETDIR>\Program.
*
* Open the Destination field properties sheet of the Example
* Files component. Select Script Defined Folders, click
* New Folder, and enter <szVarDir>. With <szVarDir> selected,
* click New Folder again and create a folder named Examples.
* With the Examples folder selected, click OK. The Example
* Files component's Destination field should contain
* <szVarDir>\Examples. You should have, at a minimum, the
* following folders in the Destination field properties tab:
*
* General Application Destination
*     Program
*
* Script Defined Folders
*     <szVarDir>
*     Examples
*
*-----*/

#include "sddialog.h"

// Define strings. In a real setup you would define these in your string tables.
#define ASKTARGET1 "Select General Target Location"
#define ASKTARGET2 "Select Example Files Target Location"
#define TARGETMSG1 "Select a general target location for the application."
#define TARGETMSG2 "Select a special target location for example files."
#define TARGETMSG3 "You can accept the general target location."
#define COMP_SELECT_TITLE "Select Components"
#define COMP_SELECT_MSG1 "Select components and subcomponents to install.\n"
#define COMP_SELECT_MSG2 "MAKE SURE YOU SELECT EXAMPLE FILES."
#define COMP_EXAMPLE_FILES "Example Files"
#define APPBASE_PATH "Your Company\\Word Processor"
#define COMPANY_NAME "Your Company"
#define PRODUCT_NAME "Word Processor"
#define PRODUCT_VERSION "1.0"
#define PRODUCT_KEY "Word Processor"
#define DEINSTALL_KEY "Word Processor"
#define UNINSTALL_NAME "Word Processor"

// Global variable declarations.
```

```

STRING  svLogFile, svDir, szData;
NUMBER  nvDisk, nResult;

// Function declarations.
prototype HandleComponentError(NUMBER);

program

    // Get a TARGETDIR target location.
    TARGETDIR = PROGRAMFILES ^ APPBASE_PATH;
    Disable( BACKBUTTON );
    AskDestPath( ASKTARGET1, TARGETMSG1, TARGETDIR, 0 );
    Enable( BACKBUTTON );

    // Set up uninstallation.
    InstallationInfo( COMPANY_NAME, PRODUCT_NAME,
                     PRODUCT_VERSION, PRODUCT_KEY );
    svLogFile = "Uninst.isu";
    DeinstallStart( TARGETDIR, svLogFile, DEINSTALL_KEY, 0 );
    RegDBSetItem( REGDB_UNINSTALL_NAME, UNINSTALL_NAME );

    // Let user select components.
    Disable( BACKBUTTON );
    svDir = TARGETDIR;
    SdComponentMult( COMP_SELECT_TITLE, COMP_SELECT_MSG1 + COMP_SELECT_MSG2,
                    svDir, "" );
    // For the sake of this example, ensure COMP_EXAMPLE_FILES is selected.
    if ( ComponentIsItemSelected( MEDIA, COMP_EXAMPLE_FILES ) = FALSE ) then
        ComponentSetData ( MEDIA , COMP_EXAMPLE_FILES ,
                          COMPONENT_FIELD_SELECTED , TRUE , szData );
    endif;

    // Get the special path that you will associate with <szVarDir>.
    AskDestPath( ASKTARGET2, TARGETMSG2 + "\n\n" + TARGETMSG3, svDir, 0 );

    // Associate svDir with <szVarDir>.
    ComponentSetTarget( MEDIA, "<szVarDir>", svDir );
    Enable( BACKBUTTON );

    // Set up the progress indicator.
    Enable( STATUSDLG );
    Enable( INDVFILESTATUS );
    StatusUpdate( ON, 99 );

    // Transfer files to the target system. Handle any errors.
    nResult=ComponentMoveData( MEDIA, nvDisk, 0);
    HandleComponentError( nResult );

    // Disable the progress indicator and its settings.
    Disable ( INDVFILESTATUS );
    Disable( STATUSDLG );

endprogram

/*-----*\
*
* Function:  HandleComponentError
*
* Purpose:  This function evaluates the value returned by a Component...
*           function and if the value is less than zero, displays the error
*           number and aborts the setup.
*
*-----*/

```

```
function HandleComponentError( nResult )

    NUMBER  nvError;
    STRING  svMedia, svComponent, svFileGroup, svFile;

begin
    if(nResult < 0) then
        ComponentError( svMedia, svComponent, svFileGroup, svFile, nvError );
        sprintfBox( INFORMATION, "Data Transfer Error Information",
            "ComponentError returned the following data transfer error.\n" +
            "Setup will now abort.\n\n" +
            "Media Name: %s\nComponent: %s\nFile Group: %s\n" +
            "File: %s\nError Number: %ld",
            svMedia, svComponent, svFileGroup, svFile, nvError );
        abort;
    endif;
end;

#include "sddialog.rul"

// Source file: Isfn615.rul
```

{button ,PI("",`comment')} [Feedback](#)



ComponentSetupTypeEnum

Syntax

```
ComponentSetupTypeEnum (szMedia, listSetupTypes);
```

Description

The ComponentSetupTypeEnum function enumerates all setup types associated with the specified media. These setup types are defined by you in the IDE and stored in the [file media library](#). ComponentSetupTypeEnum does not work with script-created components. You must create the listSetupTypes string list using the [ListCreate](#) function.

Parameters

szMedia

Specifies the name of the media.

listSetupTypes

Returns a list of all setup types in the specified media.

Return values

0

ComponentSetupTypeEnum was successful.

< 0

ComponentSetupTypeEnum failed.

{button ,JI('LANGREF.HLP>Examples','ComponentSetupTypeEnum_example')} [Example](#)

{button ,AL('Script created component set vs. file media library;Specifying components in function calls;Componentfunctions',0,'')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



ComponentSetupTypeEnum example

```
/*-----*\
 *
 * This example illustrates the use of ComponentSetupTypeEnum, which creates
 * enumerates the setup types in a media.
 *
 * Comments: To run this example script, create a project (or insert into
 *           a project) with several setup types defined. The value of
 *           DEFTYPE must be a valid name of one of your setup types.
 *
 *-----*/

#include "sddialog.h"

#define SDSHOWTITLE "Setup Type Enumeration"
#define SDSHOWMSG MEDIA + " media's enumerated setup types are:"
#define SETUPTITLE "Setup Type Selection"
#define SETUPMSG "Select a setup type."
#define DEFTYPE "Typical"

LIST listID;
STRING svSetupType;

program
    // Disable Back button, which is not needed.
    Disable( BACKBUTTON );

    listID = ListCreate( STRINGLIST );

    if ( ComponentSetupTypeEnum( MEDIA, listID ) < 0 ) then
        MessageBox( "ComponentSetupTypeEnum failed.", WARNING );
    endif;

    // Display the Enumerated Setup Types
    SdShowInfoList( SDSHOWTITLE, SDSHOWMSG, listID );

    // Now show setup types in selection dialog.
    svSetupType = DEFTYPE;
    SdSetupTypeEx( SETUPTITLE, SETUPMSG, "", svSetupType, 0 );

    ListDestroy( listID );

endprogram

#include "sddialog.rul"

//Source file: Is5fn611.rul
```

{button ,PI("",`comment')} Feedback



ComponentSetupTypeGetData

Syntax

ComponentSetupTypeGetData (szMedia, szSetupType, nInfo, nvResult, svResult);

Description

The ComponentSetupTypeGetData function retrieves data associated with a specified setup type. You can then use this data for any purpose.

Parameters

szMedia

Specifies the name of the media.

szSetupType

Specifies the setup type name. This name must be specified exactly as it appears in the InstallShield IDE, for example, "Typical".

nInfo

Specifies which information to retrieve. Pass one of the following predefined constants in this parameter:

SETUPTYPE_INFO_DESCRIPTION

Retrieves the description of the specified setup type. The description is returned in svResult.

SETUPTYPE_INFO_DISPLAYNAME

Retrieves the display name of the setup type. The name is returned in svResult.

nvResult

Returns a NUMBER or LONG value when nInfo specifies information of that type.

svResult

Returns a STRING value when nInfo specifies information of that type.

Return values

0

ComponentSetupTypeGetData was successful.

< 0

ComponentSetupTypeGetData failed. Call [ComponentError](#) for additional information.

Comments

A typical application of ComponentSetupTypeGetData might be to display setup type information in a custom setup type-related dialog box. You would call ComponentSetupTypeGetData inside the switch-case statement following the call to WaitOnDialog that displays the custom dialog box.

```
{button ,JI('LANGREF.HLP>Examples','ComponentSetupTypeGetData_example')} Example  
{button ,AL('Script created component set vs. file media library;Specifying components in function  
calls;Componentfunctions',0,','')} See also  
{button ,PI('','comment')} Feedback
```



ComponentSetupTypeGetData Example

```
/*-----*\
 *
 * This example demonstrates ComponentSetupTypeGetData, ComponentGetData,
 * ComponentSetData, SdComponentDialog2, and ComponentSelectedItem.
 *
 * Comments: To run this example script, create a project with the
 *           following components (c), subcomponents (sc), and file
 *           groups (fg):
 *
 *           (c) Program Files
 *               (fg) Program DLLS
 *               (fg) Program EXEs
 *           (c) Example Files
 *               (sc) Small Documents
 *                   (fg) Small Document Examples
 *               (sc) Books
 *                   (fg) Book Examples
 *               (sc) Graphics
 *                   (fg) Graphic Examples
 *           (c) Help Files
 *               (fg) Help Files
 *           (c) Utilities
 *               (sc) Grammar Checker
 *                   (fg) Grammar Checker
 *               (sc) Art Studio
 *                   (fg) Art Studio
 *
 * Be sure to enter descriptions into the Description fields of the component
 * properties sheets for the Program Files and Example Files components and
 * their subcomponents.
 *
 *-----*/

#include "sddialog.h"

// Define strings. In a real setup you would define these in your string tables
// and precede each constant with @ to use them in your script.
#define COMP_SELECT_TITLE           "Select Components"
#define COMP_SELECT_MSG1           "IMPORTANT! Note the various component "
#define COMP_SELECT_MSG2           "and subcomponent names, descriptions, "
#define COMP_SELECT_MSG3           "and selection settings."
#define COMP_SELECT_MSG4           "IMPORTANT! Note the CHANGED component "
#define COMP_SELECT_MSG5           "and subcomponent name, description, "
#define COMP_SELECT_MSG6           "and selection settings."
#define COMP_PROGRAMFILES_DISPLAYNAME "Program Files"
#define COMP_EXAMPLEFILES_DISPLAYNAME "Example Files"
#define COMP_SMALLDOCUMENTS_DISPLAYNAME "Small Documents"
#define COMP_BOOKS_DISPLAYNAME      "Books"
#define COMP_GRAPHICS_DISPLAYNAME   "Graphics"
#define SETUP_TYPE                  "Typical"

// Global variable declarations.
STRING  svInfo, szInfo, szComponent;
NUMBER  nvInfo, nInfo, nResult;

program

    // Get the description field data for the SETUP_TYPE setup type.
```

```

ComponentSetupTypeGetData( MEDIA, SETUP_TYPE, SETUPTYPE_INFO_DESCRIPTION,
                           nvInfo, svInfo );
sprintfBox( INFORMATION, "ComponentSetupTypeGetData demo",
           "ComponentSetupTypeGetData got the following " +
           "value from the " + SETUP_TYPE + " description field:\n\n%s",
           svInfo );

// Get the description field data for the COMP_PROGRAMFILES_DISPLAYNAME
// component using ComponentGetData.
szComponent = COMP_PROGRAMFILES_DISPLAYNAME;
nResult = ComponentGetData( MEDIA, szComponent, COMPONENT_FIELD_DESCRIPTION,
                           nvInfo, svInfo );
sprintfBox( INFORMATION, "ComponentGetData demo",
           "ComponentGetData got the following value " +
           "from the " + COMP_PROGRAMFILES_DISPLAYNAME +
           " description field:\n\n%s", svInfo );

// Show the original description field values in the component selection dialog.
Disable( BACKBUTTON); // Back button not needed or handled here.
SdComponentDialog2( COMP_SELECT_TITLE, COMP_SELECT_MSG1 + COMP_SELECT_MSG2 +
                   COMP_SELECT_MSG3, TARGETDIR, "" );

// Change the displayed names for the Program Files component and the
// Example Files component and subcomponents.
szInfo = "CHANGED Component Name!";
nResult = ComponentSetData( MEDIA, szComponent, COMPONENT_FIELD_DISPLAYNAME,
                           nInfo, szInfo );
szComponent = COMP_EXAMPLEFILES_DISPLAYNAME;
nResult = ComponentSetData( MEDIA, szComponent, COMPONENT_FIELD_DISPLAYNAME,
                           nInfo, szInfo );
szComponent = COMP_EXAMPLEFILES_DISPLAYNAME + "\\\" +
              COMP_SMALLDOCUMENTS_DISPLAYNAME;
nResult = ComponentSetData( MEDIA, szComponent, COMPONENT_FIELD_DISPLAYNAME,
                           nInfo, szInfo );
szComponent = COMP_EXAMPLEFILES_DISPLAYNAME + "\\\" +
              COMP_BOOKS_DISPLAYNAME;
nResult = ComponentSetData( MEDIA, szComponent, COMPONENT_FIELD_DISPLAYNAME,
                           nInfo, szInfo );
szComponent = COMP_EXAMPLEFILES_DISPLAYNAME + "\\\" +
              COMP_GRAPHICS_DISPLAYNAME;
nResult = ComponentSetData( MEDIA, szComponent, COMPONENT_FIELD_DISPLAYNAME,
                           nInfo, szInfo );

// Change the descriptions displayed for the Program Files component
// and the Example Files component and subcomponents.
szComponent = COMP_PROGRAMFILES_DISPLAYNAME;
szInfo = "CHANGED description field value!";
nResult = ComponentSetData( MEDIA, szComponent, COMPONENT_FIELD_DESCRIPTION,
                           nInfo, szInfo );
szComponent = COMP_EXAMPLEFILES_DISPLAYNAME;
nResult = ComponentSetData( MEDIA, szComponent, COMPONENT_FIELD_DESCRIPTION,
                           nInfo, szInfo );
szComponent = COMP_EXAMPLEFILES_DISPLAYNAME + "\\\" +
              COMP_SMALLDOCUMENTS_DISPLAYNAME;
nResult = ComponentSetData( MEDIA, szComponent, COMPONENT_FIELD_DESCRIPTION,
                           nInfo, szInfo );
szComponent = COMP_EXAMPLEFILES_DISPLAYNAME + "\\\" +
              COMP_BOOKS_DISPLAYNAME;
nResult = ComponentSetData( MEDIA, szComponent, COMPONENT_FIELD_DESCRIPTION,
                           nInfo, szInfo );
szComponent = COMP_EXAMPLEFILES_DISPLAYNAME + "\\\" +
              COMP_GRAPHICS_DISPLAYNAME;
nResult = ComponentSetData( MEDIA, szComponent, COMPONENT_FIELD_DESCRIPTION,

```

```
        nInfo, szInfo );

// Deselect the Program Files and Example Files components (and all their
// subcomponents, by extension).
ComponentSelectItem( MEDIA, COMP_PROGRAMFILES_DISPLAYNAME, FALSE );
ComponentSelectItem( MEDIA, COMP_EXAMPLEFILES_DISPLAYNAME, FALSE );

// Display the components again, noting the changed names, descriptions,
// and selection settings.
SdComponentDialog2( COMP_SELECT_TITLE, COMP_SELECT_MSG4 + COMP_SELECT_MSG5 +
                    COMP_SELECT_MSG6, TARGETDIR, "" );
endprogram

#include "sddialog.rul"

// Source file: IS5FN623.rul
```

{button ,PI("",`comment')} Feedback



ComponentSetupTypeSet

Syntax

ComponentSetupTypeSet (szMedia, szSetupType);

Description

The ComponentSetupTypeSet function sets the specified setup type in the [file media library](#) referenced by szMedia. You can use ComponentSetupTypeSet to override the selection made in a setup type dialog, such as [SdSetupTypeEx](#).

Parameters

szMedia

Specifies the name of the media.

szSetupType

Specifies the setup type to be set.

Return values

0

ComponentSetupTypeSet was successful.

< 0

ComponentSetupTypeSet failed.

{button ,JI('LANGREF.HLP>Examples','ComponentSetupTypeSet_example')} [Example](#)

{button ,AL('Script created component set vs. file media library;Specifying components in function calls;Getting setup type selection from the end user;Componentfunctions',0,','')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



ComponentSetupTypeSet example

```
/*-----*\
*
* This example illustrates the use of ComponentSetupTypeSet. It also
* includes calls to SdSetupTypeEx and SdComponentMult.
*
* Comments: To run this example script, create a project (or insert into
* a project) with several setup types and components. Make sure
* you specify the setup types' default component selections.
* If you do not include a Compact setup type, then define one
* of your setup types as SETUP_TYPE in the #define SETUP_TYPE
* line below.
*
*-----*\

#include "sddialog.h"

#define SETUP_TYPE "Compact"
#define SDSETUPTITLE "Setup Type Selection"
#define SDSETUPMSG "Select a setup type other than " + SETUP_TYPE + "."
#define SDCOMPTITLE "Component Selection"
#define SDCOMPMSG1 "Component selection before ComponentSetupTypeSet."
#define SDCOMPMSG2 "Component selection after ComponentSetupTypeSet."
#define MSG1 "ComponentSetupTypeSet will now select all\n"
#define MSG2 "components in the " + SETUP_TYPE + " setup type."

STRING szSetupType, svSetup, svDir;

program
    // Disable Back button, which is not needed.
    Disable( BACKBUTTON );

    // Select a setup type other than SETUP_TYPE to show default
    // selection settings.
    SdSetupTypeEx( SDSETUPTITLE, SDSETUPMSG, "", svSetup, 0 );

    // Display the component selections for the selected setup type.
    svDir = TARGETDIR;
    SdComponentMult( SDCOMPTITLE, SDCOMPMSG1, svDir, "" );

    MessageBox( MSG1 + MSG2, INFORMATION );

    // Now change/override the previous setup type selection by
    // selecting SETUP_TYPE's components. All others are deselected.
    szSetupType = SETUP_TYPE;
    if ( ComponentSetupTypeSet( MEDIA, szSetupType ) < 0 ) then
        MessageBox( "ComponentSetupTypeSet failed.", SEVERE );
    endif;

    // Display the new component selections.
    SdComponentMult( SDCOMPTITLE, SDCOMPMSG2, svDir, "" );

endprogram

#include "sddialog.rul"

// Source file: Is5fn624.rul
```

{button ,PI("",`comment')} Feedback



ComponentTotalSize

Syntax

ComponentTotalSize (szMedia, szComponent, bIncludeSubcomp, bTargetSize);

Description

The ComponentTotalSize function returns the total size, in bytes, of the selected component(s) referenced by szComponent. To include subcomponents in the size calculation, set bIncludeSubcomp to TRUE. To get the total size of all the components in the specified media, set szComponent to a null string ("") and set bIncludeSubcomp to TRUE.

If you compare the value returned by ComponentTotalSize to the space required values displayed in the SdComponentDialog, SdComponentDialog2, SdComponentDialogAdv, and SdComponentMult dialog boxes, you may notice a small difference. This difference is due to rounding that takes place when those dialog boxes calculate their values.

Parameters

szMedia

Specifies the name of the media.

szComponent

Specifies the name of the component whose size is to be retrieved. To retrieve the size of the entire media, pass a null string ("") in this parameter. For more information about specifying components and subcomponents in function calls, click [here](#).

bIncludeSubcomp

Indicates whether or not to include selected subcomponents of szComponent. Pass one of the following predefined constants in this parameter:

TRUE

Include selected subcomponents in the size calculation.

FALSE

Do not include subcomponents in the size calculation.

bTargetSize

Indicates whether to retrieve the original/uncompressed size or the size in the media library. Pass one of the following predefined constants in this parameter:

TRUE

Retrieve the original/uncompressed size.

FALSE

Retrieve the size in the media library.

Return values

The total size, in bytes, of the selected component(s).

{button ,JI('LANGREF.HLP>Examples',`ComponentTotalSize_example')}

[Example](#)

{button ,AL(`Script created component set vs. file media library;Specifying components in function calls;SdComponentDialog;SdComponentDialog2;SdComponentDialogAdv;SdComponentMult'0,','`)}
[also](#)

[See](#)

{button ,PI('',`comment')} [Feedback](#)



ComponentTotalSize example

```
/*-----*\
*
* This example demonstrates ComponentListItems, SdComponentMult,
* and ComponentTotalSize.
*
* Comments: To run this example script, create a project with the
* following components (c), subcomponents (sc), and file
* groups (fg):
*
* (c) Program Files
* (fg) Program DLLS
* (fg) Program EXEs
* (c) Example Files
* (sc) Small Documents
* (fg) Small Document Examples
* (sc) Books
* (fg) Book Examples
* (sc) Graphics
* (fg) Graphic Examples
* (c) Help Files
* (fg) Help Files
* (c) Utilities
* (sc) Grammar Checker
* (fg) Grammar Checker
* (sc) Art Studio
* (fg) Art Studio
*
*-----*/

#include "sddialog.h"

#define COMP_SELECT_TITLE      "Select Components"
#define COMP_SELECT_MSG       "Select components and subcomponents to install."
#define COMPTOTSIZEMSG1       "Want to change component selections and see\n"
#define COMPTOTSIZEMSG2       "size change reflected in ComponentTotalSize call?"

// Global variable declarations.
STRING  szDir, svString;
NUMBER  nResult, nDone;
LIST    listCompList, listTemp;

program
    // Disable Back button, which is not needed.
    Disable( BACKBUTTON );

    // Create a string list of all top-level components.
    listCompList = ListCreate( STRINGLIST );
    ComponentListItems( MEDIA, "", listCompList );
    // Display the string list of top-level components.
    SdShowInfoList( "List MEDIA Components", "MEDIA contains " +
        "the following top-level components:", listCompList );

    // Get each top-level component in listCompList, in turn, and
    // list and display all its subcomponents, if any.
    nResult = ListGetFirstString( listCompList, svString );
    while ( nResult != END_OF_LIST )
        listTemp = ListCreate( STRINGLIST );
        ComponentListItems( MEDIA, svString, listTemp );
        SdShowInfoList( "Subcomponent Listing", svString + " contains " +
```

```
        "the following subcomponents:", listTemp );
    ListDestroy( listTemp );
    nResult = ListGetNextString( listCompList, svString );
endwhile;

// Show component selection dialog and total size of all selected
// components. Loop to change selections and see total size change
// reflected in the call to ComponentTotalSize.
nDone = YES;
while ( nDone = YES )
    szDir = TARGETDIR;
    SdComponentMult( COMP_SELECT_TITLE, COMP_SELECT_MSG, szDir, "" );

    nResult = ComponentTotalSize( MEDIA, "", TRUE, TRUE );
    sprintfBox( INFORMATION, "", "Total size of all files " +
        "in SELECTED components:\n\n%d", nResult );
    nDone = AskYesNo( COMPTOTSIZEMSG1 + COMPTOTSIZEMSG2, YES );
endwhile;

ListDestroy( listCompList );

endprogram

#include "sddialog.rul"

// Source file: Is5fn021.rul
```

{button ,PI("",`comment')} [Feedback](#)



ComponentValidate

Syntax

ComponentValidate (szMedia, szComponent, szPassword);

Description

The ComponentValidate function validates the password of the [file media library](#) or of a specified component.



This function cannot be used with script-created components.

Parameters

szMedia

Specifies the name of the media.

szComponent

Specifies the name of the component. If this parameter is a null string (""), the entire media library is assumed. For more information about specifying components and subcomponents in function calls, click [here](#).

szPassword

Specifies the password to be validated.

Return values

0

ComponentValidate was successful.

< 0

ComponentValidate failed. Call [ComponentError](#) for additional information.

{button ,JI('LANGREF.HLP>Examples','ComponentValidate_example')} [Example](#)

{button ,AL('ComponentError;ComponentMoveData;Script created component set vs. file media library;Specifying components in function calls;Password-protect a component;Get and validate a component or file media library password',0,','')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



ComponentValidate example

```
/*-----*\
*
* This example demonstrates SdSetupTypeEx, SdComponentDialog,
* ComponentIsItemSelected, ComponentGetData, ComponentValidate,
* ComponentMoveData, ComponentError, and PlaceWindow.
*
* Comments: To run this example script, create a project with the
*           following components (c), subcomponents (sc), and file
*           groups (fg):
*
*           (c) Program Files
*               (fg) Program DLLS
*               (fg) Program EXEs
*           (c) Example Files
*               (sc) Small Documents
*                   (fg) Small Document Examples
*               (sc) Books
*                   (fg) Book Examples
*               (sc) Graphics
*                   (fg) Graphic Examples
*           (c) Help Files
*               (fg) Help Files
*           (c) Utilities
*               (sc) Grammar Checker
*                   (fg) Grammar Checker
*               (sc) Art Studio
*                   (fg) Art Studio
*           (c) Evaluation Copy
*               (fg) Evaluation Copy
*               (fg) Help Files
*
* Insert "dummy" files into the file groups. Make sure you define the
* correct filename for the main EXE (MAIN_EXE, below) that you insert into
* the Program EXEs file group. Run the setup with and without a password
* assigned to the Program Files component (remember to rebuild your media
* each time).
*
* You can also create billboards (name them Bbrd1.bmp, Bbrd2.bmp, and
* Bbrd3.bmp) and add them to the project in the Setup Files pane under
* the Language Independent folder.
*
* This example script installs files, adds an icon to the Start Programs
* menu, and provides uninstallation functionality.
*
\*-----*/

#include "sddialog.h"

// Define strings. In a real setup you would define these in your string tables
// and precede each constant with @ to use them in your script.
#define COMP_SELECT_TITLE           "Select Components"
#define COMP_SELECT_MSG             "Select components and subcomponents to
install."
#define COMP_PROGRAMFILES_DISPLAYTEXT "Program Files"
#define PASSWORD_PROMPT             "Please enter the password."
#define PASSWORD_ERRMSG             "Password incorrect. Please enter again."
#define TITLE_MAIN                  "Word Processor"
#define TITLE_CAPTIONBAR            "Word Processor Setup"
#define APPBASE_PATH                "Your Company\\Word Processor"
```

```

#define COMPANY_NAME           "Your Company"
#define PRODUCT_NAME          "Word Processor"
#define PRODUCT_VERSION       "1.0"
#define PRODUCT_KEY           "Word Processor"
#define DEINSTALL_KEY         "Word Processor"
#define UNINSTALL_NAME        "Word Processor"
#define ADDINGICON             "Adding program icon to the Start Programs
menu..."
#define PROGRAMDIR             "Program"
#define DEFAULT_FOLDER_NAME    ""
#define APP_NAME               "Word Processor"
#define COMPLETE_MSG           "Setup is complete. You can run Word
Processor from the Start Programs menu."
#define MAIN_EXE               "WRITE.EXE"
#define SETUPTYPE_TITLE        "Setup Type Selection"
#define SETUPTYPE_MSG          "Please select a setup type."
#define SETUPTYPE_CUSTOM       "Custom"

```

```

// Global variable declarations.
STRING svData, svLogFile, szProgram, szComponent, svResult, svSetupType, svDir;
BOOL    bInitStepsDone, bPwdValid;
NUMBER  nvData, nvDisk, nResult;

```

```

// Function declarations.
prototype SetUpFileTransfer();
prototype HandleComponentError(NUMBER);
prototype FinishSetup();

```

```

program
    SetUpFileTransfer();

    // Get the setup type.
    Disable( BACKBUTTON );
    svDir = TARGETDIR;
    SdSetupTypeEx( SETUPTYPE_TITLE, SETUPTYPE_MSG, "", svSetupType, 0 );
    // If user selected Custom setup type, display component selection dialog.
    if ( svSetupType = SETUPTYPE_CUSTOM ) then
        SdComponentDialog( COMP_SELECT_TITLE, COMP_SELECT_MSG, svDir, "" );
    endif;
    Enable( BACKBUTTON );

    // If the Program Files component is selected and there is a password
    // associated with it, the user must input the password and
    // it must be properly validated.
    nResult = FALSE;
    nvData = FALSE;
    nResult = ComponentIsItemSelected( MEDIA, COMP_PROGRAMFILES_DISPLAYTEXT );
    ComponentGetData( MEDIA, COMP_PROGRAMFILES_DISPLAYTEXT,
        COMPONENT_FIELD_PASSWORD, nvData, svData);
    if ( nResult && nvData ) then
        bPwdValid = FALSE;
        Disable( BACKBUTTON ); // Back button not needed or supported here.
        while ( !bPwdValid )
            AskText( PASSWORD_PROMPT, "", svResult );
            nResult = ComponentValidate( MEDIA, COMP_PROGRAMFILES_DISPLAYTEXT,
                svResult );

            if ( nResult = 0 ) then
                bPwdValid = TRUE;
            else
                MessageBox( PASSWORD_ERRMSG, SEVERE );
            endif;
        endwhile;
    endif;
endprogram

```

```

    Enable( BACKBUTTON ); // Restore Back button's default status.
endif;

// Set up the progress indicator, including locations for
// progress indicator, information gauges, and billboards.
PlaceWindow( FEEDBACK, LOWER_LEFT, LOWER_LEFT, LOWER_LEFT );
PlaceWindow( STATUSDLG, CENTERED, LOWER_RIGHT, LOWER_RIGHT );
PlaceWindow( BILLBOARD, CENTERED, CENTERED, CENTERED );
Enable( STATUSDLG );
Enable( INDVFILESTATUS );

// Indicate the final percentage the progress bar is to show when the
// following file transfer operation is complete.
StatusUpdate( ON, 95 );

// Transfer files to the target system. ComponentMoveData will prompt
// for next disk in a floppy disk installation.
nResult=ComponentMoveData( MEDIA, nvDisk, 0);

// To see the ComponentError function in action in the following call
// to the HandleComponentError user-defined function, remark out the
// while...endwhile loop above in which ComponentValidate is called,
// make sure a password is associated with the Program Files component
// in the IDE, rebuild the media, and run the setup.
HandleComponentError( nResult );

FinishSetup();
endprogram

/*-----*\
*
* Function: SetupFileTransfer()
*
* Purpose: This function sets up file transfer. The main reason for
*          abstracting this process into this function is to make it
*          easy to see the function calls this sample script demonstrates.
*
*\-----*/
function SetUpFileTransfer()
begin
    // Set up the installation screen.
    Enable( FULLWINDOWMODE );
    SetTitle( TITLE_MAIN, 24, WHITE );
    SetTitle( TITLE_CAPTIONBAR, 0, BACKGROUNDCAPTION );
    Enable( BACKGROUND );

    // Welcome the user, check that the system meets minimum requirements,
    // and verify the destination location.
    bInitStepsDone = FALSE;
    while (!bInitStepsDone)
        Disable( BACKBUTTON );
        Welcome( "", 0 );
        Enable( BACKBUTTON );

        TARGETDIR = PROGRAMFILES ^ APPBASE_PATH;
        if (AskDestPath( "", "", TARGETDIR, 0 ) != BACK) then
            bInitStepsDone = TRUE;
        endif;
    endwhile;

    // Set installation information required for registry entries and for
    // the following call to DeinstallStart.

```

```

InstallationInfo( COMPANY_NAME, PRODUCT_NAME,
                 PRODUCT_VERSION, PRODUCT_KEY );

// Initialize the uninstallation log file, including registry entry.
svLogFile = "Uninst.isu";
DeinstallStart( TARGETDIR, svLogFile, DEINSTALL_KEY, 0 );
RegDBSetItem( REGDB_UNINSTALL_NAME, UNINSTALL_NAME );
end;

/*-----*\
*
* Function:  HandleComponentError
*
* Purpose:  This function evaluates the value returned by a Component...
*           function and if the value is less than zero, displays the error
*           number and aborts the setup.
*
*-----*/
function HandleComponentError( nResult )

    NUMBER  nvError;
    STRING  svMedia, svComponent, svFileGroup, svFile;

begin
    if(nResult < 0) then
        ComponentError( svMedia, svComponent, svFileGroup, svFile, nvError );
        sprintfBox( INFORMATION, "Data Transfer Error Information",
                  "ComponentError returned the following data transfer error.\n" +
                  "Setup will now abort.\n\n" +
                  "Media Name: %s\nComponent: %s\nFile Group: %s\n" +
                  "File: %s\nError Number: %ld",
                  svMedia, svComponent, svFileGroup, svFile, nvError );
        abort;
    endif;
end;

/*-----*\
*
* Function:  FinishSetup()
*
* Purpose:  This function finishes the setup. The main reason for
*           abstracting this process into this function is to make it
*           easy to see the function calls this sample script demonstrates.
*
*-----*/
function FinishSetup()

begin
    // Indicate the final percentage the progress bar is to show when the
    // following file transfer operation is complete.
    StatusUpdate(ON, 99);

    // Increment progress bar to 99% for creation of Start Programs menu icon.
    SetStatusWindow (96 , ADDINGICON );

    // Add the APP_NAME icon to the DEFAULT_FOLDER_NAME folder.
    szProgram = TARGETDIR ^ PROGRAMDIR ^ MAIN_EXE;
    LongPathToQuote( szProgram, TRUE );
    AddFolderIcon( DEFAULT_FOLDER_NAME, APP_NAME, szProgram,
                  TARGETDIR ^ PROGRAMDIR,
                  "", 0, "", REPLACE );

    Delay( 1 );

```

```
// Disable the progress indicator and its settings.
Disable ( INDVFILESTATUS );
Disable( STATUSDLG );

// Announce setup complete and offer to view Readme file.
MessageBox( COMPLETE_MSG, INFORMATION );
end;

#define SD_SINGLE_DIALOGS 1
#define SD_SETUPTYPEEX 1
#define SD_COMPONENTDIALOG 1

#include "sddialog.rul"

// Source file: Is5fn603.rul
```

{button ,PI("",`comment')} Feedback



SdSetupTypeEx

Syntax

SdSetupTypeEx (szTitle, szMsg, szReserved, svSetupType, nReserved);

Description

The SdSetupTypeEx function displays a dialog box that enables the user to select the setup type when you specify setup types beyond Typical, Compact, and Custom.



[View sample dialog](#)

Parameters

szTitle

Specifies the title of the dialog box. To display the default title ("Setup Type"), pass a null string ("") in this parameter.

szMsg

Specifies the message to be displayed in the dialog box. To display the default instructions for this dialog box, pass a null string ("") in this parameter.

szReserved

Pass a null string ("") in this parameter in this parameter. No other value is allowed.

svSetupType

Specifies a default setup type and returns the setup type selected by the end user. To make the first setup in the list the default selection, pass a null string ("") in this parameter. This string returned in this parameters will match the name of the setup type specified in the IDE.

nReserved

Pass zero in this parameter. No other value is allowed.

Return values

0

Indicates that SdSetupTypeEx was successful.

BACK

Indicates that the Back button was clicked.

{button ,JI('LANGREF.HLP>Examples','SdSetupTypeEx_example')} [Example](#)

{button ,AL('SdAskOptions;SdProductName;SdSetupType;SetupType;ComponentSetupTypeSet;Getting setup type selection from the end user;Display text in the Description field of SdSetupTypeEx;Display a different name for a setup type in SdSetupTypeEx;Exit conditions;Caching dialogs;Displaying special characters and TM in dialogs and titles;Which fonts InstallShield dialog boxes use;Modifying Sd dialog box source files',0,',')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



SdSetupTypeEx example

```
/*-----*\
*
* This example demonstrates SdSetupTypeEx, SdComponentDialog,
* ComponentIsItemSelected, ComponentGetData, ComponentValidate,
* ComponentMoveData, ComponentError, and PlaceWindow.
*
* Comments: To run this example script, create a project with the
*           following components (c), subcomponents (sc), and file
*           groups (fg):
*
*           (c) Program Files
*               (fg) Program DLLS
*               (fg) Program EXEs
*           (c) Example Files
*               (sc) Small Documents
*                   (fg) Small Document Examples
*               (sc) Books
*                   (fg) Book Examples
*               (sc) Graphics
*                   (fg) Graphic Examples
*           (c) Help Files
*               (fg) Help Files
*           (c) Utilities
*               (sc) Grammar Checker
*                   (fg) Grammar Checker
*               (sc) Art Studio
*                   (fg) Art Studio
*           (c) Evaluation Copy
*               (fg) Evaluation Copy
*               (fg) Help Files
*
* Insert "dummy" files into the file groups. Make sure you define the
* correct filename for the main EXE (MAIN_EXE, below) that you insert into
* the Program EXEs file group. Run the setup with and without a password
* assigned to the Program Files component (remember to rebuild your media
* each time).
*
* You can also create billboards (name them Bbrd1.bmp, Bbrd2.bmp, and
* Bbrd3.bmp) and add them to the project in the Setup Files pane under
* the Language Independent folder.
*
* This example script installs files, adds an icon to the Start Programs
* menu, and provides uninstallation functionality.
*
\*-----*/

#include "sddialog.h"

// Define strings. In a real setup you would define these in your string tables
// and precede each constant with @ to use them in your script.
#define COMP_SELECT_TITLE           "Select Components"
#define COMP_SELECT_MSG             "Select components and subcomponents to
install."
#define COMP_PROGRAMFILES_DISPLAYTEXT "Program Files"
#define PASSWORD_PROMPT             "Please enter the password."
#define PASSWORD_ERRMSG             "Password incorrect. Please enter again."
#define TITLE_MAIN                  "Word Processor"
#define TITLE_CAPTIONBAR            "Word Processor Setup"
#define APPBASE_PATH                 "Your Company\\Word Processor"
```

```

#define COMPANY_NAME           "Your Company"
#define PRODUCT_NAME          "Word Processor"
#define PRODUCT_VERSION       "1.0"
#define PRODUCT_KEY           "Word Processor"
#define DEINSTALL_KEY        "Word Processor"
#define UNINSTALL_NAME       "Word Processor"
#define ADDINGICON           "Adding program icon to the Start Programs
menu..."
#define PROGRAMDIR            "Program"
#define DEFAULT_FOLDER_NAME   ""
#define APP_NAME              "Word Processor"
#define COMPLETE_MSG         "Setup is complete. You can run Word
Processor from the Start Programs menu."
#define MAIN_EXE              "WRITE.EXE"
#define SETUPTYPE_TITLE      "Setup Type Selection"
#define SETUPTYPE_MSG        "Please select a setup type."
#define SETUPTYPE_CUSTOM     "Custom"

```

```

// Global variable declarations.
STRING svData, svLogFile, szProgram, szComponent, svResult, svSetupType, svDir;
BOOL   bInitStepsDone, bPwdValid;
NUMBER nvData, nvDisk, nResult;

```

```

// Function declarations.
prototype SetUpFileTransfer();
prototype HandleComponentError(NUMBER);
prototype FinishSetup();

```

```

program
    SetUpFileTransfer();

    // Get the setup type.
    Disable( BACKBUTTON );
    svDir = TARGETDIR;
    SdSetupTypeEx( SETUPTYPE_TITLE, SETUPTYPE_MSG, "", svSetupType, 0 );
    // If user selected Custom setup type, display component selection dialog.
    if ( svSetupType = SETUPTYPE_CUSTOM ) then
        SdComponentDialog( COMP_SELECT_TITLE, COMP_SELECT_MSG, svDir, "" );
    endif;
    Enable( BACKBUTTON );

    // If the Program Files component is selected and there is a password
    // associated with it, the user must input the password and
    // it must be properly validated.
    nResult = FALSE;
    nvData = FALSE;
    nResult = ComponentIsItemSelected( MEDIA, COMP_PROGRAMFILES_DISPLAYTEXT );
    ComponentGetData( MEDIA, COMP_PROGRAMFILES_DISPLAYTEXT,
        COMPONENT_FIELD_PASSWORD, nvData, svData);
    if ( nResult && nvData ) then
        bPwdValid = FALSE;
        Disable( BACKBUTTON ); // Back button not needed or supported here.
        while ( !bPwdValid )
            AskText( PASSWORD_PROMPT, "", svResult );
            nResult = ComponentValidate( MEDIA, COMP_PROGRAMFILES_DISPLAYTEXT,
                svResult );

            if ( nResult = 0 ) then
                bPwdValid = TRUE;
            else
                MessageBox( PASSWORD_ERRMSG, SEVERE );
            endif;
        endwhile;
    endif;
endprogram

```

```

    Enable( BACKBUTTON ); // Restore Back button's default status.
endif;

// Set up the progress indicator, including locations for
// progress indicator, information gauges, and billboards.
PlaceWindow( FEEDBACK, LOWER_LEFT, LOWER_LEFT, LOWER_LEFT );
PlaceWindow( STATUSDLG, CENTERED, LOWER_RIGHT, LOWER_RIGHT );
PlaceWindow( BILLBOARD, CENTERED, CENTERED, CENTERED );
Enable( STATUSDLG );
Enable( INDVFILESTATUS );

// Indicate the final percentage the progress bar is to show when the
// following file transfer operation is complete.
StatusUpdate( ON, 95 );

// Transfer files to the target system. ComponentMoveData will prompt
// for next disk in a floppy disk installation.
nResult=ComponentMoveData( MEDIA, nvDisk, 0);

// To see the ComponentError function in action in the following call
// to the HandleComponentError user-defined function, remark out the
// while...endwhile loop above in which ComponentValidate is called,
// make sure a password is associated with the Program Files component
// in the IDE, rebuild the media, and run the setup.
HandleComponentError( nResult );

FinishSetup();
endprogram

/*-----*\
*
* Function: SetupFileTransfer()
*
* Purpose: This function sets up file transfer. The main reason for
*          abstracting this process into this function is to make it
*          easy to see the function calls this sample script demonstrates.
*
*\-----*/
function SetUpFileTransfer()
begin
    // Set up the installation screen.
    Enable( FULLWINDOWMODE );
    SetTitle( TITLE_MAIN, 24, WHITE );
    SetTitle( TITLE_CAPTIONBAR, 0, BACKGROUNDCAPTION );
    Enable( BACKGROUND );

    // Welcome the user, check that the system meets minimum requirements,
    // and verify the destination location.
    bInitStepsDone = FALSE;
    while (!bInitStepsDone)
        Disable( BACKBUTTON );
        Welcome( "", 0 );
        Enable( BACKBUTTON );

        TARGETDIR = PROGRAMFILES ^ APPBASE_PATH;
        if (AskDestPath( "", "", TARGETDIR, 0 ) != BACK) then
            bInitStepsDone = TRUE;
        endif;
    endwhile;

    // Set installation information required for registry entries and for
    // the following call to DeinstallStart.

```

```

InstallationInfo( COMPANY_NAME, PRODUCT_NAME,
                 PRODUCT_VERSION, PRODUCT_KEY );

// Initialize the uninstallation log file, including registry entry.
svLogFile = "Uninst.isu";
DeinstallStart( TARGETDIR, svLogFile, DEINSTALL_KEY, 0 );
RegDBSetItem( REGDB_UNINSTALL_NAME, UNINSTALL_NAME );
end;

/*-----*\
*
* Function:  HandleComponentError
*
* Purpose:  This function evaluates the value returned by a Component...
*           function and if the value is less than zero, displays the error
*           number and aborts the setup.
*
*-----*/
function HandleComponentError( nResult )

    NUMBER  nvError;
    STRING  svMedia, svComponent, svFileGroup, svFile;

begin
    if(nResult < 0) then
        ComponentError( svMedia, svComponent, svFileGroup, svFile, nvError );
        sprintfBox( INFORMATION, "Data Transfer Error Information",
                  "ComponentError returned the following data transfer error.\n" +
                  "Setup will now abort.\n\n" +
                  "Media Name: %s\nComponent: %s\nFile Group: %s\n" +
                  "File: %s\nError Number: %ld",
                  svMedia, svComponent, svFileGroup, svFile, nvError );
        abort;
    endif;
end;

/*-----*\
*
* Function:  FinishSetup()
*
* Purpose:  This function finishes the setup. The main reason for
*           abstracting this process into this function is to make it
*           easy to see the function calls this sample script demonstrates.
*
*-----*/
function FinishSetup()

begin
    // Indicate the final percentage the progress bar is to show when the
    // following file transfer operation is complete.
    StatusUpdate(ON, 99);

    // Increment progress bar to 99% for creation of Start Programs menu icon.
    SetStatusWindow (96 , ADDINGICON );

    // Add the APP_NAME icon to the DEFAULT_FOLDER_NAME folder.
    szProgram = TARGETDIR ^ PROGRAMDIR ^ MAIN_EXE;
    LongPathToQuote( szProgram, TRUE );
    AddFolderIcon( DEFAULT_FOLDER_NAME, APP_NAME, szProgram,
                  TARGETDIR ^ PROGRAMDIR,
                  "", 0, "", REPLACE );

    Delay( 1 );

```

```
// Disable the progress indicator and its settings.
Disable ( INDVFILESTATUS );
Disable( STATUSDLG );

// Announce setup complete and offer to view Readme file.
MessageBox( COMPLETE_MSG, INFORMATION );
end;

#define SD_SINGLE_DIALOGS 1
#define SD_SETUPTYPEEX 1
#define SD_COMPONENTDIALOG 1

#include "sddialog.rul"

// Source file: Is5fn603.rul
```

{button ,PI("",`comment')} [Feedback](#)



Ez configuration file functions

Ez configuration file functions modify the default system configuration file. Unless changed by a call to ConfigSetFileName, that file is the Config.sys file that was executed by the system during the boot sequence.

EzConfigAddDriver

Adds a device driver statement to the default system configuration file.

EzConfigAddString

Adds a statement or line of text to the default system configuration file.

EzConfigGetValue

Retrieves the value of a system configuration file parameter, such as FILES or BUFFERS.

EzConfigSetValue

Sets the value of a system configuration file parameter, such as FILES or BUFFERS.



Each of these functions opens the default system configuration file, performs its assigned task, and then saves the file back to disk. You don't load and save the configuration file as you do with the advanced configuration file functions.



EzConfigAddDriver

{button ,PI("",`comment')} [Feedback](#)

Syntax

EzConfigAddDriver (szDriver, szRefKey, nPosition);

Description

The EzConfigAddDriver function adds a device driver statement to the default system configuration file. You can specify the position of the driver statement relative to another driver statement. For example, an application may require loading a device driver before or after the Windows Himem.sys driver.

Unless changed by a call to ConfigSetFileName, the default system configuration file is the Config.sys file that was executed by the system during the boot sequence. To make another file the default system configuration file, call ConfigSetFileName. To determine the fully qualified name of the default system configuration file, call ConfigGetFileName.



Do not mix the Ez configuration file functions and the advanced configuration file functions. After calling ConfigFileLoad, you cannot call any of the Ez configuration file functions until you call ConfigFileSave to save the file.

Parameters

szDriver

Specifies the fully qualified name of the driver to add to the file. If the driver already exists in one or more places in the system configuration file, this function replaces only the last occurrence of the line containing the driver.

szRefKey

Specifies the name of the device driver relative to which you are adding szDriver.

nPosition

Indicates whether szDriver is to be added before or after the driver specified by szRefKey.

BEFORE

The statement is added before the line containing szRefKey. If szRefKey contains a null string (""), the driver

is inserted as the first driver in the system configuration file.

AFTER

The statement is added after the line containing szRefKey. If szRefKey contains a null string (""), the driver is inserted as the last driver in the system configuration file.

Return values

0

EzConfigAddDriver successfully added the device driver statement to the file.

< 0

EzConfigAddDriver was unable to add the driver statement.

{button ,JI('LANGREF.HLP>Examples','EzConfigAddDriver_example')} [Example](#)

{button ,AL('EzConfigAddString;EzConfigGetValue;EzConfigSetValue',0,'')} [See also](#)



EzConfigAddDriver example

```
/*-----*\
*
* This example illustrates the usage of the EzConfigAddDriver function.
*
* The first time EzConfigAddDriver is called, a new line of text is added
* after the last line.
*
* The second time, a new line of text is added before the first line.
*
* The third time, the driver C:\\DRDOS\\HIDOS.SYS is added after the key
* HIMEM.SYS
*
* The fourth time, the driver EXAPP.SYS is added before the key HIMEM.SYS.
*
* NOTE: In order for this script to run properly, it is necessary for the
*       EXAMPLE_SYS constant to be set to a configuration file which is
*       located on the target system.
*
\*-----*/

#define EXAMPLE_SYS "EXAMPLE\\EXAMPLE2.SYS"

    STRING szConfigFile, szDriver, szRefKey;
    NUMBER nPosition;

program

    // Set the config file to be edited to EXAMPLE_SYS.
    szConfigFile = EXAMPLE_SYS;
    ConfigSetFileName(szConfigFile);

    MessageBox(szConfigFile, INFORMATION);

    szDriver = "C:\\NEW\\MYAPP.DRV";
    szRefKey = "";
    nPosition = AFTER;
/*-----*\
*
* EzConfigAddDriver adds a driver to the end of the EXAMPLE_SYS file.
*
\*-----*/
    if (EzConfigAddDriver(szDriver, szRefKey, nPosition) < 0) then

        MessageBox("First call to EzConfigAddDriver failed.", SEVERE);
    else

        MessageBox("First call to EzConfigAddDriver successful.", INFORMATION);
    endif;

    szDriver = "C:\\SYSTEM\\DMDRVR.BIN";
    szRefKey = "";
    nPosition = BEFORE;
/*-----*\
*
* EzConfigAddDriver adds a driver to the top of the EXAMPLE_SYS file.
*
\*-----*/
    if (EzConfigAddDriver(szDriver, szRefKey, nPosition) < 0) then
```

```

    MessageBox("Second call to EzConfigAddDriver failed.", SEVERE);
else

    MessageBox("Second call to EzConfigAddDriver successful.", INFORMATION);
endif;

szDriver = "C:\\DRDOS\\HIDOS.SYS";
szRefKey = "HIMEM.SYS";
nPosition = AFTER;
/*-----*\
*
* EzConfigAddDriver adds the "HIDOS.SYS" driver after the "HIMEM.SYS" key to
* the EXAMPLE_SYS file.
*
\*-----*/
if (EzConfigAddDriver(szDriver, szRefKey, nPosition) < 0) then

    MessageBox("Third call to EzConfigAddDriver failed.", SEVERE);
else

    MessageBox("Third call to EzConfigAddDriver successful.", INFORMATION);
endif;

szDriver = "EXAPP.SYS";
szRefKey = "HIMEM.SYS";
nPosition = BEFORE;

/*-----*\
*
* EzConfigAddDriver adds the "EXAPP.SYS" driver before the "HIMEM.SYS" key in
* the EXAMPLE_SYS file.
*
\*-----*/
if (EzConfigAddDriver(szDriver, szRefKey, nPosition) < 0) then

    MessageBox("Fourth call to EzConfigAddDriver failed.", SEVERE);
else

    MessageBox("Fourth call to EzConfigAddDriver successful.", INFORMATION);
endif;

endprogram

// Source file: Is5fn065.rul

```

{button ,PI("",`comment')} [Feedback](#)



EzConfigAddString

Syntax

```
EzConfigAddString (szLine, szRefKey, nOptions);
```

Description

The EzConfigAddString function adds a line of text to the default system configuration file. You can specify the position of the line you add in reference to another statement in the file.

Unless changed by a call to ConfigSetFileName, the default system configuration file is the Config.sys file that was executed by the system during the boot sequence. To make another file the default system configuration file, call ConfigSetFileName. To determine the fully qualified name of the default system configuration file, call ConfigGetFileName.



Do not mix the Ez configuration file functions and the advanced configuration file functions. After calling ConfigFileLoad, you cannot call any of the Ez configuration file functions until you call ConfigFileSave to save the file.

Parameters

szLine

Specifies the line of text to add to the system configuration file.

szRefKey

Specifies the reference key relative to which you want to position szLine in the system configuration file. EzConfigAddString searches the system configuration file for the reference key and places the contents of the parameter szLine before or after the line containing the key, depending on the constants you enter in nOptions.

nOptions

Indicates whether the line specified by szLine is to be added before or after the line containing szRefKey. Pass one of the following predefined constants in this parameter:

BEFORE

The line specified by szLine is added before the line containing szRefKey. If szRefKey contains a null string (""), szLine is inserted as the first line of the system configuration file.

AFTER

The line specified by szLine is added after the line containing szRefKey. If szRefKey contains a null string (""), szLine is inserted as the last line of the system configuration file.

Return values

0

EzConfigAddString successfully added the string to the default system configuration file.

< 0

EzConfigAddString was unable to add the text string.

{button ,JI('LANGREF.HLP>Examples','EzConfigAddString_example')} Example

{button ,AL('EzConfigAddDriver;EzConfigGetValue;EzConfigSetValue',0,','')} See also

{button ,PI('','comment')} Feedback



EzConfigAddString example

```
/*-----*\
*
* This example illustrates the usage of the EzConfigAddString function.
*
* The first time EzConfigAddString is called, a new line of text is added
* after the last line.
*
* The second time, a new line of text is added before the first line.
*
* The third time, the line FASTOPEN=512 is added after the key LASTDRIVE.
*
* The fourth time, the line EXAPPHI=ON is added before the key EXAPPHI.SYS.
*
* NOTE: In order for this script to run properly, it is necessary for the
*       EXAMPLE_SYS constant to be set to a .SYS file which is located on the
*       target system.
*
/*-----*/

#define EXAMPLE_SYS "EXAMPLE\\EXAMPLE.SYS"

    STRING szConfigFile, szLine, szRefKey;
    NUMBER nOptions;

program

    // Set the config file to be edited to EXAMPLE_SYS.
    szConfigFile = EXAMPLE_SYS;
    ConfigSetFileName(szConfigFile);

    szLine = "SHELL=C:\\COMMAND.COM /P /E:512";
    szRefKey = "";
    nOptions = AFTER;

/*-----*\
*
* EzConfigAddString adds a line to the end of the EXAMPLE_SYS file.
*
/*-----*/
    if (EzConfigAddString(szLine, szRefKey, nOptions) < 0) then

        MessageBox("First call to EzConfigAddString failed.", SEVERE);
    else

        MessageBox("First call to EzConfigAddString successful.", INFORMATION);
    endif;

    szLine = "DEVICE=C:\\SYSTEM\\DMDRVR.BIN";
    szRefKey = "";
    nOptions = BEFORE;

/*-----*\
*
* EzConfigAddString adds a line to the top of the EXAMPLE_SYS file.
*
/*-----*/
    if (EzConfigAddString(szLine, szRefKey, nOptions) < 0) then

        MessageBox("Second call to EzConfigAddString failed.", SEVERE);
```

```

else

    MessageBox("Second call to EzConfigAddString successful.", INFORMATION);
endif;

szLine    = "FASTOPEN=512";
szRefKey  = "LASTDRIVE";
nOptions  = AFTER;

/*-----*\
*
* EzConfigAddString adds a line after the "LASTDRIVE" key to the EXAMPLE_SYS
* file.
*
\*-----*/
if (EzConfigAddString(szLine, szRefKey, nOptions) < 0) then

    MessageBox("Third call to EzConfigAddString failed.", SEVERE);
else

    MessageBox("Third call to EzConfigAddString successful.", INFORMATION);
endif;

szLine    = "EXAPPHI=ON";
szRefKey  = "EXAPPHI.SYS";
nOptions  = BEFORE;

/*-----*\
*
* EzConfigAddString add a line before the "EXAPPHI.SYS" key in the
* EXAMPLE_SYS file.
*
\*-----*/
if (EzConfigAddString(szLine, szRefKey, nOptions) < 0) then

    MessageBox("Fourth call to EzConfigAddString failed.", SEVERE);
else

    MessageBox("Fourth call to EzConfigAddString successful.", INFORMATION);
endif;

endprogram

// Source file: Is5fn066.rul

```

{button ,PI("",`comment')} [Feedback](#)



EzConfigGetValue

Syntax

EzConfigGetValue (szRefKey, nvValue);

Description

The EzConfigGetValue function retrieves the numeric value of a parameter, such as FILES or BUFFERS, from the default system configuration file. Unless changed by a call to ConfigSetFileName, the default system configuration file is the Config.sys file that was executed by the system during the boot sequence. To make another file the default system configuration file, call ConfigSetFileName. To determine the fully qualified name of the default system configuration file, call ConfigGetFileName.



Do not mix the Ez configuration file functions and the advanced configuration file functions. After calling ConfigFileLoad, you cannot call any of the Ez configuration file functions until you call ConfigFileSave to save the file..

Parameters

szRefKey

Specifies the name of the parameter whose value is to be retrieved.

nvValue

Returns the numeric value.

Return values

0

EzConfigGetValue successfully retrieved the value.

< 0

EzConfigGetValue was unable to retrieve the value.

{button ,JI('LANGREF.HLP>Examples','EzConfigGetValue_example')} [Example](#)

{button ,AL('EzConfigAddDriver;EzConfigAddString;EzConfigSetValue',0,','')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



EzConfigGetValue example

```
/*-----*\
 *
 * This example illustrates the usage of the EzConfigGetValue function.
 *
 * EzConfigGetValue is called to retrieve a value from a key in a
 * configuration file.
 *
 * NOTE: In order for this script to run properly, it is necessary to set the
 *       EXAMPLE_SYS constant to a configuration file which exists on the
 *       target system.
 *
 \*-----*/

#define EXAMPLE_SYS "EXAMPLE\\EXAMPLE.SYS"

    STRING szConfigFile, szRefKey, szTitle;
    NUMBER nvValue;

program

    // Set the default config file to the example file.
    szConfigFile = EXAMPLE_SYS;
    ConfigSetFileName(szConfigFile);

    szRefKey = "FILES";
/*-----*\
 *
 * EzConfigGetValue retrieves the numeric value of the FILES key in the
 * .SYS file. This value is displayed in a message box.
 *
 \*-----*/
    if (EzConfigGetValue(szRefKey, nvValue) < 0) then

        MessageBox("EzConfigGetValue failed.", SEVERE);
    else

        szTitle = "EzConfigGetValue Example";
        sprintfBox(INFORMATION, szTitle, "FILES = %d", nvValue);
    endif;

endprogram

// Source file: Is5fn067.rul
```

{button ,PI("",`comment')} [Feedback](#)



EzConfigSetValue

Syntax

```
EzConfigSetValue (szRefKey, nValue);
```

Description

The EzConfigSetValue function sets the value of a command in the default system configuration file. Unless changed by a call to ConfigSetFileName, the default system configuration file is the Config.sys file that was executed by the system during the boot sequence. To make another file the default system configuration file, call ConfigSetFileName. To determine the fully qualified name of the default system configuration file, call ConfigGetFileName.



Do not mix the Ez configuration file functions and the advanced configuration file functions. After calling ConfigFileLoad, you cannot call any of the Ez configuration file functions until you call ConfigFileSave to save the file.

Parameters

szRefKey

Specifies the command whose value is to be changed or added in the default system configuration file. If InstallShield cannot find the key, it is added for you.

nValue

Specifies the new value of the command in szRefKey.

Return values

0

EzConfigSetValue successfully set the value.

< 0

EzConfigSetValue was unable to set the value.

{button ,JI('LANGREF.HLP>Examples','EzConfigSetValue_example')} [Example](#)

{button ,AL('EzConfigAddDriver;EzConfigAddString;EzConfigGetValue',0,'')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



EzConfigSetValue example

```
/*-----*\
*
* This example illustrates the usage of the EzConfigSetValue function.
*
* EzConfigSetValue is called to set the value of a key in the target
* configuration file.
*
* NOTE: In order for this script to run properly, it is advised to set the
*       DEF_CONFIG constant to a valid configuration file on the target
*       system.
*
\*-----*/

#define DEF_CONFIG "EXAMPLE\\EXAMPLE.SYS"

    STRING szRefKey, szTitle, szMsg;
    NUMBER nValue;

program

    // Set the default config file to the example config file.
    ConfigSetFileName(DEF_CONFIG);

    szRefKey = "BUFFERS";
    nValue   = 30;
    szTitle  = "EzConfigSetValue Example";
    szMsg    = "Successful.\n\n%s was set to %d.";
/*-----*\
*
* The following sets BUFFERS to 30 by writing BUFFERS=30 onto the file.
*
\*-----*/
    if (EzConfigSetValue("BUFFERS", 30) < 0) then

        MessageBox("First call to EzConfigSetValue failed.", SEVERE);
    else

        sprintfBox(INFORMATION, szTitle, szMsg, szRefKey, nValue);
    endif;

    szRefKey = "FILES";
    nValue   = 20;
/*-----*\
*
* The second call to EzConfigSetValue adds the FILES key and sets it to 20.
*
\*-----*/
    if (EzConfigSetValue("FILES", 20) < 0) then

        MessageBox("Second call to EzConfigSetValue failed.", SEVERE);
    else

        sprintfBox(INFORMATION, szTitle, szMsg, szRefKey, nValue);
    endif;

endprogram

// Source file: Is5fn068.rul
```

{button ,PI("", `comment`)} Feedback



Advanced configuration file functions

The advanced configuration file functions provide the advanced developer greater flexibility and more control over system configuration files than do the Ez configuration file functions. To access and edit a system configuration file with these advanced functions, start by calling `ConfigFileLoad`. Most of the other functions can be called only after a system configuration file has been opened with `ConfigFileLoad`. When you are finished editing the system configuration file, call `ConfigFileSave` to save your changes. Note that the functions `ConfigGetFileName` and `ConfigSetFileName` can be used with both advanced and Ez configuration file functions.

ConfigAdd

Adds a statement to a system configuration file that has been loaded in memory.

ConfigDelete

Deletes an item from a system configuration file.

ConfigFileLoad

Loads a system configuration file into memory for editing.

ConfigFileSave

Saves a system configuration file that has been loaded into memory with `ConfigFileLoad`.

ConfigFind

Searches for an item in a system configuration file.

ConfigGetFileName

Retrieves the fully qualified name of default system configuration file.

ConfigGetInt

Retrieves a value from a system configuration file.

ConfigMove

Moves an item within a system configuration file.

ConfigSetFileName

Specifies the fully qualified filename of a system configuration file.

ConfigSetInt

Sets a value in a system configuration file.

Related Function

SdShowFileMods

Creates a dialog box displaying proposed file changes and offering options on how to proceed.

{button ,PI("",`comment')} [Feedback](#)



ConfigAdd

Syntax

ConfigAdd (szKey, szValue, szRefKey, nOptions);

Description

The ConfigAdd function adds a statement to the system configuration file that has been loaded into memory with [ConfigFileLoad](#). You can specify the position of the statement relative to a reference key, or you can add the statement as the first or last line of the file. You can also replace an existing line in the file.

Before calling ConfigAdd, you must first call ConfigFileLoad to load the system configuration file into memory. After you edit the file, call [ConfigFileSave](#) to save the file.



Do not mix the Ez configuration file functions with the advanced configuration file functions. After calling ConfigFileLoad, you cannot use the Ez configuration file functions until you call ConfigFileSave to save your changes.

Parameters

szKey

Specifies the keyword in the statement that is being added to the system configuration file.

szValue

Specifies the value of the keyword that is being added to the system configuration file.

szRefKey

Specifies the reference key relative to which you are adding szKey in the system configuration file. If you pass a null string (""), the line is added as the first or last line in the file, depending on which nOptions constant you use.

nOptions

Specifies whether the line is to be added before or after the line containing the reference key, or whether the line replaces an existing line. Pass one of the following predefined constants in this parameter:

BEFORE

The statement is added before the line containing szRefKey. If szRefKey is a null string (""), the statement is added as the first line of the file.

AFTER

The statement is added after the line containing szRefKey. If szRefKey is a null string (""), the statement is added as the last line of the file.

REPLACE

The statement replaces an existing line in the file. If multiple lines with same key exist, only the last line is replaced. If a line to be replaced does not exist in the file, the new line is added as the last line of the file.

Return values

0

ConfigAdd successfully added the statement to the specified system configuration file.

< 0

ConfigAdd was unable to add the statement to the specified system configuration file.

Comments

- n When the ConfigAdd function replaces a line in a system configuration file, it compares the reference keys in the two lines.
- n A reference key is a substring that identifies the line. For example, in the following statement, the reference

key is Kybrd.drv.

```
DEVICE=C:\Windows\System\Kybrd.drv /1024 /C:345
```

In the next statement, the reference key is PATH:

```
SET PATH=C:\Windows;C:\Windows\System
```

{button ,JI('LANGREF.HLP>Examples','ConfigAdd_example')} Example

{button ,AL('ConfigDelete;ConfigFileLoad;ConfigFileSave;ConfigFind;ConfigGetFileName;ConfigGetInt;ConfigMove
;ConfigSetFileName;ConfigSetInt',0,'')} See also

{button ,PI('','comment')} Feedback



ConfigAdd example

```
/*-----*\
*
* This example illustrates the usage of the ConfigAdd function.
*
* Remember that the file must be loaded using the ConfigFileLoad function.
* This file must also be saved as a backup copy of the original file using
* the ConfigFileSave function.
*
* In the first call to ConfigAdd, a path statement is added to the config
* file.
*
* In the second call to ConfigAdd, an EXENV statement is added to the loaded
* config file.
*
* The third call to ConfigAdd adds a command statement in the loaded config
* file.
*
* If any of the calls to ConfigAdd fail, the setup will exit.
*
* NOTE: In order for this script to run properly, it is advised to set the
*       preprocessor constants to valid .SYS files on the target system.
*
\*-----*/

#define EXAMPLE_SYS "EXAMPLE\\EXAMPLE.SYS"
#define EXAMSYS_BAK "EXAMPLE.BAK"

    STRING szKey, szValue, szRefKey, szMsg;
    NUMBER nOptions;

program

    // Load the target config file into memory.
    // A null string ("") would load the bootup CONFIG.SYS file by default.
    ConfigFileLoad (EXAMPLE_SYS);

    szKey    = "DEVICE";
    szValue  = "C:\\EXAPP\\EXAPP2.SYS";
    szRefKey = "EXAPP";
    nOptions = BEFORE;

/*-----*\
*
* The following adds the line DEVICE=C:\EXAPP\EXAPP2.SYS before the DEVICE
* statement in the config file, even if a duplicate line exists.
*
\*-----*/
    if (ConfigAdd (szKey, szValue, szRefKey, nOptions) < 0) then

        MessageBox ("First call to ConfigAdd failed", WARNING);
        abort;
    endif;

    szKey    = "DEVICEHIGH";
    szValue  = "C:\\OTHERAPP\\OTHERAPP";
    szRefKey = "OTHERAPP";
    nOptions = REPLACE;

/*-----*\
*
* The following adds the line DEVICEHIGH=C:\OTHERAPP\OTHERAPP. If the
```

```
* environment variable DEVICEHIGH already exists in the config file, the last
* DEVICEHIGH statement is replaced.
*
\*-----*/
  if (ConfigAdd (szKey, szValue, szRefKey, nOptions) < 0) then
    MessageBox ("Second call to ConfigAdd failed", WARNING);
    abort;
  endif;

  // Save the original backup in the same directory.
  ConfigFileSave (EXAMSYS_BAK);

  szMsg = "Successful.\n\nThe target config file has been altered. The " +
    "original was saved as " + EXAMSYS_BAK + ".";
  sprintfBox (INFORMATION, "ConfigAdd", szMsg);

endprogram

// Source file: Is5fn027.rul
```

{button ,PI("",`comment')} Feedback



ConfigDelete

Syntax

```
ConfigDelete (szKey);
```

Description

The ConfigDelete function removes lines from the system configuration file that has been loaded into memory by a call to [ConfigFileLoad](#). The parameter szKey specifies a reference key that identifies the lines to be deleted. After using advanced configuration functions to edit a system configuration file, you must call [ConfigFileSave](#) to save it your changes.



Do not mix the Ez configuration file functions with the advanced configuration file functions. After calling the ConfigFileLoad function, you cannot use the Ez configuration file functions until you use the ConfigFileSave function to save your changes.

Parameters

szKey

Specifies the reference key that identifies the line or lines to delete. Common reference keys include Himem.sys, FILES, and STACKS.

Return values

0

ConfigDelete successfully deleted the line(s) containing the reference key from the system configuration file.

< 0

ConfigDelete was unable to delete the specified line(s).

{button ,JI('LANGREF.HLP>Examples',`ConfigDelete_example`)} [Example](#)

{button ,AL(`ConfigAdd;ConfigFileLoad;ConfigFileSave;ConfigFind;ConfigGetFileName;ConfigGetInt;ConfigMove;ConfigSetFileName;ConfigSetInt',0,'`,`')} [See also](#)

{button ,PI('`,`comment')} [Feedback](#)



ConfigDelete example

```
/*-----*\
 *
 * This example illustrates the use of the ConfigDelete
 *
 * The first call to ConfigFileLoad loads a config file to be edited. The
 * original file is then saved as a backup file using ConfigFileSave.
 *
 * The ConfigDelete function is called to remove all lines containing
 * the reference key specified in the parameter passed to it.
 * NOTE: In order for this script to run properly, it is advised to set the
 *       preprocessor constants to valid filenames on the target system.
 *
 *-----*/

#define TARGET_CONFIG "C:\\EXAMPLE\\EXAMPLE.SYS"

#define BACKUP_CONFIG "EXAMPLE.BAK"

    STRING szMsg;

program

/*-----*\
 *
 * Load the target config file to be edited.
 * A null string ("") here would load the bootup CONFIG.SYS file by default.
 *
 *-----*/

    if (ConfigFileLoad(TARGET_CONFIG) < 0) then

        MessageBox("First call to ConfigFileLoad failed.", SEVERE);
    else

        MessageBox("First call to ConfigFileLoad Successful.", INFORMATION);
    endif;

/*-----*\
 *
 * Remove the lines in the file that contains "FILES" as the key
 *
 *-----*/

    if(ConfigDelete("FILES") < 0) then

        MessageBox("ConfigDelete failed",SEVERE);
    else

        MessageBox("ConfigDelete was successful",INFORMATION);
    endif;

/*-----*\
 *
 * Save the original config file with a .BAK extension.
 *
 *-----*/

    if (ConfigFileSave(BACKUP_CONFIG) < 0) then

        MessageBox("First call to ConfigFileSave failed.", SEVERE);
    else
```

```
        sprintfBox(INFORMATION, "ConfigFileSave", "First call to " +
            "ConfigFileSave successful,\n\nThe target config file has " +
            "been altered. The original was saved as " + BACKUP_CONFIG +
            ".");
    endif;

endprogram

// Source file: Is5fn608.rul
```

{button ,PI("",`comment')} [Feedback](#)



ConfigFileLoad

Syntax

ConfigFileLoad (szConfigFile);

Description

The ConfigFileLoad function loads a copy of the specified system configuration file into memory so that other advanced configuration file functions can be called to operate on the file. Specify the name of the system configuration file you want to edit in szConfigFile or pass a null string ("") in szConfigFile to edit the default system configuration file, which is set initially by InstallShield to the bootup Config.sys file used by the system.

To obtain the fully qualified name of the default system configuration file, call ConfigGetFileName. To make another file the default system configuration file, call ConfigSetFileName.

Before using any of the advanced configuration file functions, you must first call ConfigFileLoad to load the system configuration file into memory. After you modify the file, call ConfigFileSave to save it to disk.



Do not mix the Ez configuration file functions with the advanced configuration file functions. After calling the ConfigFileLoad function, you cannot use the Ez configuration file functions until you use the ConfigFileSave function to save your changes.

Parameters

szConfigFile

Specifies the fully qualified name of the system configuration file to load into memory. To load the default system configuration file, pass a null string ("") in this parameter.

Return values

0

ConfigFileLoad successfully loaded the specified system configuration file into memory.

< 0

ConfigFileLoad was unable to load the specified system configuration into memory.

{button ,JI('LANGREF.HLP>Examples','ConfigFileLoad_example')} Example

{button ,AL(' ConfigAdd;ConfigDelete;ConfigFileSave;ConfigFind;ConfigGetFileName;ConfigGetInt;ConfigMove;ConfigSetFileName;ConfigSetInt',0,'')} See also

{button ,PI('','comment')} Feedback



ConfigFileLoad example

```
/*-----*\
 *
 * This example illustrates the use of the ConfigFileLoad and the
 * ConfigFileSave function.
 *
 * The first call to ConfigFileLoad loads a config file to be edited. The
 * original file is then saved as a backup file using ConfigFileSave.
 *
 * The second call to ConfigFileLoad loads a different config file into
 * memory. The original file is then saved as a backup copy. This backup
 * copy will have a numbered extension, preventing replacing of files.
 *
 * NOTE: In order for this script to run properly, the preprocessor constants
 *       should be set to directories and filenames that exist on the target
 *       system.
 *
 *-----*/

#define TARGET_CONFIG "EXAMPLE\\EXAMPLE.SYS"
#define TARGET_CONFIG2 "EXAMPLE\\EXAMPLE2.SYS"
#define BACKUP_CONFIG "EXAMPLE.BAK"
#define BACKUP_CONFIG2 "EXAMPLE2.*"

    STRING szMsg;

program

/*-----*\
 *
 * Load the target config file to be edited.
 * A null string ("") here would load the bootup AUTOEXEC.SYS file by default.
 *
 *-----*/
    if (ConfigFileLoad(TARGET_CONFIG) < 0) then

        MessageBox("First call to ConfigFileLoad failed.", SEVERE);
    else

        MessageBox("First call to ConfigFileLoad successful.", INFORMATION);
    endif;

    // Make changes to the config file here.

/*-----*\
 *
 * Save the original config file with a .BAK extension.
 *
 *-----*/
    if (ConfigFileSave(BACKUP_CONFIG) < 0) then

        MessageBox("First call to ConfigFileSave failed.", SEVERE);
    else

        sprintfBox(INFORMATION, "ConfigFileSave", "First call to " +
            "ConfigFileSave successful,\n\nThe target config file has " +
            "been altered. The original was saved as " + BACKUP_CONFIG +
            ".");
    endif;
```

```

/*-----*\
*
* Load a different file to be edited.
*
\*-----*/
if (ConfigFileLoad(TARGET_CONFIG2) < 0) then

    MessageBox("Second call to ConfigFileLoad failed.", SEVERE);
else

    MessageBox("Second call to ConfigFileLoad Successful.", INFORMATION);
endif;

// Make changes to the config file here...

/*-----*\
*
* Save the original config file with a numbered extension.
*
\*-----*/
if (ConfigFileSave(BACKUP_CONFIG2) < 0) then

    MessageBox("Second call to ConfigFileSave failed.", SEVERE);
else

    sprintfBox(INFORMATION, "ConfigFileSave", "Second call to " +
                "ConfigFileSave successful.\n\nThe specified config file " +
                "has been altered. The original was saved as a numbered " +
                "backup copy.");
endif;

endprogram

// Source file: Is5fn029.rul

```

{button ,PI("",`comment')} Feedback



ConfigFileSave

Syntax

ConfigFileSave (szBackupFile);

Description

The ConfigFileSave function saves to disk a system configuration file that has been loaded into memory with the function [ConfigFileLoad](#). The file is saved under its original name. If a filename is specified in szBackupFile, the original file is renamed with that filename before the edited file is written to disk. If szBackupFile contains a null string (""), the original file is replaced with the modified file. If you do not call ConfigFileSave when you are finished modifying a system configuration file with [advanced configuration file functions](#), all modifications will be lost.



Do not mix the Ez configuration file functions with the advanced configuration file functions. After calling the ConfigFileLoad function, you cannot use the Ez configuration file functions until you use the ConfigFileSave function to save your changes.

Parameters

szBackupFile

Specifies the backup filename to use to back up the original file that was loaded into memory. This parameter must contain only the filename—do not include path information here. ConfigFileSave saves the original (unchanged) file with the backup filename specified in the parameter szBackupFile and also saves the new (changed) file using the original filename.

If you pass a null string ("") in this parameter, no backup file is created. If the parameter szBackupFile contains the same name as the name of the original batch file, a backup file is not created.

If you use the format "filename.*" in szBackupFile, InstallShield assigns a numeric value, starting at 001, as the extension. If a file already exists with this extension, the extension's value is increased by one until a unique filename is created.

Once a backup is made, InstallShield stores the backup filename in the system variable INFOFILENAME.

Return values

0

ConfigFileSave successfully wrote the file from memory to disk.

< 0

ConfigFileSave was unable to write the file to disk.

{button ,JI('LANGREF.HLP>Examples','ConfigFileSave_example')} [Example](#)

{button ,AL('ConfigAdd;ConfigDelete;ConfigFileLoad;ConfigFind;ConfigGetFileName;ConfigGetInt;ConfigMove;ConfigSetFileName;ConfigSetInt',0,'')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



ConfigFileSave example

```
/*-----*\
 *
 * This example illustrates the use of the ConfigFileLoad and the
 * ConfigFileSave function.
 *
 * The first call to ConfigFileLoad loads a config file to be edited. The
 * original file is then saved as a backup file using ConfigFileSave.
 *
 * The second call to ConfigFileLoad loads a different config file into
 * memory. The original file is then saved as a backup copy. This backup
 * copy will have a numbered extension, preventing replacing of files.
 *
 * NOTE: In order for this script to run properly, the preprocessor constants
 *       should be set to directories and filenames that exist on the target
 *       system.
 *
 *-----*/

#define TARGET_CONFIG "EXAMPLE\\EXAMPLE.SYS"
#define TARGET_CONFIG2 "EXAMPLE\\EXAMPLE2.SYS"
#define BACKUP_CONFIG "EXAMPLE.BAK"
#define BACKUP_CONFIG2 "EXAMPLE2.*"

    STRING szMsg;

program

/*-----*\
 *
 * Load the target config file to be edited.
 * A null string ("") here would load the bootup AUTOEXEC.SYS file by default.
 *
 *-----*/
    if (ConfigFileLoad(TARGET_CONFIG) < 0) then

        MessageBox("First call to ConfigFileLoad failed.", SEVERE);
    else

        MessageBox("First call to ConfigFileLoad successful.", INFORMATION);
    endif;

    // Make changes to the config file here.

/*-----*\
 *
 * Save the original config file with a .BAK extension.
 *
 *-----*/
    if (ConfigFileSave(BACKUP_CONFIG) < 0) then

        MessageBox("First call to ConfigFileSave failed.", SEVERE);
    else

        sprintfBox(INFORMATION, "ConfigFileSave", "First call to " +
            "ConfigFileSave successful,\n\nThe target config file has " +
            "been altered. The original was saved as " + BACKUP_CONFIG +
            ".");
    endif;
```

```

/*-----*\
*
* Load a different file to be edited.
*
\*-----*/
if (ConfigFileLoad(TARGET_CONFIG2) < 0) then

    MessageBox("Second call to ConfigFileLoad failed.", SEVERE);
else

    MessageBox("Second call to ConfigFileLoad Successful.", INFORMATION);
endif;

// Make changes to the config file here...

/*-----*\
*
* Save the original config file with a numbered extension.
*
\*-----*/
if (ConfigFileSave(BACKUP_CONFIG2) < 0) then

    MessageBox("Second call to ConfigFileSave failed.", SEVERE);
else

    sprintfBox(INFORMATION, "ConfigFileSave", "Second call to " +
                "ConfigFileSave successful.\n\nThe specified config file " +
                "has been altered. The original was saved as a numbered " +
                "backup copy.");
endif;

endprogram

// Source file: Is5fn029.rul

```

{button ,PI("",`comment')} Feedback



ConfigFind

Syntax

ConfigFind (szRefKey, svResult, nOptions);

Description

The ConfigFind function searches a system configuration file that has been loaded into memory with the function [ConfigFileLoad](#). The parameter szRefKey is a reference key that specifies the search target in that file. If the reference key is found, its value is returned in svResult. To find all occurrences of szRefKey, call this function repeatedly with nOptions set to CONTINUE. To restart the search from the top of the file, specify the constant RESTART in nOptions. After you edit the file, call [ConfigFileSave](#) to save it.



Do not mix the Ez configuration file functions with the advanced configuration file functions. After calling the ConfigFileLoad function, you cannot use the Ez configuration file functions until you use the ConfigFileSave function to save your changes.

Parameters

szRefKey

Specifies the reference key to search for. You must use only acceptable reference keys. If the reference key is a filename and you do not specify a file extension, all file extensions are included in the search. For example, if you enter Win.com, the search looks only for this reference key. If you enter WIN, the files Win.exe, Win.dll, Win.sys, etc., are all returned.

svResult

Returns the value of the reference key that was found in the system configuration file.

nOptions

Indicates whether to start the search from the beginning of the file or to continue from where the previous search was terminated. Pass one of the following predefined constants in this parameter:

RESTART

Starts the search from the beginning of the file.

CONTINUE

Starts the search from the current position in the system configuration file.

COMMAND

Indicates that the reference key in szRefKey is a command, not an environment variable. The constant COMMAND can be joined with the constant RESTART or the constant CONTINUE by using the bitwise OR operator (|), as in the following example:

```
ConfigFind("Vga.drv", svResult, CONTINUE | COMMAND);
```

Return values

0

ConfigFind successfully found the specified reference key and returned it in svResult.

< 0

ConfigFind was unable to find the specified reference key.

Comments

A system configuration file can contain both environment variables and commands. To distinguish between environment variables and commands with the same name, use the constant COMMAND to specify that you are looking for executable commands.

{button ,JI('LANGREF.HLP>Examples','ConfigFind_example')} Example

{button ,AL(' ConfigAdd;ConfigDelete;ConfigFileLoad;ConfigFileSave;ConfigGetFileName;ConfigGetInt;ConfigMove;
ConfigSetFileName;ConfigSetInt',0,'')} See also

{button ,PI('','comment')} Feedback



ConfigFind example

```
/*-----*\
*
* This example illustrates the ConfigFind function.
*
* It is necessary to load the config file using the ConfigFileLoad function
* and save it when finished making modifications using the ConfigFileSave
* function.
*
* The first call to ConfigFind checks to see if HIMEM.SYS exists. A message
* informs the user if the HIMEM.SYS reference key is found or not.
*
* The second call to ConfigFind searches for the reference key EXAPP. Each
* time it finds an entry for the EXAPP reference key, it displays the entry in
* a message box.
*
* NOTE: In order for this script to function properly, the TARGET_CONFIG
* constant must be set to a file that exists on the target system.
*
\*-----*/

#define TARGET_CONFIG "EXAMPLE\\EXAMPLE.SYS"

    STRING szRefKey, svResult, szTitle;
    NUMBER nOptions, nResult;

program

    // Load the target config file to be edited.
    // A null string ("") would load the bootup CONFIG.SYS file by default.
    ConfigFileLoad(TARGET_CONFIG);

    szRefKey = "HIMEM.SYS";
    nOptions = COMMAND;
    szTitle = "ConfigFind Example";

/*-----*\
*
* Check to see if the HIMEM.SYS reference key exists. COMMAND indicates the
* key to search for is a command statement.
*
\*-----*/
    nResult = ConfigFind(szRefKey, svResult, nOptions);

    if (nResult < 0) then

        sprintfBox(SEVERE, szTitle, "%s not found.", svResult);
    else

        sprintfBox(INFORMATION, szTitle, "%s found.", svResult);
    endif;

    szRefKey = "EXAPP";
    nOptions = RESTART;

/*-----*\
*
* Display every EXAPP reference key entry. RESTART indicates to search from
* the beginning of the file.
*
\*-----*/
```

```
nResult = ConfigFind(szRefKey, svResult, nOptions);

while (nResult = 0)

    MessageBox(svResult, INFORMATION);

    nOptions = CONTINUE;
/*-----*\
*
* Continue searching for every reference key entry. CONTINUE indicates to
* search from the current location in the file.
*
*\-----*/
    nResult = ConfigFind(szRefKey, svResult, nOptions);
endwhile;

endprogram

// Source file: Is5fn031.rul
```

{button ,PI("",`comment')} [Feedback](#)



ConfigGetFileName

Syntax

```
ConfigGetFileName (svFileName);
```

Description

The ConfigGetFileName function retrieves the fully qualified name of the default system configuration file, which is set initially by InstallShield to the Config.sys file that was executed when the target system was started. To obtain the fully qualified name of the default system configuration file, call [ConfigGetFileName](#). To make another file the default system configuration file, call [ConfigSetFileName](#).



Do not mix the Ez configuration file functions with the advanced configuration file functions. After calling the ConfigFileLoad function, you cannot use the Ez configuration file functions until you use the ConfigFileSave function to save your changes.

Parameters

svFileName

Returns the fully qualified name of the default system configuration file.

Return values

0

ConfigGetFileName successfully retrieved the fully qualified name of the default system configuration file.

< 0

ConfigGetFileName was unable to retrieve the fully qualified name of the default system configuration file.

Comments

Under rare circumstances InstallShield may not be able to determine the fully qualified name of the default configuration file. In that case, svFileName will be a null string ("").

{button ,JI('LANGREF.HLP>Examples','ConfigGetFileName_example')} [Example](#)

{button ,AL('ConfigAdd;ConfigDelete;ConfigFileLoad;ConfigFileSave;ConfigFind;ConfigGetInt;ConfigMove;ConfigSetFileName;ConfigSetInt',0,',')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



ConfigGetFileName example

```
/*-----*\
 *
 * This example illustrates the usage of the ConfigGetFileName and
 * ConfigSetFileName functions.
 *
 * ConfigGetFileName is first called to retrieve the default configuration
 * file. ConfigSetFileName is then called to set the default configuration
 * file to EXAMPLE_SYS. ConfigGetFileName is called a second time to retrieve
 * the new default configuration.
 *
\*-----*/

#define EXAMPLE_SYS "EXAMPLE\\EXAMPLE.SYS"

    STRING szTitle, szMsg, svFileName, szConfigFile;

program

    szTitle = "ConfigGetFileName & ConfigSetFileName";

/*-----*\
 *
 * ConfigGetFileName retrieves the current default configuration file.
 *
\*-----*/
    if (ConfigGetFileName(svFileName) < 0) then

        MessageBox("ConfigGetFileName failed.", SEVERE);
    else

        szMsg = "The default configuration file is %s.";
        sprintfBox(INFORMATION, szTitle, szMsg, svFileName);
    endif;

    szConfigFile = EXAMPLE_SYS;

/*-----*\
 *
 * ConfigSetFileName sets a new default configuration file.
 *
\*-----*/
    if (ConfigSetFileName(szConfigFile) < 0) then

        MessageBox("ConfigSetFileName failed.", SEVERE);
    else

        MessageBox("The default config file has changed.", INFORMATION);
    endif;

    if (ConfigGetFileName(svFileName) < 0) then

        MessageBox("ConfigGetFileName failed.", SEVERE);
    else

        szMsg = "The default configuration file now is %s.";
        sprintfBox(INFORMATION, szTitle, szMsg, svFileName);
    endif;

endprogram
```

// Source file: Is5fn032.rul

{button ,PI("",`comment')} Feedback



ConfigGetInt

Syntax

ConfigGetInt (szKey, nvValue);

Description

The ConfigGetInt function retrieves the integer value of a reference key from a system configuration file that has been loaded into memory with the function [ConfigFileLoad](#). ConfigGetInt retrieves values from commands that have only one value to the right of the equal sign (=). ConfigGetInt does not work on a command that has more than one value. For example, ConfigGetInt recognizes the statement FILES=20 and returns the number 20, but it does not recognize the statement STACKS=9,128.

Before to calling ConfigGetInt, you must first call ConfigFileLoad to load the system configuration file into memory. After you edit the file, call [ConfigFileSave](#) to save the file.



Do not mix the Ez configuration file functions with the advanced configuration file functions. After calling the ConfigFileLoad function, you cannot use the Ez configuration file functions until you use the ConfigFileSave function to save your changes.

Parameters

szKey

Specifies the reference key of the statement from which to retrieve the integer value.

nvValue

Returns the integer value of the reference key.

Return values

0

ConfigGetInt successfully retrieved the integer value.

< 0

ConfigGetInt was unable to retrieve the integer value.

{button ,JI(`LANGREF.HLP>Examples`,`ConfigGetInt_example`)} [Example](#)

{button ,AL(`ConfigAdd;ConfigDelete;ConfigFileLoad;ConfigFileSave;ConfigFind;ConfigGetFileName;ConfigMove;ConfigSetFileName;ConfigSetInt`,0,`,`)} [See also](#)

{button ,PI(``,`comment`)} [Feedback](#)



ConfigGetInt example

```
/*-----*\
 *
 * This example illustrates the usage of the ConfigGetInt and ConfigSetInt
 * functions.
 *
 * ConfigGetInt is called to see if a value is less than 40.  If the value is
 * less than 40, ConfigSetInt changes the value to 40.
 *
 * NOTE: In order for this script to run properly, it is advised to set the
 *       EXAMPLE_SYS constant to a valid .SYS file on the target system.
 *
\*-----*/

#define EXAMPLE_SYS "EXAMPLE\\EXAMPLE.SYS"

    STRING szKey;
    NUMBER nvValue;

program

    // Load the EXAMPLE.SYS file, A Null ("") string here would load the
    // default CONFIG.SYS file.
    ConfigFileLoad(EXAMPLE_SYS);

    szKey = "FILES";
/*-----*\
 *
 * ConfigGetInt is called to test the value of the "FILES=" command in the
 * configuration file.
 *
\*-----*/
    if (ConfigGetInt(szKey, nvValue) < 0) then

        MessageBox("ConfigGetInt failed.", SEVERE);
    else

        sprintfBox(INFORMATION, "ConfigGetInt Example", "FILES equaled: %d",
            nvValue);
    endif;

    // If FILES was set to a value less than 40, set it to 40.
    if (nvValue < 40) then

        nvValue = 40;
/*-----*\
 *
 * ConfigSetInt is called to set the value of "FILES=" to 40.
 *
\*-----*/
    if (ConfigSetInt(szKey, nvValue) < 0) then

        MessageBox("ConfigSetInt failed", SEVERE);
    else

        MessageBox("ConfigSetInt successful.", INFORMATION);
    endif;
endif;

    // Save the changes and save the original EXAMPLE.SYS as EXAMPLE.BAK.
```

```
    ConfigFileSave ("EXAMPLE.BAK");  
endprogram  
// Source file: Is5fn033.rul
```

{button ,PI("",`comment')} Feedback



ConfigMove

Syntax

ConfigMove (szMove, szRefKey, nOptions);

Description

The ConfigMove function moves a line in a system configuration file that has been loaded into memory with the function [ConfigFileLoad](#). The line can be moved to the first or last position in the file or before or after a specific line in the file.

Before calling the ConfigMove function, you must first call ConfigFileLoad to load the Config.sys file into memory. After you edit the file, call [ConfigFileSave](#) to save the file.



Do not mix the Ez configuration file functions with the advanced configuration file functions. After calling the ConfigFileLoad function, you cannot use the Ez configuration file functions until you use the ConfigFileSave function to save your changes.

Parameters

szMove

Specifies the line to move.

szRefKey

Specifies the key that identifies the reference line used to position the line to be moved. The position of the line to be moved is determined by the value of nOptions.

nOptions

Specifies whether you are moving the line in szMove before or after the line that contains the reference key in szRefKey. Pass one of the following predefined constants in this parameter:

BEFORE

The line specified by szMove is placed before the line containing szRefKey. If szMove is a null string (""), the line is placed before the first line in the system configuration file.

AFTER

The line specified by szMove is placed after the line containing szRefKey. If szMove is a null string (""), the line is placed after the last line in the system configuration file.

Return values

0

ConfigMove successfully moved the specified line in the system configuration file.

< 0

ConfigMove was unable to move the line.

{button ,JI('LANGREF.HLP>Examples',`ConfigMove_example`)} [Example](#)

{button ,AL(` ConfigAdd;ConfigFileLoad;ConfigFileSave;ConfigFind;ConfigGetFileName;ConfigGetInt;ConfigMove;ConfigSetFileName;ConfigSetInt',0,','`)} [See also](#)

{button ,PI('','comment')} [Feedback](#)



ConfigMove example

```
/*-----*\
 *
 * This example illustrates the use of the ConfigMove function.
 *
 * It is necessary to load the file using the ConfigFileLoad function and save
 * the file when finished making modifications using the ConfigFileSave
 * function.
 *
 * The first call to ConfigMove moves the SYSTEM statement to the last line
 * in the file.
 *
 * The second call to ConfigMove moves the HIMEM command before the HIDOS
 * statement.
 *
 * NOTE: In order for this script to run properly, the TARGET_CONFIG and
 *       BACKUP_CONFIG constants must be set to a directory and configuration
 *       file that exists on the target system.
 *
\*-----*/

#define TARGET_CONFIG "C:\\EXAMPLE\\EXAMPLE.SYS"
#define BACKUP_CONFIG "EXAMPLE.BAK"

    STRING szMove, szRefKey;
    NUMBER nOptions;

program

    // Load the target config file to be edited.
    // A null string ("") would load the bootup CONFIG.SYS file by default.
    ConfigFileLoad(TARGET_CONFIG);

    szMove   = "DMDVR.BIN";
    szRefKey = "";
    nOptions = AFTER;
/*-----*\
 *
 * The following moves the SYSTEM statement to the end of the file.
 *
\*-----*/
    if (ConfigMove(szMove, szRefKey, nOptions) < 0) then

        MessageBox("First call to ConfigMove failed.", SEVERE);
    else

        MessageBox("First call to ConfigMove successful.", INFORMATION);
    endif;

    szMove   = "HIMEM.SYS";
    szRefKey = "HIDOS.SYS";
    nOptions = BEFORE;
/*-----*\
 *
 * The following moves the HIMEM statement before the HIDOS statement.
 *
\*-----*/
    if (ConfigMove(szMove, szRefKey, nOptions) < 0) then

        MessageBox("Second call to ConfigMove failed.", SEVERE);
```

```
else

    MessageBox("Second call to ConfigMove successful.", INFORMATION);
endif;

ConfigFileSave (BACKUP_CONFIG);

endprogram

// Source file: Is5fn034.rul
```

{button ,PI("",`comment')} Feedback



ConfigSetFileName

Syntax

ConfigSetFileName (szConfigFile);

Description

The ConfigSetFileName function specifies the fully qualified name of the file you want to use as the default system configuration file. During setup initialization, InstallShield identifies the Config.sys file that was executed when the target system was started and makes it the default system configuration file. If this is the only system configuration file your setup will edit, it's unnecessary to call this function. Ez configuration files will use that file and the advanced configuration function ConfigFileLoad will open that file when its parameter is a null string ("").

However, if you want to use Ez configuration file functions to modify a configuration file other than the bootup Config.sys file, you must call ConfigSetFileName to change the default system configuration file. For example, suppose you wanted to create a Config.sys file on the target system that would not be used at bootup time. You could set a file name in the application directory. Ez configuration file functions would then operate on that file; and if you called ConfigFileLoad with a null parameter, that file would be loaded into memory, where it could be edited with advanced file functions.



Do not mix the Ez configuration file functions with the advanced configuration file functions. After calling the ConfigFileLoad function, you cannot use the Ez configuration file functions until you use the ConfigFileSave function to save your changes.

Parameters

szConfigFile

Specifies the fully qualified name of the file to set as the default system configuration file.

Return values

0

ConfigSetFileName successfully retrieved the specified system configuration file.

< 0

ConfigSetFileName was unable to retrieve the specified file.

Comments

- n The ConfigSetFileName function does not load a system configuration file into memory. You must use [ConfigFileLoad](#) to load a file into memory.
- n ConfigSetFileName does not validate the filename you specify. If you specify an invalid filename, all future configuration file functions will fail.

{button ,JI('LANGREF.HLP>Examples','ConfigSetFileName_example')} [Example](#)

{button ,AL('ConfigAdd;ConfigDelete;ConfigFileLoad;ConfigFileSave;ConfigFind;ConfigGetFileName;ConfigGetInt;ConfigMove;ConfigSetInt',0,','')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



ConfigSetFileName example

```
/*-----*\
 *
 * This example illustrates the usage of the ConfigGetFileName and
 * ConfigSetFileName functions.
 *
 * ConfigGetFileName is first called to retrieve the default configuration
 * file. ConfigSetFileName is then called to set the default configuration
 * file to EXAMPLE_SYS. ConfigGetFileName is called a second time to retrieve
 * the new default configuration.
 *
\*-----*/

#define EXAMPLE_SYS "EXAMPLE\\EXAMPLE.SYS"

    STRING szTitle, szMsg, svFileName, szConfigFile;

program

    szTitle = "ConfigGetFileName & ConfigSetFileName";

/*-----*\
 *
 * ConfigGetFileName retrieves the current default configuration file.
 *
\*-----*/
    if (ConfigGetFileName(svFileName) < 0) then

        MessageBox("ConfigGetFileName failed.", SEVERE);
    else

        szMsg = "The default configuration file is %s.";
        sprintfBox(INFORMATION, szTitle, szMsg, svFileName);
    endif;

    szConfigFile = EXAMPLE_SYS;

/*-----*\
 *
 * ConfigSetFileName sets a new default configuration file.
 *
\*-----*/
    if (ConfigSetFileName(szConfigFile) < 0) then

        MessageBox("ConfigSetFileName failed.", SEVERE);
    else

        MessageBox("The default config file has changed.", INFORMATION);
    endif;

    if (ConfigGetFileName(svFileName) < 0) then

        MessageBox("ConfigGetFileName failed.", SEVERE);
    else

        szMsg = "The default configuration file now is %s.";
        sprintfBox(INFORMATION, szTitle, szMsg, svFileName);
    endif;

endprogram
```

// Source file: Is5fn032.rul

{button ,PI("",`comment')} Feedback



ConfigSetInt

Syntax

ConfigSetInt (szKey, nValue);

Description

The ConfigSetInt function changes a specified integer value in a system configuration file that has been loaded into memory with the function [ConfigFileLoad](#). ConfigSetInt sets values in commands that have only one value to the right of the equal sign (=); it does not work on a command that has more than one value. For example, ConfigSetInt recognizes the statement FILES=20 and can change 20 to another value, but it does not recognize the statement STACKS=9,128.

Before calling ConfigSetInt, you must first call ConfigFileLoad to load the system configuration file into memory. After you edit the file, call [ConfigFileSave](#) to save the file.



Do not mix the Ez configuration file functions with the advanced configuration file functions. After calling the ConfigFileLoad function, you cannot use the Ez configuration file functions until you use the ConfigFileSave function to save your changes.

Parameters

szKey

Specifies the reference keyword of the command whose integer value is to be set.

nValue

Specifies the integer value you want to set for the command in szKey.

Return values

0

ConfigSetInt successfully set the specified integer in the system configuration file .

< 0

ConfigSetInt was unable to set the specified integer.

{button ,JI(`LANGREF.HLP>Examples`,`ConfigSetInt_example`)} [Example](#)

{button ,AL(`ConfigAdd;ConfigDelete;ConfigFileLoad;ConfigFileSave;ConfigFind;ConfigGetFileName;ConfigGetInt;ConfigMove;ConfigSetFileName`,0,`,`)} [See also](#)

{button ,PI(``,`comment`)} [Feedback](#)



ConfigSetInt example

```
/*-----*\
 *
 * This example illustrates the usage of the ConfigGetInt and ConfigSetInt
 * functions.
 *
 * ConfigGetInt is called to see if a value is less than 40.  If the value is
 * less than 40, ConfigSetInt changes the value to 40.
 *
 * NOTE: In order for this script to run properly, it is advised to set the
 *       EXAMPLE_SYS constant to a valid .SYS file on the target system.
 *
\*-----*/

#define EXAMPLE_SYS "EXAMPLE\\EXAMPLE.SYS"

    STRING szKey;
    NUMBER nvValue;

program

    // Load the EXAMPLE.SYS file, A Null ("") string here would load the
    // default CONFIG.SYS file.
    ConfigFileLoad(EXAMPLE_SYS);

    szKey = "FILES";
/*-----*\
 *
 * ConfigGetInt is called to test the value of the "FILES=" command in the
 * configuration file.
 *
\*-----*/
    if (ConfigGetInt(szKey, nvValue) < 0) then

        MessageBox("ConfigGetInt failed.", SEVERE);
    else

        sprintfBox(INFORMATION, "ConfigGetInt Example", "FILES equaled: %d",
            nvValue);
    endif;

    // If FILES was set to a value less than 40, set it to 40.
    if (nvValue < 40) then

        nvValue = 40;
/*-----*\
 *
 * ConfigSetInt is called to set the value of "FILES=" to 40.
 *
\*-----*/
    if (ConfigSetInt(szKey, nvValue) < 0) then

        MessageBox("ConfigSetInt failed", SEVERE);
    else

        MessageBox("ConfigSetInt successful.", INFORMATION);
    endif;
endif;

    // Save the changes and save the original EXAMPLE.SYS as EXAMPLE.BAK.
```

```
    ConfigFileSave ("EXAMPLE.BAK");  
endprogram  
// Source file: Is5fn033.rul
```

{button ,PI("",`comment')} Feedback



Custom dialog box functions

The functions below handle custom dialog box processing. You create the dialog box using a resource editor and you merge it in the script using these functions.

Any Windows dialog box you can create can be used in a setup script. The dialogs can have single and multi-line edit boxes, single and multi-selection list boxes, combo boxes, radio buttons, check boxes, and push buttons as standard controls. For more complex controls, advanced functions such as `CmdGetHwndDlg`, `LOWORD`, and `HIWORD` are provided.

CmdGetHwndDlg

Retrieves the handle of a dialog.

CtrlClear

Deletes the contents of an edit, static, list box, or combo box control.

CtrlDir

Fills a list box or combo box with either a directory listing or a file listing.

CtrlGetCurSel

Returns the selected item from a list box or combo box.

CtrlGetMLEText

Retrieves the text from a multi-line edit or static field.

CtrlGetMultCurSel

Returns the selected items from a multi-selection list box.

CtrlGetState

Retrieves the state of a radio button, check box, or push button control from a dialog box.

CtrlGetSubCommand

Retrieves the operation performed on the control after a `WaitOnDialog` function call.

CtrlGetText

Retrieves the text from an edit field, a static field, or the edit field of a combo box.

CtrlPGroups

Retrieves a list of program group names that exist on the target system.

CtrlSelectText

Selects the text displayed in an edit field.

CtrlSetCurSel

Finds and sets the current selection in a list box or combo box.

CtrlSetFont

Specifies a font for a control in the dialog box.

CtrlSetList

Places the contents of a list into a list box or combo box.

CtrlSetMLEText

Sets the text in a multi-line edit field.

CtrlSetMultCurSel

Sets the current selection in a multi-selection list box.

CtrlSetState

Sets the current state of a check box, radio button, or push button control.

CtrlSetText

Sets the text in an edit field, a static text field, or the edit field of a combo box.

DefineDialog

Registers a custom dialog box with `InstallShield`.

EndDialog

Closes a custom dialog box.

EzDefineDialog

Registers a custom dialog box with `InstallShield`.

GetFont

Retrieves the handle of a font.

HIWORD

Retrieves the high-order word from a 32-bit integer.

LOWORD

Retrieves the low-order word from a 32-bit integer.

ReleaseDialog

Frees the memory associated with a dialog box.

SdMakeName

SdMakeName creates a section name for a custom dialog. This section name is used in writing to and reading from an .iss file, which is used by InstallShield Silent.

SilentReadData

Instructs InstallShield Silent to read the .iss file dialog data for a custom dialog box.

SilentWriteData

Instructs InstallShield Silent to write to the .iss file dialog data for a custom dialog box.

WaitOnDialog

Presents a custom dialog box.

{button ,PI("",`comment')} Feedback



GetFont

Syntax

GetFont (szFontName, nPointSize, nAttributes);

Description

The GetFont function builds a font and retrieves its handle. You can use the font handle to specify the font used by the controls in a custom dialog box.

Parameters

szFontName

Specifies the name of the font you want to build.

nPointSize

Specifies the point size of the font you want to build.

nAttributes

Specifies the style of the font. Pass one or more of the following predefined constants in this parameter. Specify multiple styles by combining constants with the bitwise OR operator (|).

STYLE_BOLD

Specifies a bold style font.

STYLE_ITALIC

Specifies you want the font to be italicized.

STYLE_NORMAL

Specifies a normal style font.

STYLE_UNDERLINE

Specifies you want the characters to be underlined.

Return values

XXXX

Where XXXX is the handle to the font.

0

GetFont was unable to build the requested font.

Comments

InstallShield deletes all fonts created with this function when the setup terminates. In addition, InstallShield releases all system resources upon termination.

{button ,JI('LANGREF.HLP>Examples','GetFont_example')} [Example](#)

{button ,AL('CtrlSetFont',0,'')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



GetFont example

```
/*-----*\
*
* This example illustrates the usage of the GetFont and CtrlSetFont
* functions.
*
* GetFont is called to retrieve the handle of a font. This handle is then
* passed to CtrlSetFont to set the font of a static text field in a custom
* dialog box.
*
* NOTE: The custom dialog box used in this script is actually an Sd dialog
*       box, located in the _ISRES.DLL file. Also note that it may be
*       necessary to move the dialog box in order to refresh the display of
*       the static text fields.
*
*-----*\

// Dialog box controls.
#define RES_DIALOG_ID      12006    // ID of Dialog itself.
#define RES_PBUT_NEXT     1        // ID of 'Next' push button.
#define RES_PBUT_CANCEL   9        // ID of 'Cancel' push button.
#define RES_TEXT1         720      // ID of first static text box.
#define RES_TEXT2         712      // ID of second static text box.
#define RES_TEXT3         723      // ID of third static text box.

STRING szDialog, szDLL;
NUMBER nResult, nCmdValue;
HWND   hFont1, hFont2, hFont3;
BOOL   bDone;

program

    hFont1 = GetFont("Times New Roman", 8, STYLE_BOLD);

    hFont2 = GetFont("MS Linedraw", 7, STYLE_ITALIC);

    hFont3 = GetFont("Arial", 9, STYLE_NORMAL);

    if (hFont1 = 0 || hFont2 = 0 || hFont3 = 0) then

        MessageBox("Unable to build font", SEVERE);
    endif;

    szDialog = "CtrlSetFontDialog";

    // A null "" symbol is set for szDLL to have InstallShield look in _ISUSER.DLL
    // file, then the _ISRES.DLL file.
    nResult = EzDefineDialog(szDialog, "", "", RES_DIALOG_ID);

    if(nResult < 0) then

        MessageBox("Error in defining dialog", SEVERE);
        bDone = TRUE;
    else

        bDone = FALSE;
    endif;

    while (bDone = FALSE)
```

```

// Display dialog and wait for user's input
nCmdValue = WaitOnDialog(szDialog);

switch (nCmdValue)

    case DLG_ERR:

        MessageBox("Dialog failed", SEVERE);
        bDone = TRUE;
    case DLG_INIT:

        // Set the font for static text box 1.
        if(CtrlSetFont(szDialog, hFont1, RES_TEXT1) < 0) then

            MessageBox("Unable to set font 1.", SEVERE);
            endif;

        // Set font for static text box 2.
        if(CtrlSetFont(szDialog, hFont2, RES_TEXT2) < 0) then

            MessageBox("Unable to set font 2.", SEVERE);
            endif;

        // Set font for static text box 3.
        if(CtrlSetFont(szDialog, hFont3, RES_TEXT3) < 0) then

            MessageBox("Unable to set font 3.", SEVERE);
            endif;
    case DLG_CLOSE:

        bDone = TRUE;
    case RES_PBUT_NEXT:

        bDone = TRUE;
    case RES_PBUT_CANCEL:

        bDone = TRUE;

    endswitch;
endwhile;

// Close the dialog box
EndDialog(szDialog);

// Free the dialog box from memory
ReleaseDialog(szDialog);

endprogram

// Source file: Is5fn049.rul

```

{button ,PI("",`comment')} [Feedback](#)



HIWORD

Syntax

HIWORD (IValue);

Description

The HIWORD function extracts and returns the high-order word (upper two bytes) from the 32-bit integer value specified by IValue.

Parameters

IValue

Specifies the 32-bit integer from which to extract the upper two bytes.

Return values

HIWORD returns the high-order word (upper two bytes) of IValue.

{button ,JI(^LANGREF.HLP>Examples',`HIWORD_example')} [Example](#)

{button ,AL(^LOWORD',0,`,`')} [See also](#)

{button ,PI("",`comment')} [Feedback](#)



HIWORD example

```
/*-----*\
*
* This example illustrates the usage HIWORD and LOWORD
*
*
* NOTE: In order for this script to run properly, it is advised to set the
*       FOLDER constant to a valid program group on the target system.
*
*       Also note that QueryProgGroup and therefore this script have no
*       effect under Windows 95.
*
\*-----*/

#define FOLDER "InstallShield5"

    STRING  svResult, svGroupPath, szGroupName, szTitle, szMsg;
    NUMBER  nvResult, nvItemCount, nCount, nType;
    BOOL    bWinNT;

program

    // Determine if the target system is running Windows NT.
    GetSystemInfo (OS, nvResult, svResult);

    if (nvResult = IS_WINDOWSNT) then

        bWinNT = TRUE;
    else

        bWinNT = FALSE;
    endif;

    szGroupName = FOLDER;
    szTitle      = "QueryProgGroup Example";
/*-----*\
*
* Query the FOLDER group.  nvItemCount is set to the number of items in the
* group.  Under Windows NT, the high word contains the group type.  If the
* FOLDER group is of Common type, the nType variable will be set to 1.  If
* the group is of Personal type, the nType variable will be set to 0.
*
* Under Windows 3.1, nvItemCount contains the number of program items in
* the group.  Under Windows NT, nvItemCount contains the number of program
* items and the group type.  The lower 2 bytes contain the number of items
* in the group.  The upper 2 bytes contain the group type: 0 = Personal;
* and 1 = Common.
*
* HIWORD and LOWORD functions are used to retrieve the upper and lower order
* bytes respectively.
\*-----*/
    if (QueryProgGroup (szGroupName, svGroupPath, nvItemCount) < 0) then

        MessageBox("QueryProgGroup failed.", WARNING);

    endif;

    if (bWinNT = TRUE) then

        nCount = LOWORD(nvItemCount);
```

```
nType = HIWORD(nvItemCount);

if (nType = 1) then

    szMsg = "Under WinNT:\n\nGroup %s is Common.\nNumber of " +
            "items = %ld";
else

    szMsg = "Under WinNT:\n\nGroup %s is Personal.\nNumber of " +
            "items = %ld";
endif;

// Display the results for NT machine.
sprintfBox(INFORMATION, szTitle, szMsg, szGroupName, nCount, nType);
else

// Display the results for a non-NT machine.
szMsg = "The %s group has %ld items, and is located at %s.";
sprintfBox (INFORMATION, szTitle, szMsg, szGroupName, nvItemCount,
            svGroupPath);
endif;

endprogram

// Source file: Is5fn090.rul
```

{button ,PI("",`comment')} Feedback



LOWORD

Syntax

LOWORD (IValue);

Description

The LOWORD function extracts the low-order word (two bytes) from the 32-bit integer value specified by IValue.

Parameters

IValue

Specifies the 32-bit integer from which you want to extract the lower two bytes.

Return values

This function returns the low-order word (lower two bytes) of the integer.

{button ,JI('LANGREF.HLP>Examples','LOWORD_example')} [Example](#)

{button ,AL('HIWORD',0,'')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



LOWORD example

```
/*-----*\
*
* This example illustrates the usage HIWORD and LOWORD
*
*
* NOTE: In order for this script to run properly, it is advised to set the
*       FOLDER constant to a valid program group on the target system.
*
*       Also note that QueryProgGroup and therefore this script have no
*       effect under Windows 95.
*
\*-----*/

#define FOLDER "InstallShield5"

    STRING  svResult, svGroupPath, szGroupName, szTitle, szMsg;
    NUMBER  nvResult, nvItemCount, nCount, nType;
    BOOL    bWinNT;

program

    // Determine if the target system is running Windows NT.
    GetSystemInfo (OS, nvResult, svResult);

    if (nvResult = IS_WINDOWSNT) then

        bWinNT = TRUE;
    else

        bWinNT = FALSE;
    endif;

    szGroupName = FOLDER;
    szTitle      = "QueryProgGroup Example";
/*-----*\
*
* Query the FOLDER group.  nvItemCount is set to the number of items in the
* group.  Under Windows NT, the high word contains the group type.  If the
* FOLDER group is of Common type, the nType variable will be set to 1.  If
* the group is of Personal type, the nType variable will be set to 0.
*
* Under Windows 3.1, nvItemCount contains the number of program items in
* the group.  Under Windows NT, nvItemCount contains the number of program
* items and the group type.  The lower 2 bytes contain the number of items
* in the group.  The upper 2 bytes contain the group type: 0 = Personal;
* and 1 = Common.
*
* HIWORD and LOWORD functions are used to retrieve the upper and lower order
* bytes respectively.
\*-----*/
    if (QueryProgGroup (szGroupName, svGroupPath, nvItemCount) < 0) then

        MessageBox("QueryProgGroup failed.", WARNING);

    endif;

    if (bWinNT = TRUE) then

        nCount = LOWORD(nvItemCount);
```

```
nType = HIWORD(nvItemCount);

if (nType = 1) then
    szMsg = "Under WinNT:\n\nGroup %s is Common.\nNumber of " +
           "items = %ld";
else
    szMsg = "Under WinNT:\n\nGroup %s is Personal.\nNumber of " +
           "items = %ld";
endif;

// Display the results for NT machine.
sprintfBox(INFORMATION, szTitle, szMsg, szGroupName, nCount, nType);
else

// Display the results for a non-NT machine.
szMsg = "The %s group has %ld items, and is located at %s.";
sprintfBox (INFORMATION, szTitle, szMsg, szGroupName, nvItemCount,
           svGroupPath);
endif;

endprogram

// Source file: Is5fn090.rul
```

{button ,PI("",`comment')} Feedback



Extensibility functions

Extensibility functions allow you to call functions in Dynamic-Link Libraries, call Windows APIs, or launch another application or setup script. The UseDLL and UnUseDLL functions allow you to load or unload a DLL into memory and make use of the DLL. The LaunchApp and LaunchAppAndWait functions allow you to launch another Windows or DOS application while the script is still executing.

AppCommand

Controls the Program Manager.

CallDLLFx

Calls function from an external DLL.

Delay

Delays the execution of the setup script.

LaunchApp

Launches another program.

LaunchAppAndWait

Launches another program and waits for that program to terminate.

UnUseDLL

Unloads a DLL from memory.

UseDLL

Loads a DLL into memory.

{button ,PI("",`comment`)} [Feedback](#)



CallDLLFx

Syntax

```
CallDLLFx (szDLL, szFunction, IvValue, svValue);
```

Description

The CallDLLFx function calls a function within a specified DLL. That function must use the following fixed definition, where hwnd is the main window handle for the main InstallShield window:

```
LONG WINAPI YourFunction (HWND hwnd, LPLONG lpIValue, LPSTR lpszValue);
```

Parameters

szDLL

Specifies the fully qualified filename of the DLL that contains the function to execute.

szFunction

Specifies the name of the function in the DLL specified in szDLL.

IvValue

Specifies a LONG variable to pass by reference to the DLL function.

svValue

Specifies a string variable to pass to the DLL function.

Return values

The CallDLLFx function returns a LONG value from the function in the DLL.

{button ,JI('LANGREF.HLP>Examples','CallDLLFx_example')} [Example](#)

{button ,AL('UseDLL;UnUseDLL;GetWindowHandle;Call a Windows API;Call a DLL function;Requirements for external functions',0,'')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



CallDLLFx example

```
/*-----*\
*
* This example illustrates the use of the CallDLLFx function.
*
* NOTE: This script requires that the constant DLL_FILE be set to the
*       fully-qualified name of a DLL file that contains a function called
*       Test whose format matches the prototype declaration below. That
*       function should alter the values passed in the third and fourth
*       paremeters and return the new values in the same parameters.
*
*\-----*/

#define ID_NEXT    1    // Return value if user presses 'Next >' Button
#define ID_CANCEL  2    // Return value if user presses 'Cancel' Button
#define ID_BACK   4    // Return value if user presses '< Back' Button

    INT    nValue, nResult;
    STRING szString, szResult, szDLL, szValue, szReturn;

program

    SetTitle ( "CallDLLFx Example", 18, WHITE );
    szDLL = SUPPORTDIR ^ "MYDLL.DLL";

    // Initialize inputs
    nValue = 3000;
    szString = "Test String";

    // Show inputs to users
    sprintfBox( INFORMATION, "", "Before - nValue: %i , szString: %s", nValue,
szString );
    // Call DLL function; pass by value.
    nResult = CallDLLFx( szDLL, "Test", nValue, szString );

    // Show values returned by DLL function
    sprintfBox( INFORMATION, "", "Returned - nValue: %i , szString: %s", nValue,
szString );

endprogram

// Source File : IS5FN614.RUL
```

{button ,PI("",`comment')} [Feedback](#)



Delay

Syntax

Delay (nSeconds);

Description

The Delay function delays the execution of a setup script by a specified number of seconds. Other tasks running simultaneously with InstallShield proceed normally while InstallShield is delayed.

Parameters

nSeconds

Specifies the number of seconds to delay program execution.

Return values

0

Indicates that the function successfully delayed the execution of the script.

< 0

Indicates that the function was unable to delay the execution of the script.

```
{button ,JI('LANGREF.HLP>Examples',`Delay_example')} Example  
{button ,PI("",`comment')} Feedback
```



Delay example

```
/*-----*\
 *
 * This example illustrates the use of the Delay function.
 *
 * SdShowMsg is called to display a message. Delay is then called to stall
 * the script for three seconds. After the message is hidden, Delay is called
 * again to stall for three seconds. Delay is called for a third
 * time to stall the script again displaying another message for another three
 * seconds.
 *
 \*-----*/

#include "Sddialog.h";

program

    SdShowMsg("This message will display for three seconds.", TRUE);
    Delay(3);

    SdShowMsg("", FALSE);
    Delay(3);

    SdShowMsg("This is another message that will display for a mere " +
        "three seconds.", TRUE);
    Delay(3);

endprogram

#include "Sddialog.rul"

// Source file: Is5fn203.rul
```

{button ,PI("",`comment')} [Feedback](#)



File and folder functions

File and folder functions provide a comprehensive way to work with text files, binary files, and folders. Many of the functions use the variables TARGETDIR and SRCDIR as the paths and accept only filenames as parameters. Wild card characters are also accepted where appropriate.



Do not specify long filenames when calling these functions in 16-bit setups.

ChangeDirectory

Makes the specified directory the current directory.

CloseFile

Closes an open file.

CopyFile

Copies a file from one folder to another.

CreateDir

Creates a new folder.

CreateFile

Creates a file with the specified filename.

DeleteDir

Deletes a folder.

DeleteFile

Deletes a file.

ExistsDir

Determines whether or not the specified directory exists.

ExistsDisk

Determines whether or not the specified disk exists.

FileCompare

Compares one file with another.

FileDeleteLine

Deletes a line in a text file.

FileGrep

Searches a text file for specified text.

FileInsertLine

Inserts a line in a text file.

FindAllDirs

Finds all subfolders under the specified folder.

FindAllFiles

Finds all files in the specified folder and its subfolders that match a file specification.

FindFile

Finds the first file in the specified folder that matches a file specification.

GetFileInfo

Retrieves a file's attributes, date, time, and size.

GetLine

Retrieves a line of text from an open file.

OpenFile

Opens an existing file.

OpenFileMode

Sets the mode in which files will be opened with the OpenFile function.

ReadBytes

Reads the specified number of bytes from a binary file.

RenameFile

Renames a file.

SeekBytes

Positions the file pointer in a binary file.

SetFileInfo

Sets the attributes, date, and time of a file.

WriteBytes

Writes a specified number of bytes to a binary file at the current file pointer location.

WriteLine

Writes a string to a text file.

XCopyFile

Copies one or more files from a source folder to a target folder, including subfolders if specified.

Related Function

SelectDir

Allows you to select a directory. The function creates the directory if it does not exist.

{button ,PI("",`comment`)} [Feedback](#)



ChangeDirectory

Syntax

ChangeDirectory (szPath);

Description

The ChangeDirectory function sets the current directory.

Parameters

szPath

Specifies the name of the directory to set as the current directory. That name can be either a fully qualified directory name or, in a 32-bit setup, a UNC path; it must not include a trailing backslash. If necessary, call [StrRemoveLastSlash](#) before calling ChangeDirectory.

Return values

0

ChangeDirectory successfully set the specified directory as the current directory.

< 0

ChangeDirectory was unable to set the specified directory as the current directory.

Comments

When a directory is set as the current directory, you cannot delete it using the [DeleteDir](#) function.

{button ,JI('LANGREF.HLP>Examples','ChangeDirectory_example')} [Example](#)

{button ,AL('DeleteDir;ExistsDir;GetDir;StrRemoveLastSlash',0,'')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



ChangeDirectory example

```
/*-----*\
 *
 * This example illustrates the use of the ChangeDirectory function.
 *
 * ChangeDirectory is used to change the current directory to NEW_PATH.
 *
 * If an application needs the current directory set to a specific directory,
 * ChangeDirectory can be used to set the current directory to execute the
 * application.
 *
 * NOTE: In order for this script to run properly, it is advised to set the
 *       preprocessor constants to valid filenames and paths.
 *
 \*-----*/

#define NEW_PATH      "C:\\WINDOWS"
#define APPLICATION  "NOTEPAD.EXE"
#define CMD_LINE      "EXAMPLE.TXT"

    STRING szPath, szCommand, szCmdLine;

program

    szPath = NEW_PATH;

/*-----*\
 *
 * In order to launch the specific application and command line,
 * ChangeDirectory must be called.
 *
 \*-----*/
    ChangeDirectory(szPath);

    szCommand = APPLICATION;
    szCmdLine = CMD_LINE;
    LaunchApp(szCommand, szCmdLine);

endprogram

// Source file: Is5fn015.rul
```

{button ,PI("",`comment')} [Feedback](#)



CloseFile

Syntax

```
CloseFile (nvFileHandle);
```

Description

The CloseFile function closes a file that has been opened with a call to [OpenFile](#). You cannot read from or write to a file after you close it.

Parameters

nvFileHandle

Specifies the handle of the file you want to close.

Return values

0

Indicates that the function successfully closed the file.

< 0

Indicates that the function was unable to close the file.

{button ,JI('LANGREF.HLP>Examples','CloseFile_example')} [Example](#)

{button ,AL('OpenFile;OpenFileMode',0,','')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



CloseFile example

```
/*-----*\
*
* This example illustrates the usage of the OpenFile and CloseFile functions.
*
* OpenFile is called to open a file to be read. After the first line is read
* and displayed in this script, the CloseFile function is called to remove it
* from memory.
*
* NOTE: In order for this script to run properly, it is necessary for the
*       preprocessor constants to be set to a valid file name and path on the
*       target system.
*
\*-----*/

#define EXAMPLE_TXT "LICENSE.TXT"
#define EXAMPLE_DIR "C:\\WINDOWS"

    STRING  szFileName, szPath, szText, svLine;
    NUMBER  nFlag, nvFileHandle;

program

    // Set the file mode to normal.
    OpenFileMode(FILE_MODE_NORMAL);

    szFileName = EXAMPLE_TXT;
    szPath      = EXAMPLE_DIR;

/*-----*\
*
* The following opens the EXAMPLE_TXT file for editing.
*
\*-----*/
    if (OpenFile(nvFileHandle, szPath, szFileName) < 0) then
        MessageBox ("OpenFile failed.", SEVERE);
        abort;
    endif;

    // Read and display one line from the text file.
    GetLine(nvFileHandle, svLine);
    MessageBox(svLine, INFORMATION);

/*-----*\
*
* The following closes the EXAMPLE_TXT file.
*
\*-----*/

    if (CloseFile(nvFileHandle) < 0) then
        MessageBox("CloseFile failed.", SEVERE);
    endif;

endprogram

// Source file: Is5fn194.rul
```

{button ,PI("",`comment')} Feedback



CopyFile

Syntax

CopyFile (szSrcFile, szTargetFile);

Description

The CopyFile function copies a file from the source directory to the target directory. InstallShield uses the system variables SRCDIR and TARGETDIR as the source and target paths for the function. If you must set values for SRCDIR and TARGETDIR when using CopyFile, you can save the previous values using [VarSave](#) and restore them using [VarRestore](#). If you specify a directory in TARGETDIR that does not exist, the function creates the directory for you.

You cannot rename groups of files by using wildcards with CopyFile. You can, however, rename a single file using CopyFile.

To include subdirectories, call the [XCopyFile](#) function.

Parameters

szSrcFile

Specifies the name of the file to copy from the directory identified by the system variable SRCDIR. Note that the filename must be unqualified; do not specify a drive designation or path. To copy groups of files, use wild card characters in this parameter.

szTargetFile

Specifies the name to give to the copy of the file identified by szSrcFile. Note that this filename must be unqualified; do not specify a drive designation or path. The copy is created in the directory specified by the system variable TARGETDIR. If you specify a wild card character in szSrcFile, this parameter is ignored.



When a wildcard character is included in the filename specified by szSrcFile, the parameter szTargetFile is ignored; each source file that meets the wild card specification is copied to TARGETDIR with the same name. For that reason, CopyFile cannot be used to rename a group of files; SRCDIR and TARGETDIR must specify different directories if szSrcFile contains one or more wildcard characters.

Return values

0

Indicates function successfully copied the file(s) from source to target directory.

< 0

Indicates that the function was unable to copy the requested file due to one of the following conditions:

COPY_ERR_CREATEDIR

A target directory could not be created. Make sure that the path in the system variable TARGETDIR is syntactically correct and that you have access rights to the target drive.

COPY_ERR_MEMORY

The function was unable to allocate the memory required to complete the copy file process. Terminate as many running applications as possible to free memory.

COPY_ERR_NODISKSPACE

The function could not find enough disk space on the target drive to copy the files. Free disk space on the target drive.

COPY_ERR_OPENINPUT

The function was unable to open the input file in the system variable SRCDIR. Make sure the source file is a

valid filename and that both the source file and target directory exist.

COPY_ERR_OPENOUTPUT

The function was unable to copy the requested file.

COPY_ERR_TARGETREADONLY

The file in TARGETDIR is read-only. Remove the read-only attribute from the target file and try again.

All other negative values

Indicate some other unspecified error has occurred.

Comments

- n InstallShield uses the system variables SRCDIR and TARGETDIR as the source and target directories for the CopyFile operation. Files are copied as is—no translation is performed.
- n For file transfer, a preferable alternatives to CopyFile may be XCopyFile. XCopyFile can perform version checking, mark locked .dll and .exe files for update after system reboot, and increment registry reference counters for shared .dll and .exe files.
- n Since Windows 95 does not allow an empty file to be copied and Windows NT does not allow creation of empty files, CopyFile will not work under these platforms when used to copy empty files (Size = 0 KB).
- n After modifying .ini files with WriteProfString, you will need to flush the cache buffer on Windows 95 before using CopyFile. All .ini files are cached on Windows 95, which can cause a delay in writing changes to the specified files. This in turn can interfere with subsequent file operations. To avoid this problem, simply call WriteProfString with null parameters to force Windows 95 to write the data to the .ini file immediately:

```
WriteProfString ("c:\\test.ini", "Windows", "KeyboardDelay", "100");  
WriteProfString ("", "", "", ""); //null string ("") for all four parameters  
TARGETDIR = "c:\\temp";  
//CopyFile should now have access to updated file.  
CopyFile ("test.ini", "test.ini");
```

{button ,JI('LANGREF.HLP>Examples','CopyFile_example')} [Example](#)

{button ,AL('XCopyFile;VarRestore;VarSave',0,'')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



CopyFile example

```
/*-----*\
 *
 * This example illustrates the use of the CopyFile function.
 *
 * This script copies files in the SOURCE_DIR directory to the TARGET_DIR
 * directory.
 *
 * NOTE: In order for this script to run properly, it is necessary for the
 *       preprocessor constants to be set to valid paths on the target system.
 *
 \*-----*/

#define SOURCE_DIR "C:\\EXAMPLE\\SOURCE"
#define TARGET_DIR "C:\\EXAMPLE\\TARGET"

program

    // Set up source and target directories.
    SRCDIR      = SOURCE_DIR;
    TARGETDIR   = TARGET_DIR;

/*-----*\
 *
 * Copy all files in the source directory, including files in subdirectories,
 * to the target directory.
 *
 \*-----*/

    if (CopyFile("*.*", "*.*) < 0) then
        MessageBox ("Could not copy files.", SEVERE);
    else
        MessageBox ("Files successfully copied.", INFORMATION);
    endif;

endprogram

// Source file: Is5fn198.rul
```

{button ,PI("",`comment')} [Feedback](#)



CreateDir

Syntax

```
CreateDir (szDirPath);
```

Description

The CreateDir function creates one or more subdirectories on the target drive. You can use a path that contains subdirectories on more than one level, such as C:\Programs\Winapps\Myapp. If any subdirectory in the path does not exist, CreateDir creates it. For example, if neither C:\Programs\Winapps nor C:\Programs\Winapps\Myapp exists, CreateDir creates both subdirectories.

Parameters

szDirPath

Specifies the fully qualified path of the subdirectories you are creating. Separate each level in the path with the two backslashes, the backslash escape character (\\).

Return values

0

Indicates that the function successfully created the specified directory on the target drive.

< 0

Indicates that the function was unable to create the directory.

Comments

CreateDir will fail under the following conditions:

- n The path is illegal.
- n The drive or any subdirectory in the path is write-protected.
- n The drive name is invalid.
- n You do not have network privileges to create subdirectories.

{button ,JI('LANGREF.HLP>Examples','CreateDir_example')} [Example](#)

{button ,AL('DeleteDir;GetDir',0,'','')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



CreateDir example

```
/*-----*\
 *
 * This example illustrates the uses of the CreateDir function.
 *
 * At first, the user is asked to input a valid directory name.  If the
 * directory does not exist, CreateDir is called to create it.
 *
 * The second time CreateDir is called, a multi-level directory structure is
 * created.
 *
 *-----*/

#define DEFAULT_DIR "C:\\EXAMPLE\\DEF_DIR"
#define MULTI_DIR  "C:\\EXAMPLE\\N_DIR\\A_DIR"

    STRING svPath;

program

    // Back button not needed for initial dialog box.
    Disable(BACKBUTTON);

    svPath = DEFAULT_DIR;

    // Prompt user for path to be created.
    AskPath("Please enter a valid path.", "", svPath);

    // Check to see if that directory already exists.  If not, create it.
    if (ExistsDir (svPath) = EXISTS) then

        sprintfBox (WARNING, "CreateDir", "Directory %s already Exists!",
                    svPath);

/*-----*\
 *
 * The following creates the user-specified directory path.  If CreateDir
 * fails, a message is displayed.
 *
 *-----*/

    elseif (CreateDir(svPath) < 0) then

        MessageBox("Cannot create directory", SEVERE);
    else

        sprintfBox(INFORMATION, "CreateDir", "%s directory successfully " +
                    "created.", svPath);
    endif;

    svPath = MULTI_DIR;

/*-----*\
 *
 * The following creates an entire multilevel path if the subdirectories do
 * not exist.  If CreateDir fails, a message is displayed.
 *
 *-----*/

    if (CreateDir(svPath) < 0) then
```

```
    MessageBox("Failed to create directories.", WARNING);
else
    sprintfBox(INFORMATION, "CreateDir", "%s directory successfully " +
              "created.", svPath);
endif;

endprogram

// Source file: Is5fn199.rul
```

{button ,PI("",`comment`)} [Feedback](#)



CreateFile

Syntax

```
CreateFile (nvFileHandle, szPath, szFileName);
```

Description

The CreateFile function creates a new file. If a file with the same name exists, CreateFile overwrites it. Before you create a file with CreateFile, you must set the file mode with [OpenFileMode](#).

CreateFile leaves the newly created file open in read/write (binary file) or append (text file) mode so you can read from or write to the file using other functions such as [GetLine](#), [ReadBytes](#), [WriteLine](#), and [WriteBytes](#).



In addition to read/write or append mode, all newly created files automatically open in OF_SHARE_DENY_NONE mode. This means that the files are opened without denying other programs read or write access to the files. This mode has its roots in the Windows OpenFile API.

When you finish reading from and writing to a file, you must close the file using the CloseFile function.

Parameters

nvFileHandle

Returns the handle of the new file.

szPath

Specifies the fully qualified path of the subdirectory in which you want to create the new file.

szFileName

Specifies the name of the file to create.

Return values

0

Indicates that the function successfully created the new file.

< 0

Indicates that the function was unable to create the specified file.

Comments

- n CreateFile creates a new file and leaves the file open so you can then read from or write to the new file. To write to an existing file, you must first open the file in FILE_MODE_APPEND mode using the OpenFileMode and OpenFile functions.
- n The CreateFile function's actions are not logged for uninstallation when logging is enabled. If you want a file that's created by CreateFile to be logged for uninstallation, transfer a starter file with the filename you want to the target system using [XCopyFile](#) while logging is enabled. You enable and disable logging with the [Enable](#) and [Disable](#) functions. XCopyFile actions are logged when logging is enabled, so the starter file will be logged for uninstallation. After transferring the logged starter file, you can write to or overwrite the logged starter file using CreateFile and other file-related functions. The filename must remain unchanged or unInstallShield will be unable to uninstall it.

{button ,JI('LANGREF.HLP>Examples','CreateFile_example')} [Example](#)

{button ,AL('CloseFile;GetLine;OpenFile;OpenFileMode;ReadBytes;WriteBytes;WriteLine',0,','')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



CreateFile example

```
/*-----*\
 *
 * This example illustrates the CreateFile and WriteLine functions.
 *
 * Createfile is called to create a file EXAMPLE_TXT to store a string. This
 * string is written into the file by the WriteLine function.
 *
 * NOTE: In order for this script to run properly, you should set the
 *       preprocessor constants to a valid file name and path on the target
 *       system.
 *-----*/

#define EXAMPLE_DIR "C:\\\"
#define EXAMPLE_TXT "EXAMPLE.TXT"

    STRING  szPath, szFileName, szTitle, szMsg;
    NUMBER  nvFileHandle;

program

    // Set the file mode to append open files.
    OpenFileMode(FILE_MODE_APPEND);

/*-----*\
 *
 * Call the CreateFile function to create a new file and leave it open for
 * editing.
 *-----*/
    szPath      = EXAMPLE_DIR;
    szFileName  = EXAMPLE_TXT;
    szMsg       = "The CreateFile function created this text file.";

    if (CreateFile(nvFileHandle, szPath, szFileName) < 0) then

        MessageBox("CreateFile failed.", SEVERE);
        abort;

/*-----*\
 *
 * Call the WriteLine function to write a line into the open file.
 *-----*/

    elseif (WriteLine(nvFileHandle, szMsg) < 0) then
        MessageBox("WriteLine failed.", SEVERE);
    else

        szTitle = "CreateFile & WriteLine";
        szMsg    = "Successfully created and wrote to %s.";
        sprintf(szTitle, szMsg, szFileName);
    endif;

    // Close the open file.
    CloseFile(nvFileHandle);

endprogram
```

// Source file: Is5fn200.rul

{button ,PI("",`comment')} Feedback



DeleteDir

Syntax

```
DeleteDir(szDir, nFlag);
```

Description

The DeleteDir function deletes a subdirectory. Depending on the value you use in the parameter nFlag, you can delete a subdirectory only if it is empty, delete a subdirectory even if it contains files, or delete an entire root directory. Set nFlag with extreme caution.

Parameters

szDir

Specifies the path and the name of the directory you want to delete.

nFlag

Specifies deletion options. Pass one of the following predefined constants in this parameter:

ALLCONTENTS

Deletes the directory in szDir, including all subdirectories and files beneath it. The directory you are deleting must be a subdirectory and cannot be a root directory of the drive.

ONLYDIR

Deletes the directory in szDir only if it is empty. Otherwise, the function fails.

ROOT

Deletes the directory in szDir even if it is the root directory. If szDir is a root directory, DeleteDir will delete everything on the disk.

Return values

0

Indicates that the function successfully deleted the subdirectory.

< 0

Indicates that the function was unable to delete the subdirectory.

Comments

- n You cannot delete the current directory.
- n You cannot delete files on a network system where you lack the appropriate rights.
- n DeleteDir cannot delete read-only, hidden, or system files.
- n If DeleteDir encounters a read-only file, the function may fail after having deleted only some of the files in the subdirectory.

{button ,JI('LANGREF.HLP>Examples','DeleteDir_example')} [Example](#)

{button ,AL('DeleteFile',0,'')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



DeleteDir example

```
/*-----*\
 *
 * This example illustrates the DeleteDir function.
 *
 * CreateDir is first called to create a new directory. DeleteDir is then
 * called to delete it.
 *
 \*-----*/

#define EXAMPLE_DIR "NEWDIR"

    STRING szDir, szTitle;
    NUMBER nReturn, nFlag;

program

    // Create a new directory.
    szDir = EXAMPLE_DIR;
    szTitle = "DeleteDir Example";
    if (CreateDir(szDir) = 0) then

        sprintfBox(INFORMATION, szTitle, "%s created successfully.", szDir);
    else

        MessageBox("CreateDir failed.", SEVERE);
        abort;
    endif;

/*-----*\
 *
 * The following deletes the newly created directory. If the directory is not
 * empty, DeleteDir fails.
 *
 \*-----*/

    nFlag = ONLYDIR;
    nReturn = DeleteDir(szDir, nFlag);

    if (nReturn < 0) then
        MessageBox("DeleteDir failed.", SEVERE);
    else

        sprintfBox(INFORMATION, szTitle, "%s deleted successfully.", szDir);
    endif;

endprogram

// Source file: Is5fn204.rul
```

{button ,PI("",`comment')} [Feedback](#)



DeleteFile

Syntax

DeleteFile (szFile);

Description

The DeleteFile function deletes one or more files in the directory specified by the system variable TARGETDIR. DeleteFile cannot delete read-only, hidden, or system files. This function may not work if you try to delete files on a network system without the appropriate file-deletion rights.

Parameters

szFile

Specifies the name(s) of the file(s) to delete from the directory specified by the system variable TARGETDIR. Note that the filename must be unqualified; do not specify a drive designation or path. Wild card characters can be included in the szFile to delete more than one file.

Return values

0

Indicates that the function successfully deleted the specified file.

< 0

Indicates that the function was unable to delete the file.

Comments

You can use wild card characters with [FindFile](#) to locate files and then delete them with DeleteFile.

{button ,JI('LANGREF.HLP>Examples','DeleteFile_example')} [Example](#)

{button ,AL('DeleteDir;FindFile',0,'','')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



DeleteFile example

```
/*-----*\
*
* This example illustrates the usage of the DeleteFile function.
*
* DeleteFile is first called to delete the EXAMPLE.TXT file in the DEL_DIR
* directory. This function is called again to delete all .SYS files from the
* DEL_DIR directory.
*
* NOTE: In order for this script to run properly, it is necessary for the
*       preprocessor constants to be set to valid directory and file names on
*       the target system.
*
\*-----*/

#define DEL_DIR    "C:\\EXAMPLE"
#define DEL_FILE  "EXAMPLE.TXT"
#define DEL_FILES "*.SYS"

    STRING szTitle, szFile, szMsg;

program

    // This sets the target directory.
    TARGETDIR = DEL_DIR;
    szTitle   = "DeleteFile Example";

/*-----*\
*
* Delete DEL_FILE file from the target directory.
*
\*-----*/

    szFile = DEL_FILE;
    if (DeleteFile (szFile) < 0) then

        MessageBox("First call to DeleteFile failed.", SEVERE);
    else

        szMsg = "%s successfully deleted.";
        sprintfBox(INFORMATION, szTitle, szMsg, szFile);
    endif;

/*-----*\
*
* Delete DEL_FILES files from the target directory.
*
\*-----*/

    szFile = DEL_FILES;
    if (DeleteFile(szFile) < 0) then
        MessageBox("Second call to DeleteFile failed.", SEVERE);
    else

        szMsg = "%s successfully deleted.";
        sprintfBox(INFORMATION, szTitle, szMsg, szFile);
    endif;

endprogram
```

// Source file: Is5fn205.rul

{button ,PI("",`comment')} Feedback



ExistsDir

Syntax

ExistsDir (szPath);

Description

The ExistsDir function checks for the existence of a specified directory on the target system.

Parameters

szPath

Specifies the fully qualified path of the directory to find on the target system.

Return values

EXISTS

Indicates that the specified directory exists on the target system.

NOTEXISTS

Indicates that the specified directory does not exist on the target system.

{button ,JI('LANGREF.HLP>Examples','ExistsDir_example')} [Example](#)

{button ,AL('ExistsDisk',0,'')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



ExistsDir example

```
/*-----*\
 *
 * This example illustrates the usage of the ExistsDir function.
 *
 * AskPath is called to ask the user for a directory. ExistsDir is then
 * called to determine whether the directory exists.
 *
 \*-----*/

    STRING szPath, szTitle, szMsg;

program

    // Get the path to be created.
    AskPath("Please enter a path:", "", szPath);

    szTitle = "ExistsDir Example";

/*-----*\
 *
 * Check to see if the directory already exists.
 *
 \*-----*/

    if (ExistsDir(szPath) = EXISTS) then
        szMsg = "Directory %s already exists.";
        sprintfBox(INFORMATION, szTitle, szMsg, szPath);
    else
        szMsg = "Directory %s does not exist.";
        sprintfBox(INFORMATION, szTitle, szMsg, szPath);
    endif;

endprogram

// Source file: Is5fn210.rul
```

{button ,PI("",`comment')} [Feedback](#)



ExistsDisk

Syntax

ExistsDisk (szDisk);

Description

The ExistsDisk function checks for the existence of a specified disk drive on the target system.

Parameters

szDisk

Specifies the letter of the disk drive.

Return values

EXISTS

Indicates that the specified drive exists on the target system.

NOTEXISTS

Indicates that the specified drive does not exist on the target system.

{button ,JI('LANGREF.HLP>Examples','ExistsDisk_example')} [Example](#)

{button ,AL('ExistsDir',0,'')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



ExistsDisk example

```
/*-----*\
 *
 * This example illustrates the usage of the ExistsDisk function.
 *
 \*-----*/

    STRING szTitle, szMsg, svResultPath;

program

    szTitle = "ExistsDisk Example";

    szMsg = "Please enter the letter of the drive to check the existence of.";
    AskText(szMsg, "C", svResultPath);

/*-----*\
 *
 * ExistsDisk is called to see if the user-specified drive exists.
 *
 \*-----*/

    if (ExistsDisk(svResultPath) = EXISTS) then

        szMsg = "Yes, drive %s exists.";
        sprintfBox(INFORMATION, szTitle, szMsg, svResultPath);
    else

        szMsg = "Drive %s does not exist.";
        sprintfBox(INFORMATION, szTitle, szMsg, svResultPath);
    endif;

endprogram

// Source file: Is5fn211.rul
```

{button ,PI("",`comment')} [Feedback](#)



FindAllDirs

Syntax

FindAllDirs (szDir, nOp, listDirs);

Description

The FindAllDirs function searches an entire disk or directory structure and returns a string list of subdirectory names. If nOp is INCLUDE_SUBDIR, the search starts at the directory specified by szDir and continues searching the subdirectory structure.

Parameters

szDir

Specifies the name of the directory to search.

nOp

Specifies whether or not to search all of the subdirectories below szDir. Pass one of the following predefined constants in this parameter:

EXCLUDE_SUBDIR

Excludes subdirectories.

INCLUDE_SUBDIR

Includes subdirectories.

listDirs

Returns a list of fully qualified directory names.

Return values

0

FindAllDirs successfully generated the list of subdirectory names.

< 0

FindAllDirs function was unable to generate a list.

Comments

FindAllDirs searches a hierarchical subdirectory structure starting with the specified directory. If the specified directory is a root directory and nOp contains INCLUDE_SUBDIR, all the directory names on the entire disk are returned. You can use FindAllDirs to find either subdirectories of a certain directory, or you can use it to find all the directories on a disk.

{button ,JI('LANGREF.HLP>Examples','FindAllDirs_example')} [Example](#)

{button ,AL('FindAllFiles;FindFile;ListGetFirstString;ListGetNextString',0,'')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



FindAllDirs example

```
/*-----*\
 *
 * This example illustrates the usage of the FindAllDirs function.
 *
 * FindAllDirs is called to retrieve all directories under a path.
 * Subdirectories are included at the user's discretion.
 *
\*-----*/

#include "Sddialog.h";

LIST listDirs;
STRING szDefPath, svResultPath, szDir, szTitle, szMsg;
NUMBER nOp;

program

    Disable(BACKBUTTON);

    // Ask the user for a path of directories.
    szMsg = "Please enter a path to list the directories of:";
    AskPath(szMsg, szDefPath, svResultPath);

    szDir = svResultPath;

    // Create a STRING list for directory name first.
    listDirs = ListCreate(STRINGLIST);

    // Ask whether or not to include subdirectories.
    if (AskYesNo("Include subdirectories?", YES) = YES) then

        nOp = INCLUDE_SUBDIR;
/*-----*\
 *
 * Call FindAllDirs to search the path for all directories and subdirectories.
 * This data is assigned to a list.
 *
\*-----*/
        if (FindAllDirs(szDir, nOp, listDirs) < 0) then

            MessageBox("FindAllDirs failed", INFORMATION);
        endif;
    else

        nOp = EXCLUDE_SUBDIR;
/*-----*\
 *
 * Call FindAllDirs to search the path for all directories. This data is
 * assigned to a list.
 *
\*-----*/
        if (FindAllDirs(szDir, nOp, listDirs) < 0) then

            MessageBox("FindAllDirs failed", INFORMATION);
        endif;
    endif;

    szTitle = "FindAllDirs Example";
    szMsg = "All the directories under the specified path:";
```

```
// Display the list of directories.
SdShowInfoList(szTitle, szMsg, listDirs);

endprogram

#define SD_SINGLE_DIALOGS 1
#define SD_SHOWINFOLIST 1

#include "Sddialog.rul"

// Source file: Is5fn077.rul
```

{button ,PI("",`comment')} Feedback



FindAllFiles

Syntax

```
FindAllFiles (szDir, szFileName, svResult, nOp);
```

Description

The FindAllFiles function searches an entire subdirectory structure and returns the name of the first file with a particular file specification. If the argument passed to nOp is RESET, InstallShield starts searching at the directory specified in the parameter szDir and continues searching the subdirectory structure until it finds a file matching szFileName. If nOp equals CONTINUE, the search continues where it left off the last time the function was called. Call this function repeatedly to find all occurrences of files that match szFileName.

Parameters

szDir

Specifies the name of the directory to search.

szFileName

Specifies the file specification to search for. Wild card characters are allowed in this parameter.

svResult

Returns the fully qualified name of the first matching file if found. If FindAllFiles fails, svResult remains unchanged.

nOp

Indicates search options. Pass one of the following predefined constants in this parameter.

CONTINUE

Resumes the search at the location where the previous search stopped.

RESET

Starts the search at the beginning of the directory in szDir.

To find all occurrences of szFileName, start by calling FindAllFiles with nOp set to RESET; then call FindAllFiles in a loop with nOp set to CONTINUE.

Return values

0

Indicates that the function retrieved and returned a file that matched the specification.

< 0

Indicates that the function was unable to find a file that matched the specifications.

Comments

This function searches a hierarchical subdirectory structure starting with the specified directory. If the specified directory is the root directory, InstallShield searches the entire disk. The function stops at the first matching filename it finds.

The first time you call this function to begin a new search, set nOp to RESET. You can continue to search for all other occurrences of the specified file by setting nOp to CONTINUE and placing the function call in a loop that ends when the FindAllFiles function fails.



You cannot use the XCopyFile function within a FindAllFiles(..., RESET), and FindAllFiles(..., CONTINUE) loop. If you call XCopyFile inside a FindAllFiles loop, the filename returned by FindAllFiles(..., CONTINUE) may be incorrect.

{button ,JI('LANGREF.HLP>Examples','FindAllFiles_example')}

Example

{button ,AL('FindFile',0,'')} See also

{button ,PI('','comment')} Feedback



FindAllFiles example

```
/*-----*\
 *
 * This example illustrates the usage of the FindAllFiles function.
 *
 *-----*/

#include "Sddialog.h"

STRING szTitle, szMsg, svDir, svResult, svNumFiles, szQuestion, szDir;
STRING szFileName;
NUMBER nResult, nNumFiles;
LIST  filesList;

program

    szTitle    = "FindAllFiles Example";

selectdir:
    szMsg      = "Select a directory to list files.";
    svDir      = TARGETDISK + "\\\";

    // Select a search directory.
    SelectDir(szTitle, szMsg, svDir, FALSE);

    szDir = svDir;

askfile:
    szQuestion = "Select a file (type) to search for in " + szDir + ":";

    // Select a file type to search.
    if (AskText(szQuestion, "*.EXE", svResult) = BACK) then

        goto selectdir;
    endif;

    szFileName = svResult;

    // Create a string list for the file listing.
    filesList = ListCreate(STRINGLIST);

    // Set the file count to zero.
    nNumFiles = 0;

/*-----*\
 *
 * FindAllFiles returns the first file in the svResult string.
 *
 *-----*/

    nResult = FindAllFiles(szDir, szFileName, svResult, RESET);

    while(nResult = 0)

        // If a file was found in the specified directory, add the file to the
        // list and increment the file counter.
        ListAddString(filesList, svResult, AFTER);
        nNumFiles = nNumFiles + 1;

        // Find the next file using the CONTINUE option.
```

```
        nResult = FindAllFiles(szDir, szFileName, svResult, CONTINUE);
endwhile;

// Convert the file count to a string for display.
NumToStr(svNumFiles, nNumFiles);

szMsg = "There are " + svNumFiles + " " + szFileName + " files in " +
        szDir + ":";
if (SdShowInfoList(szTitle, szMsg, filesList) = BACK) then
    ListDestroy(filesList);
    goto askfile;
endif;

ListDestroy(filesList);

endprogram

#define SD_SINGLE_DIALOGS 1
#define SD_SHOWINFOLIST 1

#include "Sddialog.rul"

// Source file: Is5fn216.rul
```

{button ,PI("",`comment')} [Feedback](#)



FindFile

Syntax

FindFile (szPath, szFileName, svResult);

Description

The FindFile function searches a directory for a specified file. InstallShield returns the first matching file in the parameter svResult.

Parameters

szPath

Specifies the name of the directory to search. Subdirectories beneath this directory are not searched.

szFileName

Specifies the name of the file to search for. Wild card characters are allowed in this parameter.

svResult

Returns the name of the first file that matches szFileName. This parameter contains the unqualified filename (that is, the drive designation and path are not included).

Return values

0

Indicates that the function successfully found and returned the specified file.

< 0

Indicates that the function was unable to find the file.

Comments

This function searches only the specified subdirectory. It does not search an entire disk or directory tree.

{button ,JI('LANGREF.HLP>Examples','FindFile_example')} [Example](#)

{button ,AL('FindAllFiles',0,'')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



FindFile example

```
/*-----*\
*
* This example illustrates the usage of the FindFile function.
*
* FindFile is called to search for the file EXAMPLE_TXT in the EXAMPLE_DIR
* directory.
*
* NOTE: In order for this script to run properly, it is necessary for the
*       preprocessor constants to be set to a valid file name and path on the
*       target system.
*
*\-----*/

#define EXAMPLE_TXT "EXAMPLE.TXT"
#define EXAMPLE_DIR "C:\\EXAMPLE"

    STRING szTitle, szPath, szFileName, svResult;

program

    szTitle = "FindFile Example";

    szPath = EXAMPLE_DIR;
    szFileName = EXAMPLE_TXT;
    if (FindFile(szPath, szFileName, svResult) < 0) then
        MessageBox("FindFile failed.", SEVERE);
    else
        sprintfBox(INFORMATION, szTitle, "Found: %s in %s.", svResult, szPath);
    endif;

endprogram

// Source file: Is5fn217.rul
```

{button ,PI("",`comment')} [Feedback](#)



GetFileInfo

Syntax

```
GetFileInfo (szPathName, nType, nvResult, svResult);
```

Description

Call the GetFileInfo function to determine a file's attributes, date, time, or size. In each GetFileInfo statement, you can ask for only one of these items of data. For example, to get the date and time information for a file, you must call GetFileInfo twice, once to obtain the date and once to obtain the time.

Parameters

szPathName

Specifies the fully qualified name of the file about which you want to retrieve information.

nType

Specifies the type of file information to retrieve. If the information is a number, GetFileInfo places it in nvResult. If the information is a string, GetFileInfo places it in svResult. Pass one of the following predefined constants in this parameter:

FILE_ATTRIBUTE

The file attribute is returned in nvResult. See Comments.

FILE_DATE

The file date in the format YYYYMMDD is returned in svResult.

FILE_SIZE

The size in bytes is returned in nvResult.

FILE_TIME

The file time in the format HH:MM:SS is returned in svResult.

nvResult

Returns numeric information.

svResult

Returns string information.

Return values

0

Indicates that the function successfully retrieved the requested file information.

< 0

Indicates that the function was unable to retrieve the requested information.

Comments

After calling GetFileInfo with FILE_ATTRIBUTE as the second parameter (nType), use if-then-else logic to determine the file's attributes. If nvResult = FILE_ATTR_NORMAL, then no file attributes are set. If nvResult <> FILE_ATTR_NORMAL, test the result of bitwise AND (&) operations on nvResult and one or more the file attribute constants to determine which attributes are set.

```
if (nvResult = FILE_ATTR_NORMAL) then
    //The file is NORMAL.
else
    if (FILE_ATTR_HIDDEN & nvResult) then
        //The file is HIDDEN.
    endif;
    if (FILE_ATTR_READONLY & nvResult) then
        //The file is READ-ONLY.
```

```
endif;  
endif;
```

{button ,JI('LANGREF.HLP>Examples','GetFileInfo_example')} Example

{button ,AL('FindAllFiles;FindFile;SetFileInfo',0,'')} See also

{button ,PI('','comment')} Feedback

File attributes

FILE_ATTR_NORMAL

The file is a normal file.

FILE_ATTR_ARCHIVED

The file is archived.

FILE_ATTR_DIRECTORY

The file is a directory.

FILE_ATTR_HIDDEN

The file is hidden.

FILE_ATTR_READONLY

The file is read-only.

FILE_ATTR_SYSTEM

The file is a system file.

{button ,PI("",`comment')} [Feedback](#)



GetFileInfo example

```
/*-----*\
 *
 * This example illustrates the GetFileInfo function.
 *
 * GetFileInfo is called to retrieve the time, date, and attributes for a file.
 *
 * NOTE: In order for this script to run properly, it is necessary for the
 *       EXAMPLE_TXT constant to be set to a valid file name on the target
 *       system.
 *
 *-----*/

#define EXAMPLE_TXT "C:\\EXAMPLE\\EXAMPLE.TXT"

#include "Sddialog.h"

STRING svResult, szPathName, szTitle;
NUMBER nvResult, nType;
LIST listID;

program

start:

    szPathName = EXAMPLE_TXT;
    nType      = FILE_TIME;
    szTitle    = "GetFileInfo Example";

/*-----*\
 *
 * GetFileInfo is called to retrieve the time the file was created.
 *
 *-----*/

    if (GetFileInfo(szPathName, nType, nvResult, svResult) < 0) then
        MessageBox("First call to GetFileInfo failed.", SEVERE);
    else
        sprintfBox(INFORMATION, szTitle, "Time: %s", svResult);
    endif;

/*-----*\
 *
 * GetFileInfo is called again to retrieve the date the file was created.
 *
 *-----*/

    nType = FILE_DATE;
    if (GetFileInfo(szPathName, nType, nvResult, svResult) < 0) then
        MessageBox("Second call to GetFileInfo failed.", SEVERE);
    else
        sprintfBox(INFORMATION, szTitle, "Date: %s", svResult);
    endif;

/*-----*\
 *
 * GetFileInfo is called for the last time to find the file's attributes.
 *
 *-----*/
```

```

nType      = FILE_ATTRIBUTE;
if (GetFileInfo(szPathName, nType, nvResult, svResult) < 0) then
    MessageBox("Third call to GetFileInfo failed.", SEVERE);
    exit;
endif;

listID = ListCreate (STRINGLIST);

// Test to see if the file is a normal file (no attributes set).
// If it is not, then do a series of bitwise AND operations to
// test for each attribute.
if (nvResult = FILE_ATTR_NORMAL) then
    ListAddString(listID, "The file is NORMAL.", AFTER);
else
    if (FILE_ATTR_ARCHIVED & nvResult) then
        ListAddString(listID, "The file is ARCHIVED.", AFTER);
    endif;

    if (FILE_ATTR_HIDDEN & nvResult) then
        ListAddString(listID, "The file is HIDDEN.", AFTER);
    endif;

    if (FILE_ATTR_READONLY & nvResult) then
        ListAddString(listID, "The file is READ-ONLY.", AFTER);
    endif;

    if (FILE_ATTR_SYSTEM & nvResult) then

        ListAddString(listID, "The file is a SYSTEM FILE.", AFTER);
    endif;

    if (FILE_ATTR_DIRECTORY & nvResult) then

        ListAddString(listID, "The file is a DIRECTORY.", AFTER);
    endif;
endif;

// Display the list.
SdShowInfoList(szTitle, "The attributes of szPathName:", listID);

endprogram

#define    SD_SINGLE_DIALOGS    1
#define    SD_SHOWINFOLIST    1

#include    "Sddialog.rul"

// Source file: Is5fn224.rul
//

```

{button ,PI("",`comment')} [Feedback](#)



GetLine

Syntax

```
GetLine (nvFileHandle, svLine);
```

Description

The GetLine function reads a line of text from a text file opened in read-only mode. Before you call GetLine, you must first call [OpenFileMode](#) to set the file mode to read-only and then call [OpenFile](#) to open the file. The first call to GetLine reads the first line of text from the file. After reading a line, GetLine repositions the file pointer to the next line. The second call to GetLine reads the second line, and so forth. GetLine strips the carriage return and line feed characters from the end of the line it returns.

To write to a text file, use the [WriteLine](#) function. WriteLine always produces lines that have a carriage return and line feed character combination at the end of the line.

Parameters

nvFileHandle

Specifies the handle of a file that has been opened by a call to [OpenFile](#).

svLine

Returns a line of text from the file specified by nvFileHandle.

Return values

0

Indicates that the function successfully retrieved a line of text from an open text file.

< 0

Indicates that the function failed due to an end of file error or another error condition. This condition also indicates GetLine has read all the lines in the file.

Comments

- The maximum size of a line is 512 bytes in 16-bit setups and 1,024 bytes in 32-bit setups. To read multiple lines from a file, you can use a separate GetLine call for each line or place the GetLine statement in a loop.
- When GetLine has read all the lines in a file, it returns an end-of-file error. Note that if you open a file in append mode, the GetLine function will fail if you call it because the file pointer is at the end of the file. The function will also fail if the file specified by nvFileHandle was opened in a binary mode.

{button ,JI('LANGREF.HLP>Examples','GetLine_example')} [Example](#)

{button ,AL('OpenFile;WriteLine',0,'')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



GetLine example

```
/*-----*\
 *
 * This example illustrates the usage of the GetLine function.
 *
 * GetLine is called in this script to read a text file line by line.  GetLine
 * is called multiple times within a while loop until the end-of-file error is
 * returned.
 *
 * NOTE: In order for this script to run properly, it is necessary for the
 *       preprocessor constants to be set to a valid filename and path on the
 *       target system.
 *
 \*-----*/

#define EXAMPLE_TXT "LICENSE.TXT"
#define EXAMPLE_DIR "EXAMPLE"

    STRING  szFileName, szPath, szText, svLine;
    NUMBER  nFlag, nFileHandle;

program

    // Set the file mode to normal.
    OpenFileMode(FILE_MODE_NORMAL);

    szFileName = EXAMPLE_TXT;
    szPath      = EXAMPLE_DIR;

    // The following opens the EXAMPLE_TXT file for editing.
    OpenFile(nFileHandle, szPath, szFileName);

/*-----*\
 *
 * The following retrieves each line of text from the opened text files.  A
 * message box is then called to display it.
 *
 \*-----*/

    while (GetLine (nFileHandle, svLine) = 0)
        SprintfBox (INFORMATION, "GetLine Example", "%s", svLine);
    endwhile;

    // The following closes the EXAMPLE_TXT file.
    CloseFile(nFileHandle);

endprogram

// Source file: Is5fn225.rul
```

{button ,PI("",`comment')} [Feedback](#)



OpenFile

Syntax

OpenFile (nvFileHandle, szPath, szFileName);

Description

The OpenFile function opens an existing text file or binary file. Before you open the file you must set the file mode by calling [OpenFileMode](#).

After you open a text file, call [GetLine](#) and [WriteLine](#) to read from and write to the file. After you open a binary file, call [ReadBytes](#) and [WriteBytes](#) to read from and write to the file. You may need to use [SeekBytes](#) to position the file pointer before writing to a binary file.



You can also search, read from, and write to text files using the [FileGrep](#), [FileInsertLine](#), and [FileDeleteLine](#) functions. However, these functions do not require you to open or close the files (this is handled internally). Depending on your needs, these functions may serve you better.

When you finish reading from or writing to a file with [GetLine](#) or [WriteLine](#), you must close the file using the [CloseFile](#) function.

Use [CreateFile](#) to create a file. CreateFile leaves the new file open in append mode (text files) or read/write mode (binary files).

Parameters

nvFileHandle

Returns the file handle of the file that was opened. Use this handle to identify the file when you call other file-related InstallShield functions.

szPath

Specifies the path (which may include a drive designation), of the file you want to open.

szFileName

Specifies the unqualified name of the file you want to open.

Return values

0

Indicates that the function successfully opened the file.

< 0

Indicates that the function was unable to open the file.

{button ,JI('LANGREF.HLP>Examples','OpenFile_example')} [Example](#)

{button ,AL('CloseFile;CreateFile;GetLine;OpenFileMode;ReadBytes;SeekBytes;WriteBytes;WriteLine',0,','')}
[See also](#)

{button ,PI('','comment')} [Feedback](#)



OpenFile example

```
/*-----*\
*
* This example illustrates the usage of the OpenFile and CloseFile functions.
*
* OpenFile is called to open a file to be read. After the first line is read
* and displayed in this script, the CloseFile function is called to remove it
* from memory.
*
* NOTE: In order for this script to run properly, it is necessary for the
*       preprocessor constants to be set to a valid file name and path on the
*       target system.
*
\*-----*/

#define EXAMPLE_TXT "LICENSE.TXT"
#define EXAMPLE_DIR "C:\\WINDOWS"

    STRING  szFileName, szPath, szText, svLine;
    NUMBER  nFlag, nvFileHandle;

program

    // Set the file mode to normal.
    OpenFileMode(FILE_MODE_NORMAL);

    szFileName = EXAMPLE_TXT;
    szPath      = EXAMPLE_DIR;

/*-----*\
*
* The following opens the EXAMPLE_TXT file for editing.
*
\*-----*/
    if (OpenFile(nvFileHandle, szPath, szFileName) < 0) then
        MessageBox ("OpenFile failed.", SEVERE);
        abort;
    endif;

    // Read and display one line from the text file.
    GetLine(nvFileHandle, svLine);
    MessageBox(svLine, INFORMATION);

/*-----*\
*
* The following closes the EXAMPLE_TXT file.
*
\*-----*/

    if (CloseFile(nvFileHandle) < 0) then
        MessageBox("CloseFile failed.", SEVERE);
    endif;

endprogram

// Source file: Is5fn194.rul
```

{button ,PI("",`comment')} Feedback



OpenFileMode

Syntax

OpenFileMode (nMode);

Description

The OpenFileMode function sets the mode of the file you want to open or create. The argument you pass as the parameter nMode sets the file mode to one of the following:

- n Text file in append mode.
- n Text file in read-only mode.
- n Binary file in read/write mode.
- n Binary file in read-only mode.

After you set the file mode, call [OpenFile](#) to open an existing file or [CreateFile](#) to create and open a new file.

Parameters

nMode

Specifies which mode to use to open a file. Pass one of the following predefined constants in this parameter:

FILE_MODE_APPEND

This constant allows a text file to be opened or created in append mode. When a file is opened in append mode using OpenFile, the file pointer is at the end of the file. You can use the [WriteLine](#) function to append lines to the end of the file. Files created using CreateFile are new (empty), so lines appended to the files are written at the beginning of the files. Note that if you open a file in append mode, the [GetLine](#) function will fail if you call it because the file pointer is at the end of the file.

FILE_MODE_NORMAL

This constant allows a text file to be opened in read-only mode. When a file is opened in read-only mode using OpenFile, the file pointer is at the beginning of the file. You can use the [GetLine](#) function to read from the file. Files created using CreateFile when FILE_MODE_NORMAL is in effect will actually be created in FILE_MODE_APPEND mode.

FILE_MODE_BINARY

This constant allows a binary file to be opened and created in read/write mode. When you open or create a file in binary mode with OpenFile or CreateFile, you can call [ReadBytes](#) to read from the file and [WriteBytes](#) to write to the file. Writing to a binary file begins at the current file pointer position, which for a newly opened or created file is position 0—the beginning of the file. If you want to append to an existing binary file opened using OpenFile, you must use [SeekBytes](#) to position the file pointer before writing. To open a file on a CD-ROM or on a read-only drive, call OpenFileMode to set the file mode to read-only (FILE_MODE_BINARYREADONLY).

FILE_MODE_BINARYREADONLY

This constant is just like the constant FILE_MODE_BINARY, except that it opens the binary file in read-only mode. When opening a binary file on CD-ROM or read-only drives, use this constant to open a binary file. FILE_MODE_BINARY will fail opening binary files on CD-ROM or read-only drives.

Return values

0

Indicates that the function successfully set the file mode.

< 0

Indicates that the function was unable to set the file mode.

{button ,JI('LANGREF.HLP>Examples','OpenFileMode_example')}

[Example](#)

{button ,AL(`CloseFile;CreateFile;GetLine;OpenFile;ReadBytes;SeekBytes;WriteBytes;WriteLine;Setting a text files mode',0,';`')} See also

{button ,PI("",`comment')} Feedback



OpenFileMode example

```
/*-----*\
 *
 * This example illustrates the usage of the FileModeOpen function.
 *
 * The example opens a text file in read-only (FILE_MODE_NORMAL) mode.
 * It then retrieves and displays one line at a time from the file.
 *
 \*-----*/

#define EXAMPLE_TXT "LICENSE.TXT"
#define EXAMPLE_DIR "EXAMPLE"
#define EXAMPLE_BIN "EXAMPLE.INS"

    STRING svLine, svString;
    NUMBER nMode, nvFileHandle, nPosition, nIndex, nBytes;

program

/*-----*\
 *
 * Set the file mode to normal.
 *
 \*-----*/

    nMode = FILE_MODE_NORMAL;
    OpenFileMode(nMode);

    if (OpenFile(nvFileHandle, EXAMPLE_DIR, EXAMPLE_TXT) < 0) then
        MessageBox("OpenFile failed.", SEVERE);
    endif;

    // Display the first line of the text file.
    GetLine(nvFileHandle, svLine);
    MessageBox (svLine, INFORMATION);

    CloseFile(nvFileHandle);

/*-----*\
 *
 * Set the file mode to binary read/write.
 *
 \*-----*/

    nMode = FILE_MODE_BINARY;
    OpenFileMode(nMode);

    // Open a binary file.
    if (OpenFile(nvFileHandle, EXAMPLE_DIR, EXAMPLE_BIN) < 0) then
        MessageBox("OpenFile failed.", SEVERE);
    endif;

    nPosition = 15;

    // Set the cursor to the fifteenth byte in the binary file.
    SeekBytes(nvFileHandle, nPosition, FILE_BIN_START);

    nIndex = 0;
    nBytes = 28;
```

```
// Retrieve bytes from the binary file.
if (ReadBytes(nvFileHandle, svString, nIndex, nBytes) < 0) then
    MessageBox("ReadBytes failed.", SEVERE);
else
    // Display the string.
    MessageBox(svString, INFORMATION);
endif;

// Close the binary file.
CloseFile(nvFileHandle);

endprogram

// Source file: Is5fn235.rul
```

{button ,PI("",`comment')} Feedback



RenameFile

Syntax

RenameFile (szFileOld, szFileNew);

Description

The RenameFile function changes the name of a file or moves a file from one directory to another. Note that both parameters must specify unqualified filenames (that is, filenames without a drive designation or path). InstallShield uses the system variables SRCDIR and TARGETDIR to locate szFileOld and szFileNew, respectively.

- n To rename a file, set SRCDIR and TARGETDIR to the directory in which the file resides before calling RenameFile.
- n To move a file, set SRCDIR to the current location of szFileOld and set TARGETDIR to its new location before calling RenameFile.

Parameters

szFileOld

Specifies the current unqualified name of a file that is located in the directory specified by the system variable SRCDIR. Note that wild card characters are not permitted; only one file can be renamed or moved with each call to RenameFile.

szFileNew

Specifies an unqualified file name; a file with that name must not already exist in the directory specified by TARGETDIR. Note that when the file is to be renamed (that is, when SRCDIR and TARGETDIR specify the same directory, szFileOld and szFileNew must specify different filenames. However, when the file is to be moved, szFileNew may specify the same unqualified name as szFileOld.

Return values

0

Indicates that the function successfully changed the filename or moved the file.

< 0

Indicates that the function was unable to change the filename or move the file.

{button ,JI('LANGREF.HLP>Examples','RenameFile_example')}

Example

{button ,AL('CopyFile;FindFile',0,'')}

See also

{button ,PI('','comment')} Feedback



RenameFile example

```
/*-----*\
*
* This example illustrates the RenameFile function.
*
* RenameFile is called to first rename FILENAME1 to FILENAME2. It is then
* called again to move this FILENAME2 file to the TARGET directory.
*
* This can also be done in one call to RenameFile. The third call to
* RenameFile explains this by renaming and moving the FILENAME2 file from the
* TARGET directory to the SOURCE directory, with the FILENAME1 file name.
*
* NOTE: In order for this script to run properly, it is necessary for the
*       preprocessor constants to be set to valid file names and paths on the
*       target system.
*
\*-----*/

#define FILENAME1 "EXAMPLE.TXT"
#define FILENAME2 "EXAMPLE.BAK"
#define SOURCE_DIR "C:\\EXAMPLE\\SOURCE"
#define TARGET_DIR "C:\\EXAMPLE\\TARGET"

    STRING szFileOld, szFileNew, szTitle, szMsg;

program

    szTitle = "RenameFile Example";

    // Do not change directories.
    SRCDIR   = "C:\\EXAMPLE\\SOURCE";
    TARGETDIR = "C:\\EXAMPLE\\SOURCE";

    szFileOld = FILENAME1;
    szFileNew = FILENAME2;
/*-----*\
*
* Rename FILENAME1 to FILENAME2.
*
\*-----*/

    if (RenameFile (szFileOld, szFileNew) < 0) then
        MessageBox("First call to RenameFile failed.", SEVERE);
        abort;
    else
        szMsg = "%s successfully renamed to %s.";
        sprintf(szTitle, szMsg, szFileOld, szFileNew);
    endif;

    // Change directories.
    SRCDIR   = "C:\\EXAMPLE\\SOURCE";
    TARGETDIR = "C:\\EXAMPLE\\TARGET";

    // Assign the same value to both file names.
    szFileOld = szFileNew;
/*-----*\
*
* Move the file from the SOURCE to the TARGET directory.
*
\*-----*/
```

```

\*-----*/

if (RenameFile(szFileOld, szFileNew) < 0) then
    MessageBox("Second call to RenameFile failed.", SEVERE);
    abort;
else
    szMsg = "%s successfully moved to %s.";
    sprintfBox(INFORMATION, szTitle, szMsg, szFileNew, TARGETDIR);
endif;

// Change directories.
SRCDIR    = "C:\\EXAMPLE\\TARGET";
TARGETDIR = "C:\\EXAMPLE\\SOURCE";

// Change file names, as well.
szFileOld = FILENAME2;
szFileNew = FILENAME1;

/*-----*\
*
* Rename the file from the TARGET directory to the SOURCE directory.
*
\*-----*/

if (RenameFile(szFileOld, szFileNew) < 0) then
    MessageBox("Third call to RenameFile failed.", SEVERE);
    abort;
else
    szMsg = "%s successfully renamed to %s in the %s directory.";
    sprintfBox(INFORMATION, szTitle, szMsg, szFileOld, szFileNew,
                TARGETDIR);
endif;

endprogram

// Source file: Is5fn249.rul

```

{button ,PI("",`comment')} [Feedback](#)

{button ,PI("", `comment`)} Feedback



SetFileInfo

Syntax

```
SetFileInfo (szPathFile, nType, nAttribute, szValue);
```

Description

The SetFileInfo function sets the date or time stamp of an existing file or changes the file's attributes. To change both a file's date and time you must call SetFileInfo twice, once to change the date and once to change the time. However you can set multiple file attributes with a single call to SetFileInfo by combining constants in nAttribute with the OR (|) operator.

Parameters

szPathFile

Specifies the fully qualified name of the file whose date, time, or attributes you want to change.

nType

Specifies the file characteristic to change. Pass one of the following predefined constants in this parameter:

FILE_ATTRIBUTE

Indicates that one or more of the file's attributes will be changed.

FILE_DATE

Indicates that the file's date stamp will be changed.

FILE_TIME

Indicates the file's time stamp will be changed.

nAttribute

Specifies the file attribute when nType is FILE_ATTRIBUTE. (When nType is FILE_DATE or FILE_TIME, pass 0 in this parameter.) To specify more than one file attribute, combine one or more of the following predefined constants with the OR (|) operator.

FILE_ATTR_ARCHIVED

Sets the archive attribute.

FILE_ATTR_HIDDEN

Sets the hidden attribute.

FILE_ATTR_READONLY

Sets the read-only attribute.

FILE_ATTR_SYSTEM

Sets the system attribute.

FILE_ATTR_NORMAL

When this constant is specified alone, all file attributes are cleared. When this constant is ORed with other file attribute constants, those attributes not included in the OR expression are cleared.

szValue

Specifies one of the following:

- When nType is FILE_DATE, specifies a date in the format YYYY/MM/DD. That date must not be earlier than "1980/01/01".
- When nType is FILE_TIME, specifies a time in the format HH:MM:SS, using a 24-hour clock format; note that the seconds must be a multiple of 2.

When nType is FILE_ATTRIBUTE, pass a null string ("") in this parameter.

Return values

0

Indicates that the function successfully set the file's date stamp, time stamp or attributes.

< 0

Indicates that the function was unable to set the file's date stamp, time stamp or attributes.

{button ,JI('LANGREF.HLP>Examples','SetFileInfo_example')} [Example](#)

{button ,AL('FindAllFiles;FindFile;GetFileInfo',0,','')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



SetFileInfo example

```
/*-----*\
 *
 * This example illustrates the usage of the SetFileInfo function.
 *
 * SetFileInfo is called to set a new date, time, and attributes for
 * EXAMPLE_TXT.
 *
 \*-----*/

#define EXAMPLE_TXT "C:\\EXAMPLE\\EXAMPLE.TXT"

    STRING szPathFile, szResult, svResult, svReturnString, szMsg, szTitle;
    NUMBER nType, nResult;

program

start:

    // Prompt the user for the target file's new date.
    AskText("Please enter the new date (YYYY/MM/DD):", "1996/08/01", svResult);

    szPathFile = EXAMPLE_TXT;
    nType      = FILE_DATE;
    nResult    = 0;
    szResult   = svResult;

    // Retrieve only the file name of the target file, for use in message boxes.
    ParsePath(svReturnString, szPathFile, FILENAME);

/*-----*\
 *
 * Set the new date to the target file.
 *
 \*-----*/

    if (SetFileInfo(szPathFile, nType, nResult, szResult) < 0) then
        MessageBox("First call to SetFileInfo failed.", SEVERE);
    else

        szMsg = "The date for %s has changed to %s.";
        SprintfBox(INFORMATION, szTitle, szMsg, svReturnString, szResult);
    endif;

    // Prompt the user for the target file's new time.
    AskText("Please enter the new time (HH:MM:SS):", "13:02:24", svResult);

    nType      = FILE_TIME;
    nResult    = 0;
    szResult   = svResult;

/*-----*\
 *
 * Set the new time to the target file.
 *
 \*-----*/

    if (SetFileInfo(szPathFile, nType, nResult, szResult) < 0) then
        MessageBox("Second call to SetFileInfo failed.", SEVERE);
    else
```

```
        szMsg = "The time for %s has changed to %s.";
        sprintfBox(INFORMATION, szTitle, szMsg, svReturnString, szResult);
    endif;

    nType      = FILE_ATTRIBUTE;
    nResult    = FILE_ATTR_READONLY | FILE_ATTR_HIDDEN;
    szResult   = "";

/*-----*\
 *
 * Set the target file to be read-only and hidden.
 *
 *-----*/

    if (SetFileInfo(szPathFile, nType, nResult, szResult) < 0) then
        MessageBox("Third call to SetFileInfo failed.", SEVERE);
    else
        szMsg = "%s has changed to a read-only and hidden file.";
        sprintfBox(INFORMATION, szTitle, szMsg, svReturnString);
    endif;

endprogram

// Source file: Is5fn254.rul
```

{button ,PI("",`comment')} Feedback



WriteLine

Syntax

```
WriteLine (nvFileHandle, szLine);
```

Description

The WriteLine function writes a line of text to a text file opened in append mode. You must first set the file mode to append mode with [OpenFileMode](#), and then either create the file with [CreateFile](#), or open the file with [OpenFile](#), before calling WriteLine. This function places the line at the end of the file.

WriteLine produces lines that have a carriage return and line feed character at the end of the line. To write to a binary file, use [WriteBytes](#).



This function does not work with files opened in read-only mode.

Parameters

nvFileHandle

Specifies the file handle of an open file. The handle is obtained from [OpenFile](#) or [CreateFile](#).

szLine

Specifies a string that contains the text to write to the file.



Do not attempt to write multiple lines by embedding newline characters in the string you pass in szLine. The following code writes the string as one line, with an unprintable character between "one" and "This":

```
szString = "This is line one\nThis is two";  
WriteLine(nvFileHandle, szString);
```

To write two lines with one call to WriteLine, you must embed a return and a newline (in that order):

```
szString = "This is line one\r\nThis is two";
```

Return values

0

Indicates that the function successfully wrote the line to the file.

< 0

Indicates that the function was unable to write the line to the file.

Comments

You can embed double quotation marks inside a string by delimiting the string with single quotation marks. For example, if you want to write the string "This string contains a double "quotation mark.", you can use the following code:

```
WriteLine(nvFileHandle, 'This string contains a double "quotation mark."');
```

{button ,JI('LANGREF.HLP>Examples','WriteLine_example')} [Example](#)

{button ,AL('CloseFile;CreateFile;OpenFile;OpenFileMode',0,'')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



WriteLine example

```
/*-----*\
 *
 * This example illustrates the CreateFile and WriteLine functions.
 *
 * Createfile is called to create a file EXAMPLE.TXT to store a string. This
 * string is written into the file by the WriteLine function.
 *
 * NOTE: In order for this script to run properly, you should set the
 *       preprocessor constants to a valid file name and path on the target
 *       system.
 *-----*/

#define EXAMPLE_DIR "C:\\\"
#define EXAMPLE_TXT "EXAMPLE.TXT"

    STRING  szPath, szFileName, szTitle, szMsg;
    NUMBER  nvFileHandle;

program

    // Set the file mode to append open files.
    OpenFileMode(FILE_MODE_APPEND);

/*-----*\
 *
 * Call the CreateFile function to create a new file and leave it open for
 * editing.
 *-----*/
szPath      = EXAMPLE_DIR;
szFileName  = EXAMPLE_TXT;
szMsg       = "The CreateFile function created this text file.";

if (CreateFile(nvFileHandle, szPath, szFileName) < 0) then

    MessageBox("CreateFile failed.", SEVERE);
    abort;

/*-----*\
 *
 * Call the WriteLine function to write a line into the open file.
 *-----*/

elseif (WriteLine(nvFileHandle, szMsg) < 0) then
    MessageBox("WriteLine failed.", SEVERE);
else

    szTitle = "CreateFile & WriteLine";
    szMsg   = "Successfully created and wrote to %s.";
    sprintf(INFORMATION, szTitle, szMsg, szFileName);
endif;

// Close the open file.
CloseFile(nvFileHandle);

endprogram
```

// Source file: Is5fn200.rul

{button ,PI("",`comment')} Feedback



XCopyFile

Syntax

XCopyFile (szSrcFile, szTargetFile, nOp);

Description

The XCopyFile function copies one or more files from the source directory to the target directory. This function can copy subdirectories as well as files. It uses the system variables SRCDIR and TARGETDIR as the source and target paths respectively. If you need to temporarily set values for SRCDIR and TARGETDIR when using XCopyFile, you can save the previous values using [VarSave](#) and restore them using [VarRestore](#).

XCopyFile creates subdirectories on the target directory if necessary when the constant INCLUDE_SUBDIR is passed in the parameter nOp.

You currently cannot rename files using XCopyFile. To rename a file during a file copy operation, use the [CopyFile](#) function.

Parameters

szSrcFile

Specifies the unqualified name (that is, without a drive designation or path) of the file to copy. This file must reside on the directory specified by the system variable SRCDIR. To copy multiple files, use wild card characters in this parameter.

szTargetFile

Currently, szTargetFile is ignored. Pass a null string ("") in this parameter.

nOp

Specifies the type of copy operation to perform. Pass one of the following predefined constants in this parameter:

COMP_NORMAL

Copies files to the target system, updating existing same-named files regardless of date, time, or version information.

COMP_UPDATE_SAME

Update the files even if the date, time, or version of the source file is identical to the target file. You must also specify either COMP_UPDATE_DATE or COMP_UPDATE_VERSION with this constant. Otherwise, InstallShield ignores this constant.

COMP_UPDATE_DATE

Updates the files based on the file date and time. This constant updates the file if the source file is newer than the target file.

COMP_UPDATE_VERSION

Updates the files based on the file version. This constant updates the file if the source file is newer than the target file. If the file version does not exist in both the source and the target files, date and time are used for comparison. If the file version does not exist for only one file, InstallShield assumes that the file containing version information is the newer file.

SELFREGISTER

Carries out the self-registration process immediately, when using the "non-batch method" of installing self-registering files.

If you have called [Enable\(SELFREGISTERBATCH\)](#), this option queues up [self-registering files](#) for registration. The files are registered once [Do\(SELFREGISTRATIONPROCESS\)](#) is called, when using the "batch method" of installing self-registering files.

Always use SELFREGISTER together with the SHAREDFILE option, combining them with the bitwise OR operator (|).

SHAREDFILE

Combines shared and locked file handling by causing XCopyFile to treat all files as shared, and to record locked .dll and .exe files for update when Windows or the system restarts. See RebootDialog and SdFinishReboot.

Under Windows 95 and Windows NT, the SHAREDFILE option causes XCopyFile to treat all files as shared files and increment the registry reference counter by one when the file exists in the target directory and it has a reference count greater than 0. If the shared file does not exist in the target directory and it has no reference counter, InstallShield creates the counter and sets it to 1. If the shared file already exists in the target directory but has no reference counter, InstallShield creates the counter and initializes it to 2 as a precaution against accidental removal during uninstallation.

Under Windows 3.1, there is no registry reference counter, so no files updated using the SHAREDFILE option are logged for uninstallation.

LOCKEDFILE

Causes XCopyFile to record locked .dll and .exe files for update when Windows or the system is rebooted. A locked file is a file that is in use by an application or the system when InstallShield attempts to access or update the file. The LOCKEDFILE option works like SHAREDFILE except that LOCKEDFILE does not create registry entries or modify the registry reference counter. You cannot use the LOCKEDFILE option when using the SHAREDFILE option. There are some unshared files (such as shell extensions) for which the script writer does not want a registry entry and reference counter. These files should never be uninstalled, except by the application itself. LOCKEDFILE allows XCopyFile to handle locked files that are not shared.

EXCLUDE_SUBDIR

Specifies not to include subdirectories contained in the source path.

INCLUDE_SUBDIR

Specifies that subdirectories below the source path must also be copied.

Return values

0

Indicates that the function successfully copied the files.

< 0

Indicates that the function was unable to copy the files due to one of the following conditions:

COPY_ERR_CREATEDIR

Indicates a target directory could not be created. Make sure that the path in the system variable TARGETDIR is syntactically correct and you have access rights to the target drive.

COPY_ERR_MEMORY

Indicates that the function was unable to allocate the memory required to complete the copy file process. Terminate as many running applications as possible to free memory.

COPY_ERR_NODISKSPACE

Indicates that the function could not find enough disk space on the target drive to copy the files. Free disk space on the target drive.

COPY_ERR_OPENINPUT

Indicates that the function was unable to open the input file in the system variable SRCDIR. Make sure the source file is a valid filename, and that the source file as well as the target directory exist.

COPY_ERR_OPENOUTPUT

Indicates that the function was unable to copy the requested file.

COPY_ERR_TARGETREADONLY

Indicates that the file in TARGETDIR is read-only. Remove the read-only attribute from the target file and try again.

-51

A self-registering file did not register successfully.

All other negative values

Indicates some other unspecified error has occurred.

Comments

- n Since Windows 95 does not allow an empty file to be copied and Windows NT does not allow creation of empty files, XCopyFile will not work under these platforms when used to copy empty files (Size = 0 KB).
- n After modifying .ini files with WriteProfString, you will need to flush the cache buffer on Windows 95 before using XCopyFile. All .ini files are cached on Windows 95, which can cause a delay in writing changes to the specified files. This in turn can interfere with subsequent file operations. To avoid this problem, simply call WriteProfString with null parameters to force Windows 95 to write the data to the .ini file immediately:

```
WriteProfString ("c:\\test.ini", "Windows", "KeyboardDelay", "100");
WriteProfString ("", "", "", ""); //null string ("") for all four parameters
TARGETDIR = "c:\\temp";
//XCopyFile should now have access to updated file.
XCopyFile ("test.ini", "test.ini", INCLUDE_SUBDIR);
```

{button ,JI('LANGREF.HLP>Examples','XCopyFile_example')} [Example](#)

{button ,AL('CopyFile;VarRestore;VarSave;Installing shared files with XCopyFile and VerUpdateFile;Installing locked files with XCopyFile and VerUpdateFile;Installing self-registering files with XCopyFile and VerUpdateFile',0,','')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



XCopyFile example

```
/*-----*\
*
* This example illustrates the XCopyFile function.
*
* The first call to XCopyFile copies the readme files regardless of date,
* time, or version.
*
* The second call copies program files and creates the subdirectories that
* these files need to be located in.
*
* The third call copies template files based upon the date, writing over
* target files that have the same or earlier date as the source files.
*
* The fourth call copies sample files based upon the version number, writing
* over target files that have an older version number.
*
* NOTE: In order for this script to run properly, it is necessary for the
*       preprocessor constants to be set to a valid filename and path on the
*       target system.
*
\*-----*/

#define SDIR          "C:\\EXAMPLE\\SOURCE\\"
#define SDIR_PROGRAM "C:\\EXAMPLE\\SOURCE\\PROGRAM\\"
#define SDIR_TEMPLATE "C:\\EXAMPLE\\SOURCE\\TEMPLATE\\"
#define SDIR_SAMPLES "C:\\EXAMPLE\\SOURCE\\SAMPLES\\"
#define TDIR          "C:\\EXAMPLE\\TARGET\\"

    STRING szSrcFile;
    NUMBER nResult, nOp;

program

    // Set source and target directories.
    SRCDIR    = SDIR;
    TARGETDIR = TDIR;

    // Set variables.
    szSrcFile = "*.TXT";
    nOp = COMP_NORMAL;

/*-----*\
*
* XCopyFile is called for the first time copying all readme files in the
* source directory into the target directory.
*
\*-----*/
    if (XCopyFile(szSrcFile, "*.*", nOp) < 0) then

        MessageBox("XCopyFile failed", SEVERE);
    else

        MessageBox("Text files successfully copied.", INFORMATION);
    endif;

    // Set new source and target directories.
    SRCDIR    = SDIR_PROGRAM;
    TARGETDIR = TDIR ^ "PROGRAM";
```

```

// Set new variables.
szSrcFile = "*..*";
nOp = INCLUDE_SUBDIR;

/*-----*\
*
* XCopyFile is called for the second time copying all program files within
* a source subdirectory to a subdirectory of the target directory.
*
\*-----*/
if (XCopyFile(szSrcFile, "*..*", nOp) < 0) then
    MessageBox("XCopyFile failed", SEVERE);
else
    MessageBox("Program files successfully copied.", INFORMATION);
endif;

SRCDIR    = SDIR_TEMPLATE;
TARGETDIR = TDIR ^ "TEMPLATE";

nOp = COMP_UPDATE_SAME | COMP_UPDATE_DATE;

/*-----*\
*
* XCopyFile is called for the third time copying all template files within
* a source subdirectory to a subdirectory of the target directory.
*
\*-----*/

if (XCopyFile(szSrcFile, "*..*", nOp) < 0) then
    MessageBox("XCopyFile failed", SEVERE);
else
    MessageBox("Template files successfully copied.", INFORMATION);
endif;

SRCDIR    = SDIR_SAMPLES;
TARGETDIR = TDIR ^ "SAMPLES";

nOp = COMP_UPDATE_VERSION;

/*-----*\
*
* XCopyFile is called for the fourth time copying all sample files within
* a source subdirectory to a subdirectory of the target directory.
*
\*-----*/

if (XCopyFile(szSrcFile, "*..*", nOp) < 0) then
    MessageBox("XCopyFile failed", SEVERE);
else
    MessageBox("Sample files successfully copied.", INFORMATION);
endif;

endprogram

// Source file: Is5fn275.rul

```

{button ,PI("",`comment')} [Feedback](#)



Information functions

The following Information functions provide data about resources that are available in the operating environment: disk space, memory, and operating mode.

GetDiskSpace

Returns the amount of free space on a disk.

GetEnvVar

Returns the current value of an environment variable.

GetExtents

Returns the dimensions of the screen.

GetMemFree

Returns the amount of memory that is available to an application running under Microsoft Windows.

GetMode

Returns the mode in which Microsoft Windows is currently running.

GetSystemInfo

Retrieves system information.

GetValidDrivesList

Returns a listing of all available drives on the target system.

GetWindowHandle

Returns the handle of the main installation window.

Is

Provides file and path checking services, searches for a math coprocessor, tests for administrator status under Windows NT, and determines whether Microsoft Windows is running from a shared copy on a network.

{button ,PI("",`comment')} [Feedback](#)



GetDiskSpace

Syntax

GetDiskSpace (szDrive);

Description

Use the GetDiskSpace function to determine how much free disk space is available on a particular drive.

Parameters

szDrive

Specifies the drive designation (that is, a drive letter followed by a colon) of the disk to check for free space.

Return values

XXXX

The number of bytes free on the specified drive. The maximum value returned is 2 GB. Free disk space greater than returns as 2 GB.

< 0

Indicates that GetDiskSpace was unable to obtain the amount of free disk space.

Comment

This function does not support UNC paths. You must pass a disk drive designation in szDrive.

{button ,JI(`LANGREF.HLP>Examples`,`GetDiskSpace_example`)} [Example](#)

{button ,AL(`GetDisk;GetSystemInfo',0,`,`')} [See also](#)

{button ,PI("",`comment')} [Feedback](#)



GetDiskSpace example

```
/*-----*\
 *
 * This example illustrates the usage of the GetDiskSpace function.
 *
 * This script calls GetDiskSpace to retrieve the hard drive space on the
 * target system.
 *
 \*-----*/

    STRING szTitle, szMsg, szPath, svResultPath, szDrive, svDisk;
    NUMBER nResult;

program

start:

    szTitle = "GetDiskSpace Example";

    szMsg = "Which drive letter would you like GetDiskSpace to identify?";
    szPath = "C:\\";

    // Prompt the user for a target path.
    AskPath(szMsg, szPath, svResultPath);

    // GetDisk is called to retrieve the letter drive of the path.
    GetDisk(svResultPath, svDisk);

    szDrive = svDisk;
    nResult = GetDiskSpace(szDrive);

    if (nResult < 0) then

        MessageBox("GetDiskSpace failed.", SEVERE);
    else

        szMsg = "There are %d bytes free on %s.";
        sprintfBox(INFORMATION, szTitle, szMsg, nResult, szDrive);
    endif;

endprogram

// Source file: Is5fn221.rul
```

{button ,PI("",`comment')} [Feedback](#)



GetEnvVar

Syntax

GetEnvVar (szParameter, svValue);

Description

The GetEnvVar function retrieves the current value of an environment variable.

Parameters

szParameter

Specifies the name of the environment variable whose value is to be retrieved.

svValue

Returns the current value of the environment variable.

Return values

0

Indicates that the function retrieved the value of the environment variable.

< 0

Indicates that the function was unable to retrieve the value of the environment variable.

Comments

InstallShield does not provide a mechanism for changing the value of an environment variable. Under the Microsoft Windows environment, it is not recommended that you change the value of an environment variable. If you need to set an environment variable, set it in the Autoexec.bat file and then reboot the system.

{button ,JI('LANGREF.HLP>Examples','GetEnvVar_example')} [Example](#)

{button ,AL('Ez batch file fn;Advanced batch file fn',0,'')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



GetEnvVar example

```
/*-----*\
 *
 * This example illustrate the usage of the GetEnvVar function.
 *
 \*-----*/

    STRING szParameter, svValue, szTitle;

program

/*-----*\
 *
 * GetEnvVar is called to return the contents of the TEMP environment variable
 * to svValue.
 *
 \*-----*/

    szParameter = "TEMP";
    if (GetEnvVar(szParameter, svValue) < 0) then

        MessageBox("First call to GetEnvVar failed.", SEVERE);
    endif;
    szTitle = "GetEnvVar Example";
    sprintfBox(INFORMATION, szTitle, "%s = %s", szParameter, svValue);

/*-----*\
 *
 * GetEnvVar is called to return the contents of the PATH environment variable
 * to svValue.
 *
 \*-----*/

    szParameter = "PATH";
    if (GetEnvVar(szParameter, svValue) < 0) then

        MessageBox("Second call to GetEnvVar failed.", SEVERE);
    endif;
    sprintfBox(INFORMATION, szTitle, "%s = %s", szParameter, svValue);

endprogram

// Source file: Is5fn222.rul
```

{button ,PI("",`comment')} [Feedback](#)



GetExtents

Syntax

```
GetExtents (nvDx, nvDy);
```

Description

The GetExtents function retrieves the dimensions of a screen. The width of the screen, in pixels, is returned in nvDx; and the height, in pixels, is returned in nvDy. For example, a standard VGA monitor returns 640 in nvDx and 480 in nvDy.

Parameters

nvDx

Returns the width of the screen in pixels.

nvDy

Returns the height of the screen in pixels.

Return values

0

Indicates that the function successfully retrieved the dimensions of a screen.

< 0

Indicates that the function was unable to retrieve the values.

{button ,JI('LANGREF.HLP>Examples','GetExtents_example')} [Example](#)

{button ,AL('GetSystemInfo;PlaceWindow;SizeWindow;Using metafiles as images;Place bitmaps according to target screen resolution',0,','')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



GetExtents example

```
/*-----*\
 *
 * This example illustrates the usage of the GetExtents function.
 *
 * GetExtents is called to determine whether the target system's screen
 * resolution is VGA or greater.
 *
 \*-----*/

    NUMBER  nvDx, nvDy;

program

    if (GetExtents (nvDx, nvDy) < 0) then

        MessageBox("GetExtents failed.", SEVERE);
    endif;

    if (nvDy < 480) then

        MessageBox("This system is not running in resolution VGA or greater",
                    INFORMATION);
    else

        MessageBox("This system is running in VGA or greater resolution.",
                    INFORMATION);
    endif;

endprogram

// Source file: Is5fn223.rul
```

{button ,PI("",`comment')} [Feedback](#)



GetMemFree

Syntax

```
GetMemFree ( );
```

Description

The GetMemFree function returns the amount of memory that is available to an application running under Microsoft Windows. Because Microsoft Windows is a virtual memory system, this function does not return the actual physical memory (called RAM), but the memory available to a Windows application. To determine the amount of actual physical memory available on the target system, call [GetSystemInfo](#).

Parameters

This function has no parameters. After you type GetMemFree, type the open and close parentheses with nothing inside them, as shown below:

```
GetMemFree ( );
```

Return values

XXXX

Where XXXX is the amount of free memory in bytes available to the application.

< 0

GetMemFree was unable to return the amount of free memory.

Comments

Each time the script executes a function, InstallShield returns a value indicating the result of that function. If you plan to use the return value from this function later in the script, assign the value to a numeric variable.

```
{button ,JI('LANGREF.HLP>Examples','GetMemFree_example')} Example
```

```
{button ,AL('GetSystemInfo',0,'')} See also
```

```
{button ,PI('','comment')} Feedback
```



GetMemFree example

```
/*-----*\
 *
 * This example illustrates the usage of the GetMemFree function.
 *
 * GetMemFree checks the amount of free memory the system has.
 *
 \*-----*/

    STRING szTitle, szMsg;
    NUMBER nResult;

program

/*-----*\
 *
 * Retrieve the amount of free memory and set it to the nResult variable.
 *
 \*-----*/
    nResult = GetMemFree();

    if (nResult < 0) then

        MessageBox("GetMemFree failed.", SEVERE);
    else

        szTitle = "GetMemFree Example";
        szMsg    = "There are %i bytes of free memory";

        // Display the amount of free memory.
        sprintfBox(INFORMATION, szTitle, szMsg, nResult);
    endif;

endprogram

// Source file: Is5fn083.rul
```

{button ,PI("",`comment')} [Feedback](#)



GetMode

Syntax

```
GetMode ( );
```

Description

The GetMode function determines if Microsoft Windows is running in standard mode or enhanced mode. To retrieve the version of Windows running on the target system, call [GetSystemInfo](#).

Parameters

GetMode has no parameters. After you type GetMode, type the open and closed parentheses with nothing inside them, as shown below:

```
GetMode ( );
```

Return values

STANDARD

Windows is running in Standard mode.

ENHANCED

Windows is running in 386 Enhanced mode.

{button ,JI('LANGREF.HLP>Examples','GetMode_example')} [Example](#)

{button ,AL('GetSystemInfo',0,'')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



GetMode Example

```
/*-----*\
 *
 * This example illustrate the usage of the GetMode function.
 *
 * GetMode is called to check the mode of Windows on the target system.
 *
 \*-----*/

program

/*-----*\
 *
 * Find the mode that Windows is running under.
 *
 \*-----*/
    if (GetMode () = STANDARD) then
        MessageBox("Standard mode Windows in operation.", INFORMATION);
    else
        MessageBox("Enhanced mode Windows in operation.", INFORMATION);
    endif;

endprogram

// Source file: Is5fn084.rul
```

{button ,PI("",`comment')} Feedback



GetSystemInfo

Syntax

GetSystemInfo (nItem, nvResult, svResult);

Description

The GetSystemInfo function retrieves information about the target system.

Parameters

The first parameter, nItem, is used to specify the type of information to retrieve. Consult the table below for a list of constants you can pass in this parameter to retrieve system information. Note that when using certain constants (such as DISK_TOTALSPACE), you must specify additional information in the parameter svResult *before* calling the function.

System information is returned in nvResult and/or svResult. Numeric data is returned in nvResult. String data is returned in svResult. The table below shows which type of data is returned for each of the constants you can pass in nItem.

nItem:	BASEMEMORY
nvResult:	For 16-bit installations, this parameter returns the base memory on the target system. This parameter should not be used with 32-bit installations; instead use the EXTENDEDMEMORY parameter to get the total physical memory of the system.
svResult:	N/A
nItem:	BOOTUPDRIVE
nvResult:	The ID of the bootup drive, where 1= A:, 2 = B:, 3 = C:.. It is possible to convert this number to the appropriate drive letter by adding 64 (DECIMAL) to the value and then setting a string variable to this value. Use the following syntax to convert: <code>svResult[0]=64+nvResult;</code>
svResult:	N/A
nItem:	CDROM
nvResult:	TRUE or FALSE Indicates whether a CD ROM is available.
svResult:	N/A
nItem:	COLORS
nvResult:	Returns the number of colors available on the user's system. The result is retrieved from the video driver on the target system, rather than from the monitor card. If the card can support 256 colors but the driver can handle only 16 colors, the number of colors returned is 16.
svResult:	N/A
nItem:	CPU
nvResult:	One of the following constants will be returned: IS_UNKNOWN - The user's CPU is unknown. IS_286 - The user has a 286 processor. IS_386 - The user has a 386 processor. IS_486 - The user has a 486 processor. IS_PENTIUM - The user has a PENTIUM processor. IS_ALPHA - The user has an ALPHA processor. IS_MIPS - The user has an MIPS processor. IS_POWERPC - The user has a PowerPC processor.
svResult:	N/A
nItem:	DATE
nvResult:	N/A
svResult:	The current system date in MM-DD-YYYY format.
nItem:	DISK_TOTALSPACE
nvResult:	Returns the total capacity of the disk drive specified in svResult. The maximum value returned is 2 GB. Free disk space greater than that still returns as 2 GB.
svResult:	The letter of the drive. Note that this parameter is passed <i>to</i> the function. Also note that you must

include the colon (:) after the drive letter; otherwise the function will fail.

nItem: DRIVE
nvResult: Returns the type of the drive specified in svResult. One of the following constants will be returned:
IS_UNKNOWN - Target drive is unknown.
IS_REMOVABLE - Target drive is a floppy drive.
IS_FIXED - Target drive is a fixed drive.
IS_CDROM - Target drive is a CD-ROM drive.
IS_REMOTE - Target drive is a network drive.
svResult: The letter of the drive followed by a colon (:). Note this parameter is passed to the function.

nItem: ENVSPACE
nvResult: Retrieves the size of the environment space from the Config.sys file statement SHELL=C:\Dos\Command.com /P/E:xxxx, where xxxx is an environment space value.
svResult: N/A

nItem: EXTENDEDMEMORY
nvResult: For 16-bit installations, returns the amount of extended memory installed. Due to operating system limitations a 16-bit installation running on 32-bit platform will not be able to determine correctly the amount of extended memory installed; therefore 16-bit installations should use this parameter only when running on a 16-bit platform. For 32-bit installations, returns the *total* amount of memory installed on the machine. Due to operating system limitations, the value returned may be slightly different than the actual amount of physical memory installed on the system. This value will normally be within 100K (1 MB) of the actual value.
svResult: N/A

nItem: FREEENVSPACE
nvResult: Returns the amount of environment space free at the time GetSystemInfo is called.
svResult: N/A

nItem: ISTYPE
nvResult: The InstallShield type running the script, where 16 indicates 16-bit InstallShield, and 32 indicates 32-bit InstallShield.
svResult: N/A

nItem: LANGUAGE
nvResult: The InstallShield [language constant](#) for the target system will be returned in this parameter. The returned constant can be used to determine which language specific file groups to install during the setup using the [ComponentFilterLanguage](#) function.



InstallShield International supports 21 languages, while Windows supports over 100 languages. If you intend to filter file groups based on the value of nvResult, you must use a switch statement to determine the InstallShield language identifier constant to use based on the constant returned by this function. The behavior of this function with this parameter is highly system dependent. For information about determining the default language of the target system, click [here](#).

svResult: The equivalent language name string for the language constant returned in nvResult is returned in this parameter.

nItem: LANGUAGE_DRV
nvResult: N/A
svResult: The name of the dynamic-link library that supplies the language-specific functions for Windows, retrieved from the language.dll key in the [boot] section of the System.ini file. This key will have a value only on non-English versions of 16-bit Windows or English versions of 16-bit Windows in which the user has changed the default language setting using the International Icon of the Windows control panel. On all other platforms, including all 32-bit platforms and English versions of 16-bit Windows that have English (American) set as the current language (the default), a null string ("") will be returned in this parameter.

nItem: MOUSE
nvResult: N/A
svResult: The name of the installed mouse from System.ini.

nItem: MOUSE_DRV

nvResult:	N/A
svResult:	The name of the installed mouse driver from System.ini.
nItem:	NETWORK
nvResult:	N/A
svResult:	The name of the network from System.ini.
nItem:	NETWORK_DRV
nvResult:	N/A
svResult:	The name of the network driver from System.ini.
nItem:	OS
nvResult:	Returns the platform of the target operating system. One of the following constants will be returned: IS_WINDOWS - Operating system is Windows 3.1. IS_WINDOWSNT - Operating system is Windows NT. IS_WINDOWS95 - Operating system is Windows 95.
svResult:	N/A
nItem:	OSMAJOR
nvResult:	Returns the major version of the operating system.
svResult:	A string in the format ##.## for the major and minor versions of the operating system.
nItem:	OSMINOR
nvResult:	Returns the minor version of the operating system.
svResult:	A string in the format ##.## for the major and minor versions of the operating system.
nItem:	PARALLEL
nvResult:	Returns the number of physical parallel ports available.
svResult:	N/A
nItem:	SERIAL
nvResult:	Returns the number of physical serial ports available.
svResult:	N/A
nItem:	SHARE
nvResult:	TRUE or FALSE, indicating whether or not file sharing is enabled on the target system. Both the normal (Share.exe) driver and the virtual-mode (VShare.386) driver can be detected by using this parameter. Note that because file sharing is automatic for all 32-bit operating systems, this parameter will always return TRUE on 32-bit platforms.
svResult:	N/A
nItem:	TIME
nvResult:	N/A
svResult:	Returns current system time in HH:MM:SS format.
nItem:	VIDEO
nvResult:	Returns the type of video adapter installed. (InstallShield cannot detect CGA or monochrome video drivers.) One of the following constants will be returned: IS_UNKNOWN - The user's video is unknown. IS_EGA - EGA resolution. IS_VGA - VGA resolution. IS_SVGA - Super VGA (800 x 600) resolution. IS_XVGA - XVGA (1024 x 768) resolution. IS_UVGA - Greater than 1024 x 768 resolution.
svResult:	N/A
nItem:	VOLUMELABEL
nvResult:	Returns the volume label.
svResult:	This parameter must be set to the drive letter, including the colon, for which you are trying to determine the volume label. The volume label of the specified drive is then returned in this parameter after calling the function. If the drive has no volume label, it returns a null string ("").
nItem:	WINMAJOR
nvResult:	32-bit setups return the major version of Windows. Due to operating system limitations, 16-bit setups always return the value 3 in this parameter, regardless of which version of Windows is running.
svResult:	32-bit setups return a string in the format ##.##, which indicates the major and minor versions of Windows. 16-bit setups running under Windows 95 return "3.95". Due to operating system limitations, 16-bit setups always return "3.10" in this parameter when running under any other version of Windows.
nItem:	WINMINOR

nvResult:	32-bit setups return the minor version of Windows. 16-bit setups running under Windows 95 return the value 95. Due to operating system limitations, 16-bit setups always return the value 10 in this parameter when running under any other version of Windows.
svResult:	32-bit setups return a string in the format ##.## , which indicates the major and minor versions of Windows. 16-bit setups running under Windows 95 return "3.95". Due to operating system limitations, 16-bit setups always return "3.10" in this parameter when running under any other version of Windows.
nItem:	WIN32SINSTALLED
nvResult:	Returns a value of TRUE if any version of Win32s is installed. FALSE if Win32s is not installed.
svResult:	N/A
nItem:	WIN32SMAJOR
nvResult:	Returns one of the following values indicating the major version of Win32s: 1 for Win32s 1.00, 1.15, 1.20, 1.25, and 1.30 2 for Win32s 2.x 0 if Win32s is not installed. Use the WIN32SINSTALLED parameter to determine if Win32s is installed before using this parameter.
svResult:	N/A
nItem:	WIN32SMINOR
nvResult:	Returns one of the following values indicating the minor version of Win32s: 15 for Win32s 1.15 20 for Win32s 1.20 25 for Win32s 1.25 30 for Win32s 1.30 0 for Win32s 1.00 Use the WIN32SINSTALLED parameter to determine if Win32s is installed before using this parameter. To get the build number of the Win32s version, it is necessary to call the windows API function, GetWin32sInfo.
svResult:	N/A

Return values

0

Indicates that the function successfully returned the specified information.

< 0

Indicates that the function was unable to return the specified information.

Compatibility

The following chart shows which types of information, defined by the nItem constant, are accessible using GetSystemInfo on different operating systems. An **X** indicates that the information is accessible on the operating system(s) shown at the head of the column. ***X** indicates that the information is accessible, but as of this writing nItem does not successfully retrieve the information. **N/A** indicates information that is not accessible or not relevant on a given operating system.

nItem Constant	Windows NT	Windows 95	Windows 3.1
BASEMEMORY	X	X	X
BOOTUPDRIVE	X	X	X
CDROM	X	X	X
COLORS	X	X	X
CPU	X	X	*X
DATE	X	X	X
DISK_TOTALSPACE	X	X	X
DRIVE	X	X	X
ENVSPACE	N/A	N/A	X
EXTENDEDMEMORY	X	X	X

FREEENVSPACE	N/A	N/A	X
ISTYPE	X	X	X
LANGUAGE	X	*X	X
LANGUAGE_DRV	*X	*X	*X
MOUSE	*X	*X	X
MOUSE_DRV	*X	*X	X
NETWORK	X	*X	X
NETWORK_DRV	X	*X	X
OS	X	X	X
OSMAJOR	N/A	N/A	X
OSMINOR	N/A	N/A	X
PARALLEL	*X	*X	X
SERIAL	*X	X	X
SHARE	N/A	N/A	X
TIME	X	X	X
VIDEO	X	X	X
VOLUMELABEL	X	X	X
WINMAJOR	X	X	X
WINMINOR	X	X	X

Comments

The PARALLEL and SERIAL options are not supported when running a 32-bit setup under Win32s.

{button ,JI('LANGREF.HLP>Examples','GetSystemInfo_example')} [Example](#)

{button ,AL('GetDir;GetDisk;GetFileInfo;Prepare my setup for uninstallation',0,'')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



GetSystemInfo example

```
/*-----*\
 *
 * This example illustrates some of the many uses of the GetSystemInfo
 * function.
 *
 * This script uses many of the constants available for GetSystemInfo
 * and tests for all possible return values. After retrieving the
 * information, the results are displayed in a message box.
 *
 *-----*/

#include "Sddialog.h"

STRING  szTitle, szMsg, svResult, szInfo;
NUMBER  nvResult;
LIST    listInfo;

program

listInfo = ListCreate(STRINGLIST);

if (GetSystemInfo (BASEMEMORY, nvResult, svResult) < 0) then
    szInfo = "Couldn't get BASEMEMORY info.";
else
    Sprintf(szInfo, "Your machine has %d K base memory.", nvResult);
endif;

ListAddString(listInfo, szInfo, AFTER);

if (GetSystemInfo (BOOTUPDRIVE, nvResult, svResult) < 0) then
    szInfo = "Couldn't get BOOTUPDRIVE info.";
else
    Sprintf(szInfo, "Boot drive: %d.", nvResult);
endif;

ListAddString(listInfo, szInfo, AFTER);

if (GetSystemInfo (CDROM, nvResult, svResult) < 0) then
    szInfo = "Couldn't get CD-ROM info.";
else
    Sprintf(szInfo, "CDROM: %d.", nvResult);
endif;

ListAddString(listInfo, szInfo, AFTER);

if (GetSystemInfo (COLORS, nvResult, svResult) < 0) then
    szInfo = "Couldn't get COLORS info.";
else
    Sprintf(szInfo, "Number of colors: %d.", nvResult);
endif;

ListAddString(listInfo, szInfo, AFTER);

if (GetSystemInfo (DATE, nvResult, svResult) < 0) then
    szInfo = "Couldn't get DATE info.";
else
    Sprintf(szInfo, "DATE: %s.", svResult);
endif;
```

```

ListAddString(listInfo, szInfo, AFTER);

if (GetSystemInfo (EXTENDEDMEMORY, nvResult, svResult) < 0) then
    szInfo = "Couldn't get EXTENDEDMEMORY info.";
else
    Sprintf(szInfo, "Extended Mem size: %d bytes", nvResult);
endif;

ListAddString(listInfo, szInfo, AFTER);

if (GetSystemInfo (ISTYPE, nvResult, svResult) < 0) then
    szInfo = "Couldn't get ISTYPE info.";
else
    Sprintf(szInfo, "InstallShield Type: %d-bit", nvResult);
endif;

ListAddString(listInfo, szInfo, AFTER);

if (GetSystemInfo (OS, nvResult, svResult) < 0) then
    szInfo = "Couldn't get Operating System info.";
else
    switch (nvResult)
        case IS_WINDOWS:
            szInfo = "OS: Windows 3.1";
        case IS_WINDOWSNT:
            szInfo = "OS: Windows NT";
        case IS_WINDOWS95:
            szInfo = "OS: Windows 95";
        case IS_WIN32S:
            szInfo = "OS: Win32s";
    endswitch;
endif;

ListAddString(listInfo, szInfo, AFTER);

if (GetSystemInfo (TIME, nvResult, svResult) < 0) then
    szInfo = "Couldn't get TIME info.";
else
    Sprintf(szInfo, "TIME: %s.", svResult);
endif;

ListAddString(listInfo, szInfo, AFTER);

if (GetSystemInfo (VIDEO, nvResult, svResult) < 0) then
    szInfo = "Couldn't get VIDEO info.";
else
    switch (nvResult)
        case IS_UNKNOWN:
            szInfo = "VIDEO: UNKNOWN";
        case IS_SVGA:
            szInfo = "VIDEO: SVGA";
        case IS_XVGA:
            szInfo = "VIDEO: XVGA";
        case IS_UVGA:
            szInfo = "VIDEO: UVGA";
    endswitch;
endif;

ListAddString(listInfo, szInfo, AFTER);

szTitle = "System Information";
szMsg = "The following is some information related to your system:\n";

```

```
SdShowInfoList (szTitle, szMsg, listInfo);  
endprogram  
#include "Sddialog.rul"  
// Source file: Is5fn226.rul
```

{button ,PI("",`comment')} Feedback



GetValidDrivesList

Syntax

```
GetValidDrivesList (listID, nDriveType, nMinDriveSpace);
```

Description

The GetValidDrivesList function retrieves a list of all the drives attached to the target system that meet a certain criterion. This criterion includes the type of drive and the minimum amount of space on the drive. If a drive door is open, the drive name is still inserted into the list.

You can specify the type of drive to search for (nDriveType) and the minimum amount of disk space that must be available before the drive is listed (nMinDriveSpace).

Parameters

listID

Returns a list of valid drive letters.

nDriveType

Specifies the type of drive to search for. Pass one of the following predefined constants in this parameter:

FIXED_DRIVE

Searches only for fixed drives.

REMOTE_DRIVE

Searches only for remote drives. Remote drives are generally located on a network.

REMOVEABLE_DRIVE

Searches only for removable drives. Floppy drives are removable drives.

CDROM_DRIVE

Searches only for CD-ROM drives.

nMinDriveSpace

The minimum amount of disk space in bytes that must be free on the drive to allow the drive to be included in the return list. If nMinDriveSpace is less than zero, GetValidDrivesList will not check for the minimum space on the drive. This is useful for floppy drives.

Return values

0

GetValidDrivesList successfully retrieved the requested list.

< 0

GetValidDrivesList was unable to retrieve the list.

Comments

Network mapping drives may also be returned as remote drives. This function may not return all drives on the network. Only those drives designated as mapping drives are returned.

{button ,JI('LANGREF.HLP>Examples','GetValidDrivesList_example')} [Example](#)

{button ,AL('GetDisk;GetDiskSpace',0,'')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



GetValidDrivesList example

```
/*-----*\
 *
 * This example illustrates the usage of the GetValidDrivesList function.
 *
 * This function is called once to return a list that contains any removeable
 * drive with a minimum of 120,000 free bytes.
 *
 * It is called again to return a list that contain all fixed drives with a
 * minimum of 1,000,000 bytes free.
 *
 \*-----*/

#include "Sddialog.h"

STRING szTitle, szMsg;
NUMBER nDriveType, nDriveMinSpace
LIST listID;

program

// Set GetValidDrivesList variables.
listID = ListCreate(STRINGLIST);
nDriveType = REMOVEABLE_DRIVE;
nDriveMinSpace = 120000;

// Set SdShowInfoList variables.
szTitle = "GetValidDrivesList Example";
szMsg = "Removable drives with 120,000 bytes free:";

/*-----*\
 *
 * The following adds to the newly created list the drive names that are
 * removable and contain at least 120,000 free bytes.
 *
 \*-----*/
if (GetValidDrivesList(listID, nDriveType, nDriveMinSpace) < 0) then

// Error Check.
MessageBox("GetValidDrivesList failed.", SEVERE);
abort;
else

// Display the results in a dialog box.
SdShowInfoList(szTitle, szMsg, listID);
endif;

// Destroy list.
ListDestroy(listID);

// Create new list.
listID = ListCreate(STRINGLIST);

nDriveType = FIXED_DRIVE;
nDriveMinSpace = 1000000;

szMsg = "Fixed drives with 1,000,000 bytes free:";
/*-----*\
 *
 * The following adds drive names that are fixed and contain at least
```

```
* 120,000 free bytes to the list.
*
\*-----*/
  if (GetValidDrivesList(listID, nDriveType, nDriveMinSpace) < 0) then

    MessageBox("GetValidDrivesList failed.", SEVERE);
    abort;
  else

    SdShowInfoList(szTitle, szMsg, listID);
  endif;

endprogram

#define SD_SINGLE_DIALOGS 1
#define SD_SHOWINFOLIST 1

#include "Sddialog.rul"

// Source file: Is5fn087.rul
```

{button ,PI("",`comment')} [Feedback](#)



GetWindowHandle

Syntax

GetWindowHandle (nHwndFlag);

Description

The GetWindowHandle function gets the handle of the main window of the setup.

Parameters

nHwndFlag

Specifies the window handle of InstallShield's main window. Pass the predefined constant HWND_INSTALL in this parameter.

Return values

X

Where X is the handle of the window.

1

Indicates that the handle is equal to HWND_DESKTOP.

< 0

GetWindowHandle was unable to retrieve the handle.

{button ,JI('LANGREF.HLP>Examples','GetWindowHandle_example')} [Example](#)

{button ,AL('FindWindow;GetFont',0,','')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



GetWindowHandle example

```
/*-----*\
 *
 * This example illustrates the usage of the GetWindowHandle function.
 *
 * This script retrieves and displays the handle of InstallShield's main
 * window.
 *
 \*-----*/

    NUMBER nHwndFlag;
    HWND hInstallHwnd;

program

    nHwndFlag = HWND_INSTALL;
/*-----*\
 *
 * GetWindowHandle retrieves the handle of the installation main window.
 *
 \*-----*/
    hInstallHwnd = GetWindowHandle(nHwndFlag);

    if (hInstallHwnd < 0) then

        MessageBox("GetWindowHandle failed.", SEVERE);
    else

        sprintfBox(INFORMATION, "INFO", "Window Handle: %i", hInstallHwnd);
    endif;

endprogram

// Source file: Is5fn088.rul
```

{button ,PI("",`comment')} [Feedback](#)



Is

Syntax

Is (nlsFlag, szlsData);

Description

The Is function retrieves information commonly needed in a script.

Parameters

nlsFlag

Specifies the type of information to retrieve. Pass one of the following predefined constants in this parameter.

DIR_WRITEABLE

Can I write to the directory specified in szlsData?

FILE_EXISTS

Does the file specified in szlsData exist?

FILE_LOCKED

Is the file locked?

FILE_WRITEABLE

Can I write to the file specified in szlsData?

MATH_COPROCESSOR

Does a math coprocessor exist in the target system?

PATH_EXISTS

Does the path specified in szlsData exist?

USER_ADMINISTRATOR

When the target operating system is Windows NT, does the current user have administrator privileges?

VALID_PATH

Is the path specified in szlsData a legal path? This will not confirm the existence of a path, it will only check its syntax. You can use this constant when you retrieve path information from the user. The function will then check to see if the path information was entered correctly.

WINDOWS_SHARED

Is Microsoft Windows running a shared copy from a network?

szlsData

Specifies information that is dependent on the constant passed in nlsFlag, as shown below:

- n If nlsFlag is DIR_WRITEABLE, szlsData specifies the fully qualified path to be checked.
- n If nlsFlag is FILE_EXISTS, szlsData specifies the fully qualified filename.
- n If nlsFlag is FILE_LOCKED, szlsData specifies the fully qualified filename.
- n If nlsFlag is FILE_WRITEABLE, szlsData specifies the fully qualified filename.
- n If nlsFlag is MATH_COPROCESSOR, szlsData specifies szlsData is ignored.
- n If nlsFlag is PATH_EXISTS, szlsData specifies the fully qualified path.
- n If nlsFlag is USER_ADMINISTRATOR, szlsData is ignored.
- n If nlsFlag is VALID_PATH, szlsData specifies the fully qualified path.
- n If nlsFlag is WINDOWS_SHARED, szlsData is ignored.

Return values

TRUE

Indicates that the answer is true.

FALSE

Indicates that the answer is false.

< 0

The Is function was unable to answer the question.

Comments

The constant WINDOWS_SHARED is applicable to the Microsoft Windows version only. A shared copy of Microsoft Windows is installed on a network and has common files that are shared by many users.

{button ,JI('LANGREF.HLP>Examples','Is_example')} Example

{button ,AL('GetDisk;GetFileInfo;GetSystemInfo',0,'')} See also

{button ,PI('','comment')} Feedback



Is example

```
/*-----*\
*
* This example illustrates the usage of the Is function.
*
* The Is function is first called to check if a file is write protected. It
* is called a second time to check if the directory is write protected. This
* is useful for checking the write protection of directories on a network.
* The third time the Is function is called, the function checks if Windows
* is installed on a network server or if it is installed on the local system.
*
* NOTE: In order for this script to run properly, it is advised to set the
*       constant EXAMPLE_TXT to a valid file on the target system.
*
\*-----*/

#define EXAMPLE_TXT "EXAMPLE\\EXAMPLE.TXT"

    STRING szIsData, szTitle;
    NUMBER nResult, nIsFlag;

program

    nIsFlag = FILE_WRITEABLE;
    szIsData = EXAMPLE_TXT;
/*-----*\
*
* The Is function is called to check if the EXAMPLE_TXT file is
* write-protected or not.
*
\*-----*/
    nResult = Is(nIsFlag, szIsData);

    szTitle = "Is Example";

    if (nResult = TRUE) then

        sprintfBox(INFORMATION, szTitle, "%s is writeable.", szIsData);
    elseif (nResult = FALSE) then

        sprintfBox(INFORMATION, szTitle, "%s is not writeable.", szIsData);
    elseif (nResult < 0) then

        MessageBox("First call to Is failed.", SEVERE);
    endif;

    nIsFlag = DIR_WRITEABLE;
    szIsData = SRCDIR;
/*-----*\
*
* The Is function is called to check if the directory is write-protected or
* not.
*
\*-----*/
    nResult = Is(nIsFlag, szIsData);

    if (nResult = TRUE) then

        sprintfBox(INFORMATION, szTitle, "%s is writeable.", szIsData);
    elseif (nResult = FALSE) then
```

```

        SprintfBox(INFORMATION, szTitle, "%s is not writeable.", szIsData);
    elseif (nResult < 0) then

        MessageBox("Second call to Is failed.", SEVERE);
    endif;

    nIsFlag = WINDOWS_SHARED;
    szIsData = "";
/*-----*\
*
* The Is function is called to check if Windows is being shared on a network
* or not.
*
*\-----*/
    nResult = Is(nIsFlag, szIsData);

    if (nResult = TRUE) then

        MessageBox("Windows is shared.", INFORMATION);
    elseif (nResult = FALSE) then

        MessageBox("Windows is not shared.", INFORMATION);
    elseif (nResult < 0) then

        MessageBox("Third call to Is failed.", SEVERE);
    endif;

endprogram

// Source file: Is5fn091.rul

```

{button ,PI("",`comment')} Feedback



List processing functions

Lists are used to store related information such as strings or numbers. In InstallShield they are used to return information to you. For example, the [GetGroupNameList](#) function returns a list of all the group names that currently exist in the Program Manager. Other InstallShield functions require you to fill a list and then give that list to the function. Lists can also be used to store related information. Below are the functions to implement lists in a setup script.

There are two types of lists, string lists and number lists. Two sets of functions are provided to work with lists. Functions that end with "Item" work on number lists. Functions that end with "String" work on string lists. You cannot use number list functions on string lists and vice versa.

ListAddItem

Adds an item to a list.

ListAddString

Adds a string to a list.

ListCount

Returns the number of string or numeric elements in a specified list.

ListCreate

Creates a new string or number list.

ListCurrentItem

Returns the current item in a list.

ListCurrentString

Returns the current string in a list.

ListDeleteItem

Deletes the current item in a list.

ListDeleteString

Deletes the current string in a list.

ListDestroy

Destroys a list.

ListFindItem

Makes the specified item the current item in a numeric list.

ListFindString

Makes the specified item the current item in a string list.

ListGetFirstItem

Retrieves the first element from a number list.

ListGetFirstString

Retrieves the first string from a string list.

ListGetNextItem

Retrieves the element after the current element from a number list.

ListGetNextString

Retrieves the element after the current element from a string list.

ListReadFromFile

Reads a text file into a list.

ListSetCurrentItem

Sets the current element of a number list.

ListSetCurrentString

Sets the current element of a string list.

ListSetIndex

Uses an index to set the current element of a list.

ListWriteToFile

Writes a string list to a file.

{button ,PI("", `comment`)} Feedback



ListAddItem

Syntax

ListAddItem (listID, nItem, nPlacementFlag);

Description

The ListAddItem function adds a numeric element to a number list before or after the current element. To traverse a list, first call [ListGetFirstItem](#) to get the first element in the list; then call [ListGetNextItem](#) repeatedly until you reach the end of the list. To make a specific element in the list the current element, call [ListSetIndex](#).

Parameters

listID

Specifies the name of a number list. This list must already have been created by a call to [ListCreate](#).

nItem

Specifies the numeric element to add to the list.

nPlacementFlag

Specify where to put nItem relative to the current element. The new element will go either before or after the current element. Pass one of the following predefined constants in this parameter:

AFTER

Adds the new element after the current element in the list.

BEFORE

Adds the new element before the current element in the list.

Return values

0

ListAddItem successfully added the element to a number list.

< 0

ListAddItem was unable to add the element.

Comments

ListAddItem works only with number lists.

{button ,JI('LANGREF.HLP>Examples','ListAddItem_example')}

[Example](#)

{button ,AL('ListProcessing',0,'')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



ListAddItem example

```
/*-----*\
 *
 * This example illustrates the usage of the ListAddItem function.
 *
 * ListAddItem is called at first to add a number to the item list. It is
 * then called again to add another number after the first. This item is
 * also set as the current element. ListAddItem is called for the third
 * time to add a number before the current element.
 *
 *-----*/

#include "Sddialog.h";

STRING szTitle, szMsg;
NUMBER nItem, nvItem;
LIST listID;

program

    // Create the number list.
    listID = ListCreate(NUMBERLIST);

    if(listID = LIST_NULL) then

        MessageBox("Unable to create list.", SEVERE);
        abort;
    endif;

/*-----*\
 *
 * The following calls ListAddItem to add a number to the list.
 *
 *-----*/
    nItem = 1;
    if (ListAddItem(listID, nItem, AFTER) < 0) then

        MessageBox("First call to ListAddItem failed.", INFORMATION);
    endif;

/*-----*\
 *
 * The following calls ListAddItem to add a number to the list.
 *
 *-----*/
    nItem = 3;
    if (ListAddItem(listID, nItem, AFTER) < 0) then

        MessageBox("Second call to ListAddItem failed.", INFORMATION);
    endif;

/*-----*\
 *
 * The following calls ListAddItem to add a number before the current element.
 *
 *-----*/
    nItem = 2;
    if (ListAddItem(listID, nItem, BEFORE) < 0) then

        MessageBox("Third call to ListAddItem failed.", INFORMATION);
    endif;

```

```
endif;

szTitle = "ListAddItem Example";
szMsg   = "The following is a list of the items added:";

// Show the list of items.
SdShowInfoList(szTitle, szMsg, listID);

// Retrieve and display the current element.
ListCurrentItem(listID, nvItem);
SprintfBox(INFORMATION, szTitle, "Current Item: %d", nvItem);

// Remove the list from memory.
ListDestroy(listID);

endprogram

#define SD_SINGLE_DIALOGS 1
#define SD_SHOWINFOLIST  1

#include "Sddialog.rul"

// Source file: Is5fn094.rul
```

{button ,PI("",`comment')} [Feedback](#)



ListAddString

Syntax

ListAddString (listID, szString, nPlacementFlag);

Description

The ListAddString function adds a string to a string list before or after the current element. To traverse a list, first call [ListGetFirstItem](#) to get the first element in the list; then call [ListGetNextItem](#) repeatedly until you reach the end of the list. To make a specific element in the list the current element, call [ListSetIndex](#).

Parameters

listID

Specifies the name of a string list. This list must already have been created by a call to [ListCreate](#).

szString

Specifies the string to add to the list.

nPlacementFlag

Specifies where to put szString relative to the current element. The new string will go either before or after the current element. Pass one of the following predefined constants in this parameter:

AFTER

Adds the new string after the current element in the list.

BEFORE

Adds the new string before the current element in the list.

Return values

0

ListAddString successfully added the string to the list.

< 0

ListAddString was unable to add the string to the list.

Comments

ListAddString works only with string lists.

{button ,JI('LANGREF.HLP>Examples','ListAddString_example')} [Example](#)

{button ,AL('ListProcessing',0,'')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



ListAddString example

```
/*-----*\
 *
 * This example illustrates the usage of the ListAddString function.
 *
 * ListAddString is called at first to add a string to the string list. It is
 * then called again to add another string after the first. This string is
 * also set as the current element. ListAddString is called for the third
 * time to add a string before the current element.
 *
 *-----*/

#include "Sddialog.h";

STRING szTitle, szMsg, szString, svString;
LIST listID;

program

listID = ListCreate(STRINGLIST);

if (listID = LIST_NULL) then

    MessageBox("Unable to create list.", SEVERE);
    abort;
endif;

/*-----*\
 *
 * The following calls ListAddString to add a string to the list.
 *
 *-----*/
szString = "Value one";
if (ListAddString(listID, szString, AFTER) < 0) then

    MessageBox("ListAddString failed.", INFORMATION);
endif;

/*-----*\
 *
 * The following calls ListAddString to add a string to the list. this string
 * is set as the current element.
 *
 *-----*/
szString = "Value three";
if (ListAddString(listID, szString, AFTER) < 0) then

    MessageBox("ListAddString failed.", INFORMATION);
endif;

/*-----*\
 *
 * The following calls ListAddString to add a string before the previous
 * current element.
 *
 *-----*/
szString = "Value two";
if (ListAddString (listID, szString, BEFORE) < 0) then

    MessageBox("ListAddString failed.", INFORMATION);
```

```
endif;

szTitle = "ListAddString Example";
szMsg   = "The following is a list of the strings added:";

// Show the list of strings.
SdShowInfoList(szTitle, szMsg, listID);

// Retrieve and display the current element.
ListCurrentString(listID, svString);
MessageBox(svString, INFORMATION);

ListDestroy(listID);

endprogram

#define SD_SINGLE_DIALOGS 1
#define SD_SHOWINFOLIST  1

#include "Sddialog.rul"

// Source file: Is5fn095.rul
```

{button ,PI("",`comment')} [Feedback](#)



ListCount

Syntax

ListCount (listID);

Description

Use the ListCount function to obtain the number of elements in a list.

Parameters

listID

Specifies the name of a string or number list.

Return values

>= 0

The number of items in the list.

< 0

ListCount was unable to determine the number of elements in the list.

Comments

This function works with strings and number lists.

{button ,JI('LANGREF.HLP>Examples','ListCount_example')} [Example](#)

{button ,AL('ListProcessing',0,'')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



ListCount example

```
/*-----*\
 *
 * This example illustrates the usage of the ListCount function.
 *
 * The following adds the names of the program folders to a string list and
 * then determines how many strings are in the list.
 *
 \*-----*/

STRING svString;
LIST listID;
NUMBER nCount;

program

// Create a string list.
listID = ListCreate(STRINGLIST);

if (listID = LIST_NULL) then

    MessageBox("Unable to create list.", SEVERE);
    abort;
endif;

// The following retrieves the names of the program folders into a list
GetGroupNameList(listID);

/*-----*\
 *
 * ListCount retrieves the number of elements in a list.
 *
 \*-----*/
nCount = ListCount(listID);

if (nCount < 0) then

    MessageBox("ListCount failed.", SEVERE);
else

    SprintfBox(INFORMATION, "ListCount", "Since there are %i program " +
        "folders on your desktop, there are %i items in the list.",
        nCount, nCount);
endif;

endprogram

// Source file: Is5fn096.rul
```

{button ,PI("",`comment')} [Feedback](#)



ListCreate

Syntax

ListCreate (nListType);

Description

The ListCreate function creates an empty string or number list. Remember that a list cannot contain both types of elements. InstallShield supplies separate sets of functions to work with string lists and with number lists. You must not use the ID of a number list with the string list functions and vice versa. Use list functions that end in "Item" on number lists and use list functions that end in "String" on string lists.

When you no longer need the list, you can destroy the list with the [ListDestroy](#) function.

Each list has a pointer that identifies an element as the "current" element of the list. The various list functions reposition the current element of the list.

When calling any of the list functions, you must pass a valid ID of the list returned by this function. Verify this function was successful in creating the list. Otherwise, all the list functions will fail on the invalid list.

Parameters

nListType

Specifies the type of list to build. Pass one of the following predefined constants in this parameter:

NUMBERLIST

Specifies a number list.

STRINGLIST

Specifies a string list.

Return values

ListID

The ID of the newly created, empty list. You must use this ID whenever you want to use this list in other InstallShield list functions. You must check this variable and be sure the function did not return LIST_NULL.

LIST_NULL

Indicates that InstallShield is unable to create a list. This is a seldom seen condition that is the result of a serious memory problem. You may experience difficulties in continuing the setup with such memory problems.

Comments

Before you can pass a valid list ID to any function that requires a list, you must build the list using ListCreate. You can create any number of lists in a script. A list may contain any number of elements. The only constraint is the amount of available free memory.

{button ,JI('LANGREF.HLP>Examples','ListCreate_example')} [Example](#)

{button ,AL('ListProcessing',0,'')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



ListCreate example

```
/*-----*\
 *
 * This example illustrates the usage of the ListCreate and ListDestroy
 * functions.
 *
 * ListCreate is called to create a number list. ListDestroy is called to
 * destroy it.
 *
\*-----*/

LIST listID;
NUMBER nItem, nvItem;

program

/*-----*\
 *
 * The following creates an empty number list.
 *
\*-----*/
listID = ListCreate(NUMBERLIST);

// Check if the list was actually created.
if (listID = LIST_NULL) then

    MessageBox("Unable to create list.", SEVERE);

    abort;
endif;

// Add numbers to the list
nItem = 1078;
ListAddItem(listID, nItem, AFTER);
nItem = 304;
ListAddItem(listID, nItem, AFTER);

// Retrieve the current item in the list (304).
ListCurrentItem(listID, nvItem);

PrintfBox(INFORMATION, "ListCreate", "Current item in list: %d", nvItem);

// Retrieve the first item in the list (1078).
ListGetFirstItem(listID, nvItem);

PrintfBox(INFORMATION, "ListCreate", "First item in list: %d", nvItem);

/*-----*\
 *
 * The following removes the list from memory.
 *
\*-----*/
ListDestroy(listID);

endprogram

// Source file: Is5fn097.rul
```

{button ,PI("",`comment')} Feedback



ListCurrentItem

Syntax

ListCurrentItem (listID, nvItem);

Description

The ListCurrentItem function retrieves the current element from the number list specified in listID.

Parameters

listID

Specifies the name of a number list.

nvItem

Returns the number of the current element in the list.

Return values

0

Indicates that the function successfully retrieved the current element in a number list.

< 0

Indicates that the function was unable to retrieve the current element in a number list.

END_OF_LIST

Indicates that the list is empty and therefore does not have a current element.

Comments

- n This function works only with number lists.
- n You can also use the [ListGetFirstItem](#) and [ListGetNextItem](#) functions to traverse the list and make any element the current element.

{button ,JI('LANGREF.HLP>Examples','ListCurrentItem_example')} [Example](#)

{button ,AL('ListProcessing',0,'')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



ListCurrentItem example

```
/*-----*\
 *
 * This example illustrates the usage of the ListDeleteItem and
 * ListCurrentItem functions.
 *
 * This script first creates a list of numbers and then displays the current
 * number in a dialog box. ListDeleteItem is then called twice, deleting the
 * last two numbers in the list. The current number is displayed again,
 * showing that only the first item remained.
 *
\*-----*/

STRING    szTitle, szMsg;
LIST      listID;
NUMBER    nItem, nvItem;

program

    // Create the number list.
    listID = ListCreate (NUMBERLIST);

    if (listID = LIST_NULL) then

        MessageBox ("Unable to create list.", SEVERE);
        abort;
    endif;

    // Add numbers to the list.
    nItem = 1;
    ListAddItem (listID, nItem, AFTER);
    nItem = 2;
    ListAddItem (listID, nItem, AFTER);
    nItem = 3;
    ListAddItem (listID, nItem, AFTER);

    szTitle = "ListDeleteItem Example";
/*-----*\
 *
 * Display the current item in the list.
 *
\*-----*/
    if (ListCurrentItem(listID, nvItem) < 0) then

        MessageBox("First call to ListCurrentItem failed.", SEVERE);
    else

        szMsg = "Current number in list: %d";
        sprintfBox(INFORMATION, szTitle, szMsg, nvItem);
    endif;

/*-----*\
 *
 * ListDeleteItem destroys the current item, 3. Then ListDeleteItem is called
 * again to destroy the new current item, 2.
 *
\*-----*/
    if (ListDeleteItem(listID) < 0) then

        MessageBox("First call to ListDeleteItem failed.", SEVERE);
```

```
endif;

if (ListDeleteItem (listID) < 0) then

    MessageBox("Second call to ListDeleteItem failed.", SEVERE);
endif;

/*-----*\
*
* Display the new current item in the list, 1.
*
*\-----*/
if (ListCurrentItem(listID, nvItem) < 0) then

    MessageBox("Second call to ListCurrentItem failed.", SEVERE);
else

    szMsg = "Current number in list: %d";
    sprintfBox(INFORMATION, szTitle, szMsg, nvItem);
endif;

// Remove the list from memory.
ListDestroy (listID);

endprogram

// Source file: Is5fn098.rul
```

{button ,PI("",`comment')} Feedback



ListCurrentString

Syntax

```
ListCurrentString (listID, svString);
```

Description

The ListCurrentString function retrieves the current element from the string list specified in listID.

Parameters

listID

Specifies the name of a string list.

svString

Returns the number of the current element in the list.

Return values

0

Indicates that the function successfully retrieved the current element in a string list.

< 0

Indicates that the function was unable to retrieve the current element in a string list.

END_OF_LIST

Indicates that the list is empty and therefore does not have a current element.

Comments

- n This function works only with string lists.
- n You can also use the [ListGetFirstString](#) and [ListGetNextString](#) functions to traverse the list and make any element the current element.

{button ,JI('LANGREF.HLP>Examples','ListCurrentString_example')} [Example](#)

{button ,AL('ListProcessing',0,'')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



ListCurrentString example

```
/*-----*\
 *
 * This example illustrates the usage of the ListCurrentString function.
 *
 * ListAddString retrieves the current string element from a string list.
 *
 \*-----*/

    STRING szString, svString, szTitle, szMsg;
    LIST listID;

program

    // Create the string list.
    listID = ListCreate(STRINGLIST);

    // Error-check string list.
    if (listID = LIST_NULL) then

        MessageBox ("Unable to create list.", SEVERE);
        abort;
    endif;

    // Add strings to the list.
    szString = "This is the first string";
    ListAddString (listID, szString, AFTER);
    szString = "This is the second string";
    ListAddString (listID, szString, AFTER);

/*-----*\
 *
 * The following returns the current element in the string list to the
 * svString variable.
 *
 \*-----*/

    if (ListCurrentString (listID, svString) < 0) then

        MessageBox("ListCurrentString failed.", SEVERE);
    else

        szTitle = "ListCurrentString Example";
        szMsg = "Current element in list: '%s'";
        sprintfBox (INFORMATION, szTitle, szMsg, svString);
    endif;

endprogram

// Source file: Is5fn099.rul
```

{button ,PI("",`comment')} [Feedback](#)



ListDeleteItem

Syntax

ListDeleteItem (listID);

Description

The ListDeleteItem function removes the current element from the number list you specify in listID.

Parameters

listID

Specifies the name of the number list from which to delete the current element.

Return values

0

Indicates that the function successfully deleted the current element from a number list.

< 0

Indicates that the function was unable to delete the current element from a number list.

END_OF_LIST

Indicates that the list is empty and therefore does not have a current element.

Comments

- n This function works only with number lists.
- n You can also use the [ListGetFirstItem](#) and [ListGetNextItem](#) functions to traverse the list and make any element the current element.

{button ,JI('LANGREF.HLP>Examples','ListDeleteItem_example')}

[Example](#)

{button ,AL('ListProcessing',0,'')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



ListDeleteItem example

```
/*-----*\
 *
 * This example illustrates the usage of the ListDeleteItem and
 * ListCurrentItem functions.
 *
 * This script first creates a list of numbers and then displays the current
 * number in a dialog box. ListDeleteItem is then called twice, deleting the
 * last two numbers in the list. The current number is displayed again,
 * showing that only the first item remained.
 *
 *-----*/

STRING    szTitle, szMsg;
LIST      listID;
NUMBER    nItem, nvItem;

program

    // Create the number list.
    listID = ListCreate (NUMBERLIST);

    if (listID = LIST_NULL) then

        MessageBox ("Unable to create list.", SEVERE);
        abort;
    endif;

    // Add numbers to the list.
    nItem = 1;
    ListAddItem (listID, nItem, AFTER);
    nItem = 2;
    ListAddItem (listID, nItem, AFTER);
    nItem = 3;
    ListAddItem (listID, nItem, AFTER);

    szTitle = "ListDeleteItem Example";
/*-----*\
 *
 * Display the current item in the list.
 *
 *-----*/
    if (ListCurrentItem(listID, nvItem) < 0) then

        MessageBox("First call to ListCurrentItem failed.", SEVERE);
    else

        szMsg = "Current number in list: %d";
        sprintfBox(INFORMATION, szTitle, szMsg, nvItem);
    endif;

/*-----*\
 *
 * ListDeleteItem destroys the current item, 3. Then ListDeleteItem is called
 * again to destroy the new current item, 2.
 *
 *-----*/
    if (ListDeleteItem(listID) < 0) then

        MessageBox("First call to ListDeleteItem failed.", SEVERE);
```

```
endif;

if (ListDeleteItem (listID) < 0) then

    MessageBox("Second call to ListDeleteItem failed.", SEVERE);
endif;

/*-----*\
*
* Display the new current item in the list, 1.
*
*\-----*/
if (ListCurrentItem(listID, nvItem) < 0) then

    MessageBox("Second call to ListCurrentItem failed.", SEVERE);
else

    szMsg = "Current number in list: %d";
    sprintfBox(INFORMATION, szTitle, szMsg, nvItem);
endif;

// Remove the list from memory.
ListDestroy (listID);

endprogram

// Source file: Is5fn098.rul
```

{button ,PI("",`comment')} Feedback



ListDeleteString

Syntax

ListDeleteString (listID);

Description

The ListDeleteString function removes the current element from the string list you specify in listID.

Parameters

listID

Specifies the name of the string list from which to delete the current element.

Return values

0

Indicates that the function successfully deleted the current element from a string list.

< 0

Indicates that the function was unable to delete the current element from a string list.

END_OF_LIST

Indicates that the list is empty and therefore does not have a current element.

Comments

- n This function works only with string lists.
- n You can also use the [ListGetFirstString](#) and [ListGetNextString](#) functions to traverse the list and make any element the current element.

{button ,JI('LANGREF.HLP>Examples','ListDeleteString_example')} [Example](#)

{button ,AL('ListProcessing',0,'')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



ListDeleteString example

```
/*-----*\
 *
 * This example illustrates the usage of the ListDeleteString function.
 *
 * This script first creates a list of strings and then displays the current
 * string in a dialog box. ListDeleteString is then called twice, deleting
 * the last two strings in the list. The current string is displayed again,
 * showing that only the first string remained.
 *
 *-----*/

STRING  szString, svString, szTitle, szMsg;
LIST    listID;

program

    // Create the string list.
    listID = ListCreate (STRINGLIST);

    if (listID = LIST_NULL) then

        MessageBox ("Unable to create list.", SEVERE);
        abort;
    endif;

    // Add strings to the list.
    szString = "Value one";
    ListAddString (listID, szString, AFTER);
    szString = "Value two";
    ListAddString (listID, szString, AFTER);
    szString = "Value three";
    ListAddString (listID, szString, AFTER);

    szTitle = "ListDeleteString & ListCurrentString";
/*-----*\
 *
 * Display the current string in the list.
 *
 *-----*/
    if (ListCurrentString(listID, svString) < 0) then

        MessageBox("First call to ListCurrentString failed.", SEVERE);
    else

        szMsg  = "Current string in list: %s";
        sprintfBox(INFORMATION, szTitle, szMsg, svString);
    endif;

/*-----*\
 *
 * ListDeleteString destroys the current string, "Value three".
 * ListDeleteString is then called again to destroy the new current string,
 * "Value two".
 *
 *-----*/
    if (ListDeleteString (listID) < 0) then

        MessageBox("First call to ListDeleteString failed.", SEVERE);
    endif;
```

```
    if (ListDeleteString (listID) < 0) then
        MessageBox("Second call to ListDeleteString failed.", SEVERE);
    endif;

/*-----*\
*
* Display the current string in the new list.
*
\*-----*/
    if (ListCurrentString(listID, svString) < 0) then
        MessageBox("Second call to ListCurrentString failed.", SEVERE);
    else
        szMsg = "Current string in list: %s";
        sprintfBox(INFORMATION, szTitle, szMsg, svString);
    endif;

    // Remove the list from memory.
    ListDestroy (listID);

endprogram

// Source file: Is5fn101.rul
```

{button ,PI("",`comment')} Feedback



ListDestroy

Syntax

```
ListDestroy (listID);
```

Description

The ListDestroy function destroys the contents of a list and the list itself. Use this function to remove the string or number list identified in listID.

Parameters

listID

Specifies the name of the string or number list to destroy.

Return values

0

Indicates that the function successfully destroyed the list, removing it from memory.

< 0

Indicates that the function was unable to destroy the list.

Comments

- n This function works with both string and number lists. Once you destroy a list, do not use that listID in any list function.
- n Destroy all the lists you create when you no longer need them or at the end of the setup script. When you destroy a list, you free all memory associated with the list.

{button ,JI(`LANGREF.HLP>Examples`,`ListDestroy_example`)} Example

{button ,AL(`ListProcessing`,0,`,`)} See also

{button ,PI(``,`comment`)} Feedback



ListDestroy example

```
/*-----*\
 *
 * This example illustrates the usage of the ListCreate and ListDestroy
 * functions.
 *
 * ListCreate is called to create a number list. ListDestroy is called to
 * destroy it.
 *
 \*-----*/

    LIST listID;
    NUMBER nItem, nvItem;

program

/*-----*\
 *
 * The following creates an empty number list.
 *
 \*-----*/
    listID = ListCreate(NUMBERLIST);

    // Check if the list was actually created.
    if (listID = LIST_NULL) then

        MessageBox("Unable to create list.", SEVERE);

        abort;
    endif;

    // Add numbers to the list
    nItem = 1078;
    ListAddItem(listID, nItem, AFTER);
    nItem = 304;
    ListAddItem(listID, nItem, AFTER);

    // Retrieve the current item in the list (304).
    ListCurrentItem(listID, nvItem);

    SprintfBox(INFORMATION, "ListCreate", "Current item in list: %d", nvItem);

    // Retrieve the first item in the list (1078).
    ListGetFirstItem(listID, nvItem);

    SprintfBox(INFORMATION, "ListCreate", "First item in list: %d", nvItem);

/*-----*\
 *
 * The following removes the list from memory.
 *
 \*-----*/
    ListDestroy(listID);

endprogram

// Source file: Is5fn097.rul

)
```

{button ,PI("",`comment')} Feedback



ListFindItem

Syntax

ListFindItem (listID, nItem);

Description

The ListFindItem function searches for a specific element in a number list, starting at the current element and continuing down through the list from that point. If you want to start the search from the beginning of the list, use the [ListGetFirstItem](#) function. When ListFindItem finds the element, it becomes the current element in the list.

Parameters

listID

Specifies the name of a number list to search.

nItem

Specifies the item to find in the list.

Return values

0

Indicates that the function successfully found the requested element.

< 0

Indicates that the function was unable to find the requested element.

END_OF_LIST

Indicates InstallShield reached the end of the number list.

Comments

This function works only with number lists.

{button ,JI('LANGREF.HLP>Examples','ListFindItem_example')}

[Example](#)

{button ,AL('ListProcessing',0,'')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



ListFindItem example

```
/*-----*\
 *
 * This example illustrates the usage of the ListFindItem function.
 *
 * This script creates a number list and sets various numbers to it.
 * The user is then prompted to enter a number. ListFindItem is called to
 * search the list for the number the user entered. A message box is
 * displayed if the number was or was not found.
 *
 *-----*/

    STRING svItem, szTitle;
    NUMBER nItem, nResult;
    LIST listID;

program

    // Create a number list.
    listID = ListCreate(NUMBERLIST);

    if (listID = LIST_NULL) then

        MessageBox("Unable to create list.", SEVERE);
        abort;
    endif;

    // Add items to the list
    nItem = 1;
    ListAddItem(listID, nItem, AFTER);
    nItem = 9;
    ListAddItem(listID, nItem, AFTER);
    nItem = 5;
    ListAddItem(listID, nItem, AFTER);

    // Set the first item to the current element.
    ListGetFirstItem(listID, nItem);

    // Prompt user for a number.
    AskText("Please enter a number between 1 and 10.", "5", svItem);

    // Change the string version of the number to an actual number.
    StrToNum(nItem, svItem);

    szTitle = "ListFindItem Example";

/*-----*\
 *
 * The following searches for the list of numbers. A message box is displayed
 * if the number is or is not found.
 *
 *-----*/

    nResult = ListFindItem(listID, nItem);

    if (nResult < 0) then

        MessageBox("ListFindItem failed.", SEVERE);
    elseif (nResult = END_OF_LIST) then

        SprintfBox(WARNING, szTitle, "Sorry, %d is not in the list.", nItem);
```

```
elseif (nResult = 0) then

    SprintfBox(INFORMATION, szTitle, "Yes, %d is in the list.", nItem);
endif;

// Remove list from memory.
ListDestroy(listID);

endprogram

// Source file: Is5fn103.rul
```

{button ,PI("",`comment')} [Feedback](#)



ListFindString

Syntax

```
ListFindString (listID, szString);
```

Description

The ListFindString function searches for a specified element in a string list, starting at the current element and continuing from that point. If you want to start the search from the beginning of the string list, call the [ListGetFirstString](#) function. When ListFindString finds the string, it becomes the current element in the list.

This function performs a case-sensitive comparison of the strings.

Parameters

listID

Specifies the name of a string list to search.

szString

Specifies the string to find in the list. Note that InstallShield performs a case-sensitive comparison when searching for this string.

Return values

0

Indicates that the function successfully found the requested element.

< 0

Indicates that the function was unable to find the requested element.

END_OF_LIST

Indicates that InstallShield reached the end of the list.

Comments

This function works only with string lists.

```
{button ,JI('LANGREF.HLP>Examples','ListFindString_example')}
```

[Example](#)

```
{button ,AL('ListProcessing',0,'')} See also
```

```
{button ,PI('','comment')} Feedback
```



ListFindString example

```
/*-----*\
 *
 * This example illustrates the usage of the ListFindString function.
 *
 * This script prompts the user to input the name of a program folder on the
 * target system. ListFindString is then called to search for this folder
 * name out of a list comprised of all folder names on the target system. A
 * message is displayed if the folder name is or is not found.
 *
 * NOTE: The function GetGroupNameList used in this example may return an
 *       error if the target system has a shell other than the Windows 95
 *       shell or program manager.
 *
 *-----*/

    STRING szString, szTitle;
    LIST   listID;
    NUMBER nResult;

program

    // Create a number list
    listID = ListCreate(STRINGLIST);

    if (listID = LIST_NULL) then

        MessageBox("Unable to create list.", SEVERE);
        abort;
    endif;

    // Fill the list with the names of program folders.
    GetGroupNameList(listID);

    // Reset list to the top.
    ListGetFirstString(listID, szString);

    // Prompt user to enter a folder name.
    AskText("Please enter a folder name:", "Startup", szString);

/*-----*\
 *
 * The following searches the entire list for the string the user entered. A
 * message box is displayed if the string is or is not found in the list.
 *
 *-----*/

    nResult = ListFindString(listID, szString);

    szTitle = "ListFindString Example";
    if (nResult < 0) then

        MessageBox("ListFindString failed.", SEVERE);
    elseif (nResult = END_OF_LIST) then

        sprintfBox(WARNING, szTitle, "%s could not be found.", szString);
    elseif (nResult = 0) then

        sprintfBox(INFORMATION, szTitle, "ListFindString found: %s.", szString);
    endif;
```

```
// Remove list from memory.  
ListDestroy(listID);  
  
endprogram  
  
// Source file: Is5fn104.rul
```

```
{button ,PI("",`comment')} Feedback
```



ListGetFirstItem

Syntax

```
ListGetFirstItem (listID, nvItem);
```

Description

The ListGetFirstItem function retrieves the first element from a number list. The first item becomes the current element in the list.

Parameters

listID

Specifies the name of a valid number list from which to retrieve the first element.

nvItem

Returns the first element of the number list.

Return values

0

Indicates that the function successfully retrieved the first element in a number list.

-1

Indicates that an error prevented the function from retrieving the first element in a number list.

END_OF_LIST

Indicates that the list is empty.

Comments

This function works only with number lists.

{button ,JI('LANGREF.HLP>Examples','ListGetFirstItem_example')} [Example](#)

{button ,AL('ListProcessing',0,'')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



ListGetFirstItem example

```
/*-----*\
 *
 * This example illustrates the usage of the ListGetFirstItem and
 * ListGetNextItem functions.
 *
 * This script first creates a number list and adds various numbers to it.
 * ListGetFirstItem is called, setting the nvItem variable to the first item,
 * and also setting this first item to be the current element. nvItem is
 * displayed in a message box from within a loop. Next ListGetNextItem is
 * called, setting nvItem to the next item in the list. ListGetNextItem also
 * sets this item to be the current element. This loop is continued until the
 * end of the list is reached.
 *
 *-----*/

    STRING szTitle, szMsg;
    LIST   listID;
    NUMBER nItem, nvItem, nResult;

program

    // Create and set a variable for a number list.
    listID = ListCreate(NUMBERLIST);

    if (listID = LIST_NULL) then

        MessageBox("Unable to create list.", SEVERE);
        abort;
    endif;

    // Add items to the number list.
    nItem = 1078;
    ListAddItem(listID, nItem, AFTER);
    nItem = 304;
    ListAddItem(listID, nItem, AFTER);

/*-----*\
 *
 * The following retrieves the first item in a number list. ListGetFirstItem
 * also sets this to the current element. nResult is set to END_OF_LIST when
 * the end of the number list has been reached. nResult is set to 0 if the
 * retrieval was successful, 1 otherwise.
 *
 *-----*/

    nResult = ListGetFirstItem(listID, nvItem);

    while (nResult != END_OF_LIST)

        szTitle = "ListGetFirstItem & ListGetNextItem";

        // Display nvItem.
        sprintfBox(INFORMATION, szTitle, "%i", nvItem);

/*-----*\
 *
 * The following retrieves the next item in the number list. ListGetNextItem
 * also sets this to the current element. nResult is set to END_OF_LIST when
 * the end of the number list has been reached. nResult is set to 0 if the
 * retrieval was successful, 1 otherwise.
```

```
*
\*-----*/
    nResult = ListGetNextItem(listID, nvItem);
    endwhile;

    // Remove the list from memory
    ListDestroy(listID);

endprogram

// Source file: Is5fn105.rul
```

{button ,PI("",`comment')} [Feedback](#)



ListGetFirstString

Syntax

```
ListGetFirstString (listID, svString);
```

Description

The ListGetFirstString function retrieves the first element from a string list. The first string becomes the current element in the list.

Parameters

listID

Specifies the name of a valid string list from which to retrieve the first element.

svString

Returns the first element of the string list.

Return values

0

Indicates that the function successfully retrieved the first element in a string list.

-1

Indicates that an error prevented the function from retrieving the first element in a string list.

END_OF_LIST

Indicates that the list is empty.

Comments

This function works only with string lists.

{button ,JI('LANGREF.HLP>Examples','ListGetFirstString_example')} [Example](#)

{button ,AL('ListProcessing',0,'')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



ListGetFirstString example

```
/*-----*\
 *
 * This example illustrates the usage of the ListGetFirstString and
 * ListGetNextString functions.
 *
 * ListGetFirstString is called to retrieve the first string in a list, and to
 * set the first string as the current element.
 *
 * ListGetNextString is called to retrieve the next string in a list, and to
 * set the next string as the current element.
 *
 \*-----*/

    STRING svString;
    LIST listID;
    NUMBER nResult;

program

    listID = ListCreate(STRINGLIST);

    // The following places the names of the program folders into a list
    GetGroupNameList(listID);

/*-----*\
 *
 * The following retrieves the first string in the list and returns it in the
 * svString variable. nResult is also set to any return values
 * ListGetFirstString returns.
 *
 \*-----*/
    nResult = ListGetFirstString(listID, svString);

    while (nResult != END_OF_LIST)

        // Display the current element each time the loop is traversed.
        MessageBox(svString, INFORMATION);

/*-----*\
 *
 * The following retrieves the next string in the list and returns it in the
 * svString variable. nResult is also set to any return values
 * ListGetFirstString returns.
 *
 \*-----*/
    nResult = ListGetNextString(listID, svString);
    endwhile;

    ListDestroy(listID);

endprogram

// Source file: Is5fn106.rul
```



ListGetNextItem

Syntax

ListGetNextItem (listID, nvItem);

Description

The ListGetNextItem function retrieves the item after the current element in a number list. The retrieved item becomes the current element in the list.

Parameters

listID

Specifies the name of a valid number list from which to retrieve the next element.

nvItem

Returns the item that follows the current element in the number list. That item becomes the current element in the list.

Return values

0

Indicates that the function successfully retrieved the element after the current element in a number list.

-1

Indicates that an error prevented the function from retrieving the specified element in a number list.

END_OF_LIST

Indicates that the current item is the last element in the list.

Comments

This function works only with number lists.

{button ,JI('LANGREF.HLP>Examples','ListGetNextItem_example')} [Example](#)
{button ,AL('ListProcessing',0,'')} [See also](#)
{button ,PI('','comment')} [Feedback](#)



ListGetNextItem example

```
/*-----*\
*
* This example illustrates the usage of the ListGetFirstItem and
* ListGetNextItem functions.
*
* This script first creates a number list and adds various numbers to it.
* ListGetFirstItem is called, setting the nvItem variable to the first item,
* and also setting this first item to be the current element. nvItem is
* displayed in a message box from within a loop. Next ListGetNextItem is
* called, setting nvItem to the next item in the list. ListGetNextItem also
* sets this item to be the current element. This loop is continued until the
* end of the list is reached.
*
*-----*/

STRING szTitle, szMsg;
LIST listID;
NUMBER nItem, nvItem, nResult;

program

// Create and set a variable for a number list.
listID = ListCreate(NUMBERLIST);

if (listID = LIST_NULL) then

    MessageBox("Unable to create list.", SEVERE);
    abort;
endif;

// Add items to the number list.
nItem = 1078;
ListAddItem(listID, nItem, AFTER);
nItem = 304;
ListAddItem(listID, nItem, AFTER);

/*-----*\
*
* The following retrieves the first item in a number list. ListGetFirstItem
* also sets this to the current element. nResult is set to END_OF_LIST when
* the end of the number list has been reached. nResult is set to 0 if the
* retrieval was successful, 1 otherwise.
*
*-----*/
nResult = ListGetFirstItem(listID, nvItem);

while (nResult != END_OF_LIST)

    szTitle = "ListGetFirstItem & ListGetNextItem";

    // Display nvItem.
    sprintfBox(INFORMATION, szTitle, "%i", nvItem);

/*-----*\
*
* The following retrieves the next item in the number list. ListGetNextItem
* also sets this to the current element. nResult is set to END_OF_LIST when
* the end of the number list has been reached. nResult is set to 0 if the
* retrieval was successful, 1 otherwise.
```

```
*
\*-----*/
    nResult = ListGetNextItem(listID, nvItem);
    endwhile;

    // Remove the list from memory
    ListDestroy(listID);

endprogram

// Source file: Is5fn105.rul
```

{button ,PI("",`comment')} [Feedback](#)



ListGetNextString

Syntax

```
ListGetNextString (listID, svString);
```

Description

The ListGetNextString function retrieves the element after the current element in a string list. The retrieved element becomes the current element in the list.

Parameters

listID

Specifies the name of a valid string list from which to retrieve the next element.

svString

Returns the string that follows the current element of the string list. This string becomes the current element in the list.

Return values

0

Indicates that the function successfully retrieved the element after the current element in a string list.

-1

Indicates that an error prevented the function from retrieving the specified element in a string list.

END_OF_LIST

Indicates that the current item is the last element in the list.

Comments

This function works only with string lists.

{button ,JI('LANGREF.HLP>Examples','ListGetNextString_example')} [Example](#)
{button ,AL('ListProcessing',0,'')} [See also](#)
{button ,PI('','comment')} [Feedback](#)



ListGetNextString example

```
/*-----*\
 *
 * This example illustrates the usage of the ListGetFirstString and
 * ListGetNextString functions.
 *
 * ListGetFirstString is called to retrieve the first string in a list, and to
 * set the first string as the current element.
 *
 * ListGetNextString is called to retrieve the next string in a list, and to
 * set the next string as the current element.
 *
 \*-----*/

    STRING svString;
    LIST listID;
    NUMBER nResult;

program

    listID = ListCreate(STRINGLIST);

    // The following places the names of the program folders into a list
    GetGroupNameList(listID);

/*-----*\
 *
 * The following retrieves the first string in the list and returns it in the
 * svString variable. nResult is also set to any return values
 * ListGetFirstString returns.
 *
 \*-----*/
    nResult = ListGetFirstString(listID, svString);

    while (nResult != END_OF_LIST)

        // Display the current element each time the loop is traversed.
        MessageBox(svString, INFORMATION);

/*-----*\
 *
 * The following retrieves the next string in the list and returns it in the
 * svString variable. nResult is also set to any return values
 * ListGetFirstString returns.
 *
 \*-----*/
    nResult = ListGetNextString(listID, svString);
    endwhile;

    ListDestroy(listID);

endprogram

// Source file: Is5fn106.rul
```



ListReadFromFile

Syntax

ListReadFromFile (listID, szFile);

Description

The ListReadFromFile function reads a text file into a list. Once you load a text file into a list, you can use it for various functions in the setup, such as displaying a README file at the end of the setup or writing a string list to the disk with [ListWriteToFile](#).

This function gives you an easy way to load an entire file into a list, rather than building the list one item at a time.

Parameters

listID

Specifies the name of an empty string list to which each line of the file will be added as a separate element. The string list must be created first by calling the [ListCreate](#) function.

szFile

Specifies the fully qualified name of the file which will be read into the list.

Return values

0

Indicates that the function successfully read the lines of text in a file into a list.

< 0

Indicates that the function was unable to read the lines from a text file into a list.

Comments

- n This function detects the newline characters at the end of each string and uses the characters as delimiters for each element in the list.
- n This function operates on string lists and text files only.

{button ,JI('LANGREF.HLP>Examples','ListReadFromFile_example')} [Example](#)

{button ,AL('ListProcessing',0,'')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



ListReadFromFile example

```
/*-----*\
 *
 * This example illustrates the usage of the ListReadFromFile and
 * the ListWriteToFile functions.
 *
 * ListReadFromFile is called to read a file into a list. This list is then
 * altered, after which ListWriteToFile is called to write the new list into
 * a file.
 *
 * NOTE: In order for this script to run properly, it is advised to set the
 *       preprocessor constants to a valid directory and files on the target
 *       system.
 *-----*/

#define FILE_TO_LIST "EXAMPLE\\FILELIST.TXT"
#define LIST_TO_FILE "EXAMPLE\\LISTFILE.TXT"
#define EXAMPLE_PATH "EXAMPLE"

#include "Sddialog.h";

STRING  szOldFileName, szNewFileName, szPath, szTitle;
LIST    listID

program

listID      = ListCreate(STRINGLIST);
szPath      = EXAMPLE_PATH;
szOldFileName = FILE_TO_LIST;
szNewFileName = LIST_TO_FILE;
szTitle     = "ListReadFromFile & ListWriteToFile";

SRCDIR      = szPath;
TARGETDIR   = szPath;

/*-----*\
 *
 * The following reads a file into a string list.
 *-----*/
if (ListReadFromFile(listID, szOldFileName) < 0) then

    // Error check.
    MessageBox("ListReadFromFile failed.", SEVERE);
else

    // Display the list.
    SdShowInfoList(szTitle, "The example list file:", listID);
endif;

// Add a new string to the list.
ListAddString(listID, "This will be added to the list.", AFTER);

/*-----*\
 *
 * The following writes the new list into a file.
 *-----*/
if (ListWriteToFile(listID, szNewFileName) < 0) then
```

```
        MessageBox("ListWriteToFile Failed", SEVERE);
    else

        MessageBox("Successfully wrote to new file.", INFORMATION);
    endif;

endprogram

#define SD_SINGLE_DIALOGS 1
#define SD_SHOWINFOLIST 1

#include "Sddialog.rul"

// Source file: Is5fn109.rul
```

{button ,PI("",`comment')} Feedback



ListSetCurrentItem

Syntax

ListSetCurrentItem (listID, nItem);

Description

The ListSetCurrentItem function assigns the value of nItem to the current element in a number list.

Parameters

listID

Specifies the name of a number list whose current element is to be updated. This list must already have been created by a call to [ListCreate](#).

nItem

Specifies the numeric value that will replace the current element.

Return values

0

Indicates that the function successfully updated the current element in a number list.

< 0

Indicates that the function was unable to update the current element in a number list.

END_OF_LIST

Indicates that the list is empty.

Comments

This function works only with number lists.

{button ,JI('LANGREF.HLP>Examples','ListSetCurrentItem_example')} [Example](#)

{button ,AL('ListProcessing',0,'')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



ListSetCurrentItem example

```
/*-----*\
 *
 * This example illustrates the usage of the ListSetCurrentItem function.
 *
 * This script first creates and adds items to a list. This list is displayed
 * in a dialog box. ListSetCurrentItem is then called to update the value of
 * the current element in the list. This new list is then displayed.
 *
 \*-----*/

#include "Sddialog.h"

STRING  szTitle, szMsg;
LIST    listID;
NUMBER  nItem;

program

    listID = ListCreate(NUMBERLIST);

    if (listID = LIST_NULL) then

        MessageBox("Unable to create list.", SEVERE);
        abort;
    endif;

    // Add numbers to the list
    nItem = 1078;
    ListAddItem(listID, nItem, AFTER);
    nItem = 1304;
    ListAddItem(listID, nItem, AFTER);

    szTitle = "ListSetCurrentItem Example";
    szMsg    = "Elements in listID:";

    // Display the current list.
    SdShowInfoList(szTitle, szMsg, listID);

    nItem = 305;
/*-----*\
 *
 * The following sets the current item, 1304, to a new number, 305.
 *
 \*-----*/

    if (ListSetCurrentItem(listID, nItem) < 0) then

        MessageBox("ListSetCurrentItem failed.", SEVERE);
        endif;

    // Display the new altered list.
    SdShowInfoList(szTitle, szMsg, listID);

    // Remove list from memory.
    ListDestroy(listID);

endprogram

#define SD_SINGLE_DIALOGS 1
#define SD_SHOWINFOLIST 1
```

```
#include "Sddialog.rul"  
// Source file: Is5fn110.rul
```

```
{button ,PI("",`comment')} Feedback
```



ListSetCurrentString

Syntax

```
ListSetCurrentString (listID, szString);
```

Description

The ListSetCurrentString function assigns the value of szString to the current element in the string list.

Parameters

listID

Specifies the name of a string list whose current element is to be updated. This list must already have been created by a call to [ListCreate](#).

szString

Specifies the string value that will replace the current element.

Return values

0

Indicates that the function successfully updated the current element in a number list.

< 0

Indicates that the function was unable to update the current element in a number list.

END_OF_LIST

Indicates that the list is empty.

Comments

This function works only with string lists.

{button ,JI('LANGREF.HLP>Examples','ListSetCurrentString_example')} [Example](#)

{button ,AL('ListProcessing',0,'')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



ListSetCurrentString example

```
/*-----*\
 *
 * This example illustrates the usage of the ListSetCurrentString function.
 *
 * In this script, two strings are added to a null list. This list is
 * displayed in a dialog box. ListSetCurrentString is called to set the
 * current element to a new string. The list is then displayed again in a
 * dialog box.
 *
 *-----*/

#include "Sddialog.h"

STRING  szString, szTitle, szMsg;
LIST    listID;

program

    listID = ListCreate(STRINGLIST);

    if (listID = LIST_NULL) then

        MessageBox("Unable to create list.", SEVERE);
        abort;
    endif;

    szString = "First value";
    ListAddString(listID, szString, AFTER);

    szString = "Second value";

    // ListAddString sets the latest string added as the current element.
    ListAddString(listID, szString, AFTER);

    szTitle = "ListSetCurrentString Example";
    szMsg    = "Elements in listID:";

    // Display the list.
    SdShowInfoList(szTitle, szMsg, listID);

    szString = "This will replace the second value.";
/*-----*\
 *
 * The following will set the current element to szString, "This will replace
 * the second value.".
 *
 *-----*/
    if (ListSetCurrentString(listID, szString) < 0) then

        MessageBox("ListSetCurrentString failed.", SEVERE);
    endif;

    // Display the new altered list.
    SdShowInfoList(szTitle, szMsg, listID);

    // Remove list from memory.
    ListDestroy(listID);

endprogram
```

```
#define SD_SINGLE_DIALOGS 1
#define SD_SHOWINFOLIST 1

#include "Sddialog.rul"

// Source file: Is5fn111.rul
```

```
{button ,PI("",`comment')} Feedback
```



ListSetIndex

Syntax

ListSetIndex (listID, nIndex);

Description

The ListSetIndex function makes a specific element in a string or number list the current element using an index. You can also use constants to traverse a list an element at a time or to jump to the beginning or end of a list. By using indexes to access items in a list, you can treat numeric and string lists as arrays.

Index numbers starts at zero (0). For example, if you enter 5 in the parameter nIndex, the item in the sixth physical location in the list becomes the current element.

Use [ListCurrentItem](#) and [ListCurrentString](#) to retrieve the value of the current element.

Parameters

listID

Specifies the name of the string or number list whose index is to be set.

nIndex

Specifies the number of the element you want to set as the current element. List element numbering begins at zero (0). Pass a numeric value or one of the following predefined constants in this parameter:

LISTFIRST

Moves to the first element in the list.

LISTLAST

Moves to the last element in the list.

LISTNEXT

Moves to the next element in the list.

LISTPREV

Moves to the previous element in the list.

Return values

0

Indicates that the function successfully updated the current element in the list.

< 0

Indicates that the function was unable to update the current element the list.

END_OF_LIST

Indicates that the index is out of range of available list elements.

Comments

- n After you set the indexed element as the current element, you can use either the [ListCurrentItem](#) or [ListCurrentString](#) function in the script to retrieve the value of the indexed (current) item.
- n This function works on both string and number lists.

{button ,JI(`LANGREF.HLP>Examples`,`ListSetIndex_example`)} [Example](#)

{button ,AL(`ListProcessing`,0,`,`)} [See also](#)

{button ,PI(``,`comment`)} [Feedback](#)



ListSetIndex example

```
/*-----*\
 *
 * This example illustrates the usage of the ListSetIndex function.
 *
 * First a string list is created, retrieving all the program folder names
 * that currently exist in the shell. The SdShowInfoList dialog box displays
 * each element. ListCurrentString is then called to retrieve each line in
 * the string list, beginning with the third element (nIndex = 2). The
 * ListSetIndex function advances the index to the next element in the list,
 * and the MessageBox function individually displays each item.
 *
 * NOTE: If the target system has a shell other than the Windows 95 shell or
 *       Program Manager, the GetGroupNameList function may return an error.
 *
 *-----*/

#include "Sddialog.h"

STRING   svString, szTitle, szMsg;
LIST     listID;
NUMBER   nCheck, nIndex;

program

    // Create string list.
    listID = ListCreate(STRINGLIST);

    // Retrieve program folders into list.
    GetGroupNameList(listID);

    szTitle = "ListSetIndex Example";
    szMsg    = "Please note the order in which these values are listed.";
    // Display the list.
    SdShowInfoList(szTitle, szMsg, listID);

/*-----*\
 *
 * Beginning with the third element, ListSetIndex is called from within a
 * while loop to traverse the list. Each time the loop is traversed, a
 * message box displays the current element.
 *
 *-----*/

    nIndex = 2;
    nCheck = ListSetIndex(listID, nIndex);

    while (nCheck != END_OF_LIST)

        ListCurrentString(listID, svString);
        MessageBox(svString, INFORMATION);
        nCheck = ListSetIndex(listID, LISTNEXT);
    endwhile;

    // Destroy the list from memory.
    ListDestroy(listID);

endprogram

#define SD_SINGLE_DIALOGS 1
#define SD_SHOWINFOLIST 1
```

```
#include "Sddialog.rul"
```

```
// Source file: Is5fn112.rul
```

```
{button ,PI("",`comment')} Feedback
```



ListWriteToFile

Syntax

ListWriteToFile (listID, szFileName);

Description

The ListWriteToFile function writes a string list to a text file on the disk. Without this function you must write the list one element at a time into the text file. This function detects the newline characters at the end of each element and uses the characters as delimiters for each string in the list. ListWriteToFile operates on string lists and text files only.

Parameters

listID

Specifies the name of a valid string list to write into a file.

szFileName

Specifies the fully qualified name of the file to which the string list is to be written.

Return values

0

Function successful.

< 0

Function failed.

{button ,JI('LANGREF.HLP>Examples','ListWriteToFile_example')}

Example

{button ,AL('ListProcessing',0,','')} See also

{button ,PI('','comment')} Feedback



ListWriteToFile example

```
/*-----*\
 *
 * This example illustrates the usage of the ListReadFromFile and
 * the ListWriteToFile functions.
 *
 * ListReadFromFile is called to read a file into a list. This list is then
 * altered, after which ListWriteToFile is called to write the new list into
 * a file.
 *
 * NOTE: In order for this script to run properly, it is advised to set the
 *       preprocessor constants to a valid directory and files on the target
 *       system.
 *-----*/

#define FILE_TO_LIST "EXAMPLE\\FILELIST.TXT"
#define LIST_TO_FILE "EXAMPLE\\LISTFILE.TXT"
#define EXAMPLE_PATH "EXAMPLE"

#include "Sddialog.h";

STRING  szOldFileName, szNewFileName, szPath, szTitle;
LIST    listID

program

listID      = ListCreate(STRINGLIST);
szPath      = EXAMPLE_PATH;
szOldFileName = FILE_TO_LIST;
szNewFileName = LIST_TO_FILE;
szTitle     = "ListReadFromFile & ListWriteToFile";

SRCDIR      = szPath;
TARGETDIR   = szPath;

/*-----*\
 *
 * The following reads a file into a string list.
 *-----*/
if (ListReadFromFile(listID, szOldFileName) < 0) then

    // Error check.
    MessageBox("ListReadFromFile failed.", SEVERE);
else

    // Display the list.
    SdShowInfoList(szTitle, "The example list file:", listID);
endif;

// Add a new string to the list.
ListAddString(listID, "This will be added to the list.", AFTER);

/*-----*\
 *
 * The following writes the new list into a file.
 *-----*/
if (ListWriteToFile(listID, szNewFileName) < 0) then
```

```
        MessageBox("ListWriteToFile Failed", SEVERE);
    else

        MessageBox("Successfully wrote to new file.", INFORMATION);
    endif;

endprogram

#define SD_SINGLE_DIALOGS 1
#define SD_SHOWINFOLIST 1

#include "Sddialog.rul"

// Source file: Is5fn109.rul
```

{button ,PI("",`comment')} Feedback



Long filename functions

The following functions create long filenames from short filenames, convert short filenames to long filenames, and place double quotation marks around long filenames so that operating systems that handle long filenames can recognize them.

LongPathFromShortPath

Creates a long filename from a short filename.

LongPathToQuote

Inserts or removes double quotation marks around a long filename.

LongPathToShortPath

Creates a short filename from a long filename.

{button ,AL(^Long Filenames;Long Filename Format;Long Filenames and 16 bit Applications;Long Filenames and Double Quotation Marks',0,'')} [See also](#)

{button ,PI("",`comment')} [Feedback](#)



Long filenames

Windows 95 and Windows NT 4.0 support long filenames. Long filenames allow users to give directories and files more meaningful names. The term "long filename" refers to both long filenames and long paths.

InstallShield provides the [long filename functions](#) to facilitate the installation of 16-bit applications and 32-bit applications that do not recognize long filenames. It is your responsibility to determine your application's requirements. InstallShield provides the tools to help you install any kind of application.

{button ,AL(`Long Filenames;Long Filename Fn;Long Filename Format;Long Filenames and Double Quotation Marks;Long Filenames and 16 bit Applications',0,`,`)}} [See also](#)

{button ,PI(`",`comment')}} [Feedback](#)



Long filename format

Long filenames contain names longer than the conventional 8.3 (8 characters plus 3-digit extension) short filename limit. Long filenames allow the use of all the characters used in short filenames. In addition, long filenames can contain plus signs (+), commas (,), semicolons (;), equal signs (=), left and right square brackets ([]), and spaces. Leading and trailing spaces are ignored. The fully qualified long filename (including null terminating character) can be up to 256 characters long on NTFS file systems and 260 characters long on VFAT file systems.

Windows 95 and Windows NT 4.0 create a short filename for every long filename. The short filename consists of the first six characters of the long filename, a tilde (~), and a number.

{button ,AL(`Long Filenames;Long Filename Fn;Long Filename Format;Long Filenames and Double Quotation Marks;Long Filenames and 16 bit Applications',0,`,`')} [See also](#)

{button ,PI(`",`comment')} [Feedback](#)



Long filenames and double quotation marks

Under Windows 95, if you pass a long filename containing one or more space characters to the command line (such as in a DOS shell or in the Command Line field in an icon's properties sheet) you must enclose the long filename in double quotation marks. This is necessary because the command line recognizes the space character as a delimiter separating a command from other arguments. The double quotation marks convert the long filename to a string literal, allowing the command line to receive it as a single argument.

Under Windows NT, as under Windows 95, if you pass a long filename containing one or more space characters to the command line, it must be enclosed in double quotation marks. However, due to the manner in which Windows NT accesses icon files, if a long filename in double quotation marks is used in the Command Line field in an icon's properties sheet, the Windows NT default icon may display instead of the application's icon.

To ensure that your application's icon displays, you can specify the icon's path in the parameter `szIconPath` of the function [AddFolderIcon](#) when using it to add an icon to a program folder.

Double quotation marks must be removed from long filenames before they can be converted to short filenames in InstallShield. Refer to the [LongPathToQuote](#) and [LongPathToShortPath](#) functions.

{button ,AL('Long Filenames;Long Filename Functions;Long Filename Format;Long Filenames and Double Quotation Marks;Long Filenames and 16 bit Applications',0,'')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



Long filenames and 16-bit applications

You can launch 16-bit applications using long filenames, but you cannot pass long filenames as arguments to 16-bit applications. 16-bit applications require the short filename versions of long filenames to function correctly.

If your setup passes a filename to a 16-bit application, you must provide a valid short filename to the application. Long filenames will not work.

If your setup writes filenames to files such as .ini files and 16-bit applications are expected to use the files, you must write short filenames that the 16-bit applications can use.

{button ,AL(^Long Filenames;Long Filename Fn;Long Filename Format;Long Filenames and Double Quotation Marks;Long Filenames and 16 bit Applications'), , ,)} [See also](#)

{button ,PI("",`comment')} [Feedback](#)



LongPathFromShortPath

Syntax

LongPathFromShortPath (svPath);

Description

Use the LongPathFromShortPath function to convert a short filename to its equivalent long filename.



This function is supported only in 32-bit setups. 16-bit setups cannot convert to or from a long path name. When this function is called by a 16-bit setup, svPath is returned unmodified and no error code is returned.

Parameters

svPath

Specifies a short filename and returns its associated long filename.

Return values

0

Indicates that the function was successful.

< 0

Indicates that the function was not successful.

Comments

For an explanation of long filenames, see [Long filenames](#).

{button ,JI('LANGREF.HLP>Examples','LongPathFromShortPath_example')} [Example](#)

{button ,AL('LongPathToQuote;LongPathToShortPath;Long filenames;Long filename format;Long filenames and double quotation marks;Long filenames and 16-bit applications',0,','')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



LongPathFromShortPath example

```
/*-----*\
 *
 * This example illustrates the LongPathToShortPath and LongPathFromShortPath
 * functions.
 *
 * LongPathToShortPath is called to convert a long path to a short one.
 *
 * LongPathFromShortPath is called to convert the short path to a long
 * one. The message boxes created by this example are shown below the script.
 *
\*-----*/

#define LONG_PATH "C:\\Program files\\InstallShield\\InstallShield\\Support"

    STRING svPath, szTitle, szMsg;

program

    szTitle = "LongPathToShortPath & LongPathFromShortPath";
    szMsg = "Please enter an existing long path:";

    // Prompt user to enter a long path.
    AskPath(szMsg, LONG_PATH, svPath);

    // Display the long path.
    szMsg = "The long path is shown below: \n\n%s";
    sprintfBox(INFORMATION, szTitle, szMsg, svPath);

/*-----*\
 *
 * Convert the long path to a short path name.
 *
\*-----*/

    if (LongPathToShortPath(svPath) < 0) then;
        MessageBox("LongPathToShortPath failed.", SEVERE);
        abort;
    else

        // Display the short path.
        szMsg = "The short path is shown below: \n\n%s";
        sprintfBox(INFORMATION, szTitle, szMsg, svPath);
    endif;

/*-----*\
 *
 * Restore the long path from the short path.
 *
\*-----*/

    if (LongPathFromShortPath(svPath) < 0) then
        MessageBox("LongPathFromShortPath failed.", SEVERE);
    else

        // Display the restored long path.
        szMsg = "The restored long path is shown below: \n\n%s";
        sprintfBox(INFORMATION, szTitle, szMsg, svPath);
    endif;
```

endprogram

// Source file: Is5fn228.rul

{button ,PI("",`comment')} [Feedback](#)



LongPathToQuote

Syntax

LongPathToQuote (svPath, nParameter);

Description

The LongPathToQuote function places double quotation marks around a long filename or removes the double quotation marks from a long filename.

Add double quotation marks to long filenames that contain spaces before passing the long filenames to the command line. You must remove the double quotation marks from long filenames before converting them to short filenames using the [LongPathToShortPath](#) function. If you do not, the quoted long filename remains intact.



This function will add quotation marks only if there is a space character in the filename. For example, quotation marks will not be added around C:\\ThisismyApp since it is a long filename without a space.

Parameters

svPath

Specifies a long filename and returns its associated short filename.

nParameter

Specifies whether or not quotation marks are to be added to the long path or removed from the long path. Pass one of the following predefined constants in this parameter:

TRUE

Quotation marks are added to the long path.

FALSE

Quotation marks are removed from the long path.

Return values

0

Indicates that the function was successful.

< 0

Indicates that the function was not successful.

Comments

For an explanation of long filenames, see [Long filenames](#).

{button ,JI('LANGREF.HLP>Examples','LongPathToQuote_example')} [Example](#)

{button ,AL('LongPathFromShortPath;LongPathToShortPath;Long filenames;Long filename format;Long filenames and double quotation marks;Long filenames and 16-bit applications',0,',')} See also

{button ,PI('','comment')} [Feedback](#)



LongPathToQuote example

```
/*-----*\
 *
 * This example illustrates the LongPathToQuote function.
 *
 * This script calls LongPathToQuote to place double quotation marks around a
 * long file name. After this string is displayed, LongPathToQuote is called
 * again to remove the quotation marks. The string returned is displayed in
 * another message box
 *
 \*-----*/

// Define a constant for the base path (a long filename).
#define BASE_PATH "C:\\Program Files\\InstallShield\\InstallShield\\Support"

    STRING  szProgram, szMainDirectory, szMsg;

program

    szProgram = BASE_PATH;

/*-----*\
 *
 * Place double quotation marks around the long file name in szProgram.
 *
 \*-----*/
    if (LongPathToQuote(szProgram, TRUE) < 0) then
        MessageBox("First call to LongPathToQuote failed.", SEVERE);
    endif;

    // Display the quoted long file name in szProgram.
    szMsg = "The quoted long file name:\n\n" + szProgram;
    MessageBox(szMsg, INFORMATION);

/*-----*\
 *
 * Remove the quotation marks from the long file name in szProgram.
 *
 \*-----*/

    if (LongPathToQuote(szProgram, FALSE) < 0) then
        MessageBox("Second call to LongPathToQuote failed.", SEVERE);
    endif;

    // Display the long file name with quotation marks removed.
    szMsg = "The unquoted long file name is shown below: \n\n" + szProgram;
    MessageBox(szMsg, INFORMATION);

endprogram

// Source file: Is5fn229.rul
```

{button ,PI("",`comment')} Feedback



LongPathToShortPath

Syntax

LongPathToShortPath (svPath);

Description

The LongPathToShortPath function converts a long filename to its equivalent short filename. Short filenames are compatible with 16-bit programs, such as Notepad.exe or Mviewer2.exe. 16-bit programs cannot accept long filenames. The parameter svPath can be an [absolute path](#) or a [relative path](#), and it may include a filename; but the folder or file it specifies must exist on the target system.



This function is supported only in 32-bit setups. 16-bit setups cannot convert to or from a long path name. When this function is called by a 16-bit setup, svPath is returned unmodified and no error code is returned.

Parameters

svPath

Specifies a long filename and returns its associated short filename.



LongPathToShortPath removes trailing backslashes from long filenames.

Return values

0

Indicates that the function was successful.

< 0

Indicates that the function was not successful.

Comments

- n For an explanation of long filenames, see [Long Filenames](#).
- n Because LongPathToShortPath can succeed only if the specified folder or file can be found on the target system, you will usually have to set the current folder before specifying a relative path. For example if svPath contains the relative path "InstallShield\InstallShield5 Professional Edition", which exists in the folder "C:\Program Files", setup won't be able to find it unless the current folder is "C:\Program Files". Use the ChangeDirectory function to change the current folder when necessary before calling LongPathToShortPath so that the target folder or path can be found.



Use [ChangeDirectory](#) to specify a new directory.

{button ,JI('LANGREF.HLP>Examples','LongPathToShortPath_example')} [Example](#)

{button ,AL('LongPathFromShortPath;LongPathToQuote;Long filenames;Long filename format;Long filenames and double quotation marks;Long filenames and 16-bit applications',0,',')} [See also](#)

{button ,PI('','comment')} [Feedback](#)

Absolute path

An absolute path includes all of the information necessary to locate a file by starting at the root directory of a specified drive. For example, C:\Program Files\InstallShield is the absolute path to the InstallShield folder when installed on drive C.

{button ,PI("",`comment')} [Feedback](#)

Relative path

A relative path includes all of the information necessary to locate a file by starting at the current folder on the current drive, for example, "InstallShield\InstallShield5 Professional Edition". That folder can be located along that relative path only if it exists in the current directory.

{button ,PI("",`comment`)} [Feedback](#)



LongPathToShortPath example

```
/*-----*\
 *
 * This example illustrates the LongPathToShortPath and LongPathFromShortPath
 * functions.
 *
 * LongPathToShortPath is called to convert a long path to a short one.
 *
 * LongPathFromShortPath is called to convert the short path to a long
 * one. The message boxes created by this example are shown below the script.
 *
 \*-----*/

#define LONG_PATH "C:\\Program files\\InstallShield\\InstallShield\\Support"

    STRING svPath, szTitle, szMsg;

program

    szTitle = "LongPathToShortPath & LongPathFromShortPath";
    szMsg = "Please enter an existing long path:";

    // Prompt user to enter a long path.
    AskPath(szMsg, LONG_PATH, svPath);

    // Display the long path.
    szMsg = "The long path is shown below: \n\n%s";
    sprintfBox(INFORMATION, szTitle, szMsg, svPath);

/*-----*\
 *
 * Convert the long path to a short path name.
 *
 \*-----*/

    if (LongPathToShortPath(svPath) < 0) then;
        MessageBox("LongPathToShortPath failed.", SEVERE);
        abort;
    else

        // Display the short path.
        szMsg = "The short path is shown below: \n\n%s";
        sprintfBox(INFORMATION, szTitle, szMsg, svPath);
    endif;

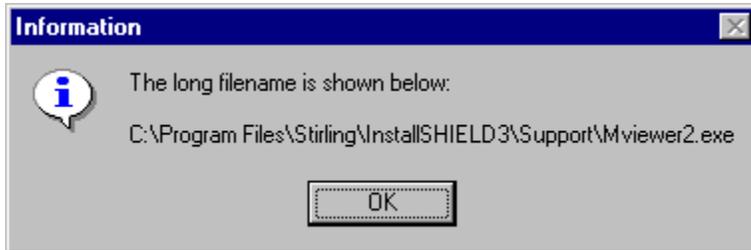
/*-----*\
 *
 * Restore the long path from the short path.
 *
 \*-----*/

    if (LongPathFromShortPath(svPath) < 0) then
        MessageBox("LongPathFromShortPath failed.", SEVERE);
    else

        // Display the restored long path.
        szMsg = "The restored long path is shown below: \n\n%s";
        sprintfBox(INFORMATION, szTitle, szMsg, svPath);
    endif;
```

endprogram

// Source file: Is5fn228.rul



{button ,PI("",`comment')} Feedback



Miscellaneous functions

The following functions serve various purposes, such as low-level hardware interface, component creation and manipulation, and user output.

Do

Executes the currently defined EXIT and HELP handlers.

DoInstall

Launches another InstallShield setup.

Handler

Specifies a label to branch to in response to exit and help events.

MessageBeep

Produces a standard warning beep.

SendMessage

Sends a message to another window or application.

Sprintf

Returns a formatted string composed of one or more character, numeric or string values.

System

Exits to DOS, restarts Windows, or reboots the computer.

VarRestore

Restores the values of the system variables SRCDIR and TARGETDIR that were saved by the last call to VarSave.

VarSave

Saves the current value of the system variables SRCDIR and TARGETDIR.

{button ,PI("",`comment')} [Feedback](#)



Do

Syntax

Do (nOperation);

Description

The Do function executes the currently defined EXIT and HELP handlers. The Do function gives you greater control over launching Help- and Exit-related events, which normally rely upon the user pressing the F1 key (HELP), the F3 key (EXIT), or the Cancel button (EXIT). Using the Do function, you can execute EXIT or HELP handlers in response to almost any user input, such as a Yes response to a dialog box. You can use the Do function to test EXIT and HELP handler functionality during setup script development.

Use the Do function to register queued [self-registering files](#). Files are queued for registration using the "batch method" for installing self-registering files. When you call Do(SELFREGISTRATIONPROCESS), InstallShield carries out self-registration of all queued files, even if one of them fails.

If Do fails for any reason, it will return -1. The names of the files that failed to self-register are stored in the InstallShield system variable ERRORFILENAME, separated by a semicolon (;).

Parameters

nOperation

Specifies the type of operation to perform. Pass one of the following predefined constants in this parameter:

EXIT

Initiates the Exit operation. If no EXIT handler is defined, the default Exit dialog is displayed.

HELP

Initiates the Help operation. If no HELP handler is defined, the function takes no action.

SELFREGISTRATIONPROCESS

Registers all self-registering files that have been queued for registration.

Return values

0

The Do function successfully initiated the specified operation.

< 0

The Do function was unable to initiate the specified operation.

Comments

The Do function allows the currently defined HELP and EXIT handlers to execute without the user's pressing F1 or F3. The Do function also provides more versatility than the goto statement, which can be used to call HELP and EXIT handler labels. The goto statement cannot be used in all circumstances, but the Do function can be called virtually anytime. Refer to the Handler function for more information on default and custom HELP and EXIT handlers.

```
{button ,JI('LANGREF.HLP>Examples','Do_example')}
```

[Example](#)

```
{button ,AL('Disable;Enable;Handler',0,'')} See also
```

```
{button ,PI('','comment')} Feedback
```



Do example

```
/*-----*\
 *
 * This example illustrates the uses of the Do function.
 *
 * This script calls Do to test the HELP and EXIT handler functionality.
 *
 \*-----*/

program

    Handler (EXIT, Exit_Handler);
    Handler (HELP, Help_Handler);

/*-----*\
 *
 * Call Do to execute the help handler.
 *
 \*-----*/
    Do (HELP);

/*-----*\
 *
 * Call Do to execute the exit handler.
 *
 \*-----*/
    Do (EXIT);

    // After testing, exit setup.
    abort;

    // The exit handler.
    Exit_Handler:

        // A message asks if the user is sure about exiting.
        if (AskYesNo ("Do you really wish to exit ?", NO) = YES) then

            // Exit if the user is sure.
            MessageBox ("Exiting...", INFORMATION);

        abort;
        else

            // Continue if not sure.
            MessageBox ("Continue the installation...", INFORMATION);
        endif;

    return;

    // The help handler.
    Help_Handler:

        // A message displaying help.
        MessageBox ("This is help.", INFORMATION);

    return;

endprogram

// Source file: Is5fn056.rul
```

{button ,PI("",`comment')} Feedback



Handler

Syntax

Handler (nObject, Label);

Description

The Handler function creates custom handlers for events such as the Help accelerator key (F1), the Cancel button, and the Exit accelerator key (F3).

If the user presses the F1 key, the currently defined HELP handler will be executed. If the user presses the F3 key or the Cancel button, the currently defined EXIT handler will be executed. If you have not defined custom HELP or EXIT handlers using the Handler function, the default handlers will be executed. The default EXIT handler displays the Exit dialog box. The default HELP handler does nothing.

To execute custom handlers defined using Handler, InstallShield calls a unique label specified in the parameter Label when the nObject event occurs. When InstallShield reaches a return statement in the handler code (under the label), control returns to the statement that would have executed next if the handler label were not called.

Using Handler, you can specify custom handling of EXIT or HELP. You can display the MessageBox, SprintfBox, and AskYesNo dialog boxes inside Exit handlers; however, you cannot display Sd dialog boxes.

Parameters

nObject

Specifies the event to trap. Pass one of the following predefined constants in this parameter:

EXIT

Specifies that a custom handler is called when the Cancel button or the F3 accelerator key is pressed. If you do not define a handler, a default Exit dialog box appears when the Cancel button or F3 accelerator key is pressed.

HELP

Specifies that a custom handler is called when the F1 accelerator key is pressed. If you do not define a handler, nothing happens when the F1 accelerator key is pressed.

Label

Specifies the name of the label to which the program should jump if the specified button or accelerator key is pressed. Do not define this label as a numeric value or as a string variable.

To cancel a currently defined handler and reinstall the default handler, pass -1 in this parameter. This method is useful in scripts that install a custom handler only for certain processes and then revert to the default handler.

Return values

0

Handler successfully created the handle.

< 0

Handler was unable to create the handle.

Comments

- n Do not define the label name as a numeric value or as a string variable.
- n The accelerators available are the F1 function key (Help) and the F3 function key (Exit), which gives the same result as the Cancel button in script dialogs.
- n Help (F1) allows you to launch the Help Engine or provide other suitable Help. When the user presses the F1 key, InstallShield calls the currently defined Help handler. You can provide any type of Help functionality in the Help handler. If you want to provide context sensitive Help, you must keep track of the context in your script. For example, you can have a string variable that contains a current context string. You can then use that string variable in a switch-case statement inside the Help handler to execute the appropriate Help events

based on the value of the context string. You can also test the value of the nSdDialog global variable in your Help handling routine. nSdDialog is set to the dialog ID (as described in Sdrc.h in the InstallShield Include folder) of the currently executing Sd dialog. (When no Sd dialog is executing, nSdDialog is not defined.) You can therefore give the user access to Sd dialog-specific Help when an Sd dialog is executing.

- n As with Help handlers, you can also define and execute custom and Sd dialog-specific EXIT handlers.
- n In the script, EXIT and HELP handler labels and their associated code must be placed before the endprogram statement.
- n For examples of custom HELP and EXIT handlers, see the example script for the Handler function in the online Language Reference help.

{button ,JI('LANGREF.HLP>Examples',`Handler_example')}} [Example](#)

{button ,AL(`Do',0,`,`')}} [See also](#)

{button ,PI("",`comment')}} [Feedback](#)



Handler example

```
/*-----*\
*
* This example illustrates the use of the Handler function.
*
\*-----*/

program

/*-----*\
*
* The following sets the exit and help handlers to specific constants to jump
* to later in the script.
*
\*-----*/
    Handler(EXIT, Exit_Handler);
    Handler(HELP, Help_Handler);

    MessageBox("After pressing OK, type F3 to Exit or F1 for Help.",
                INFORMATION);
    Delay(30);

    MessageBox("Time has passed.", WARNING);

    abort;

    Help_Handler:
        MessageBox("Display help here.", INFORMATION);
        abort;
        return;

    Exit_Handler:
        MessageBox("Setup will now exit.", INFORMATION);
        abort;
        return;

endprogram

// Source file: Is5fn089.rul
```

{button ,PI("",`comment')} [Feedback](#)



MessageBeep

Syntax

MessageBeep (nReserved);

Description

The MessageBeep function sounds a standard warning beep.

Parameters

nReserved

Pass zero in this parameter. No other value is allowed.

Return values

This function has no return values.

Comment

You can also give audio cues by calling [PlayMMedia](#) to play an audio file.

{button ,JI(^LANGREF.HLP>Examples',`MessageBeep_example')}} [Example](#)

{button ,AL(`MessageBox',0,'')} [See also](#)

{button ,PI("",`comment')}} [Feedback](#)



MessageBeep example

```
/*-----*\
 *
 * This example illustrates the MessageBeep function.
 *
 \*-----*/

program

/*-----*\
 *
 * MessageBeep sounds a beep before and after creating a message box.
 *
 \*-----*/

    MessageBeep(0);

    MessageBox("Please listen for a sound.", INFORMATION);

    MessageBeep(0);

endprogram

// Source file: Is5fn231.rul
```

{button ,PI("",`comment')} Feedback



Sprintf

Syntax

```
Sprintf (svResult, szFormat [,arg] [...]);
```

Description

The Sprintf function creates a string from variable data using format specifier and matching variables. The Sprintf function works like the Microsoft Windows API wsprintf.

Parameters

svResult

Returns a string created from the arguments passed in the third and subsequent parameters and formatted according the specifications in the second parameter, szFormat.

szFormat

Specifies a string than can include literal text and must include one format specifier for each argument to be embedded in the string returned by svResult.

arg

You may specify up to nine arguments to be included in the message. You must have one argument for each format specifier in the message; the type of each argument must match the type of its respective format specifier.

Sprintf will generate a compiler error or fail at run time under the following conditions:

- n More than ten format specifiers and arguments are specified: compiler error.
- n The number of arguments does not match the number of format specifications. Sprintf will return a null (empty) string in svResult.
- n A variable does not match the type of its respective format specifier. Sprintf will return a null (empty) string in svResult.

Return values

If the Sprintf function is successful, the return value is the length (the number of characters) in the string stored in the variable svResult, not including the terminating null character.

{button ,JI('LANGREF.HLP>Examples','Sprintf_example')} [Example](#)

{button ,AL('MessageBox;SprintfBox',0,','')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



Sprintf example

```
/*-----*\
 *
 * This example illustrates the uses of the Sprintf function.
 *
 * The script calls the Sprintf function to format a string with a string and
 * number data type added to it.
 *
 \*-----*/

    STRING  szStr,svResult;
    NUMBER  nNum, nResult;

program

    szStr = "This is a sample text string";
    nNum = 420;

/*-----*\
 *
 * The following combines variables and text to make a valid string.
 *
 \*-----*/

    Sprintf(svResult, "Sprintf is used to add strings-->'%s', " +
        "Numbers-->'%d', and other data types to strings.", szStr, nNum);
    MessageBox(svResult, INFORMATION);

endprogram

// Source file: Is5fn171.rul
```

{button ,PI("",`comment')} [Feedback](#)



System

Syntax

System (nOp);

Description

Use the System function to restart Windows or reboot the system. The System function does not perform an aborted setup (i.e., it does not remove installed files). However, InstallShield does remove any temporary directories and temporary files it placed on the system to carry out the setup. This function has no effect when used with 16-bit InstallShield on Windows NT.



The System function is documented for backward compatibility with earlier releases of InstallShield. With the current release of InstallShield, the best functions to use to restart Windows or reboot the system are [RebootDialog](#) and [SdFinishReboot](#). Of these two, SdFinishReboot provides the most functionality. Refer to the individual function descriptions to see which one best meets your needs.

Parameters

nOp

Specifies which action to perform after terminating the setup. Pass one of the following predefined constants in this parameter:

SYS_BOOTMACHINE

Reboots the system.

SYS_BOOTWIN

When using 16-bit InstallShield on Windows 3.1 systems, restarts Windows. On all other operating systems and when using 32-bit InstallShield, reboots the system.

SYS_TODOS

When using 16-bit InstallShield on Windows 3.1 systems, exits Windows to DOS. On Windows 95 when using 16-bit InstallShield reboots the system. On other operating systems, does nothing.

Return values

< 0

Indicates that the function was unable to terminate the setup program in the requested manner.

Comments

- n This function calls the Microsoft Windows API ExitWindows. Due to the wide variety of BIOS types in systems today, this function is highly dependent on the BIOS interaction with the system.
- n Some systems may not restart or may hang when this function is called. Many setup routines (including installation for system software such as MS-DOS) display a warning message to the user before they restart the system. This warning message indicates what is happening and instructs the user to reboot the system manually if the command fails.
- n To ensure that .dll and .exe files that were locked get updated properly, you must call the [CommitSharedFiles](#) function before calling System. When InstallShield file transfer functions that use the SHAREDFILE option encounter locked .dll or .exe files, the system variable BATCH_INSTALL is set to TRUE and the files are transferred to the target system with unique names. You must call CommitSharedFiles before restarting Windows or the system in order for the locked files to be updated.

{button ,JI('LANGREF.HLP>Examples',`System_example`)} [Example](#)

```
{button ,AL(' CommitSharedFiles;GetSystemInfo;RebootDialog;SdFinish;SdFinishReboot;VerUpdateFile',0,'')}
```

See also

```
{button ,PI('','comment')} Feedback
```



System example

```
/*-----*\
 *
 * This example illustrates the usage of the System function.
 *
 \*-----*/

program

/*-----*\
 *
 * This call to System reboots Windows (Windows 3.1) or logs off the current
 * Windows user (Windows 95).
 *
 \*-----*/

    System(SYS_BOOTWIN);

endprogram

// Source file: Is5fn268.rul
```

{button ,PI("",`comment')} [Feedback](#)



VarRestore

Syntax

VarRestore (nType);

Description

The VarRestore function restores the system variables SRCDIR and TARGETDIR that were saved by the [VarSave](#) function.

InstallShield uses the system variables SRCDIR and TARGETDIR extensively. Many functions use these variables as the source and target directories for the operations they perform. Therefore, you may need to save the current value of the system variables SRCDIR and TARGETDIR, change their values temporarily, and restore their values back to original values.

VarSave and VarRestore functions are a convenient way of saving and restoring the value of the system variables SRCDIR and TARGETDIR.

Advanced developers may recognize these as push and pop operations.

Parameters

nType

Specifies the constant SRCTARGETDIR to specify that you want InstallShield to restore the previously saved values of the system variables SRCDIR and TARGETDIR.

Return values

0

Indicates the number of values that will be in storage after this function is called.

< 0

Indicates that no values in storage can be restored. This error condition occurs if you call VarRestore without calling VarSave first, or you call VarRestore more times than you call VarSave.

{button ,JI('LANGREF.HLP>Examples','VarRestore_example')} [Example](#)

{button ,AL('VarSave',0,'')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



VarRestore example

```
/*-----*\
 *
 * This example illustrates the uses of the VarSave and VarRestore functions.
 *
 *-----*/

#define NEW_SOURCE_DIR "EXAMPLE\\SOURCE"
#define NEW_TARGET_DIR "EXAMPLE\\TARGET"

program

    // Print out initial directory variables
    SprintfBox (INFORMATION, "System Directories", "Source: %s\n" +
                "Target: %s ", SRCDIR, TARGETDIR);

/*-----*\
 *
 * Save the variables.
 *
 *-----*/

    VarSave (SRCTARGETDIR);

    // Change to new directories
    SRCDIR = NEW_SOURCE_DIR;
    TARGETDIR = NEW_TARGET_DIR;
    SprintfBox (INFORMATION, "New System Directories", "New Source: %s\n" +
                "New Target: %s", SRCDIR, TARGETDIR);

/*-----*\
 *
 * Restore the old variables.
 *
 *-----*/

    VarRestore (SRCTARGETDIR);

    SprintfBox (INFORMATION, "Old System Directories", "Old Source: %s\n" +
                "Old Target: %s", SRCDIR, TARGETDIR);

endprogram

// Source file Is5fn181.rul
```

{button ,PI("",`comment')} [Feedback](#)



VarSave

Syntax

VarSave (nType);

Description

The VarSave function saves the current value of the system variables SRCDIR and TARGETDIR.

InstallShield uses the system variables SRCDIR and TARGETDIR extensively. Many functions use these variables as the source and target directories for the operations they perform. Therefore, you may need to save the current value of the system variables SRCDIR and TARGETDIR, change their values temporarily, and restore their values back to original values.

VarSave and [VarRestore](#) provide a convenient way of saving and restoring the value of the system variables SRCDIR and TARGETDIR.

Advanced developers may recognize these as push and pop operations.

Parameters

nType

Specifies the constant SRCTARGETDIR to specify that you want to save the current values of the system variables SRCDIR and TARGETDIR.

Return values

0

Indicates the number of values currently saved.

< 0

Indicates that the system variables SRCDIR and TARGETDIR were not saved due to an internal error. The main cause of this error is insufficient available memory (which will seldom occur).

{button ,JI('LANGREF.HLP>Examples','VarSave_example')} [Example](#)

{button ,AL('VarRestore',0,'')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



VarSave example

```
/*-----*\
 *
 * This example illustrates the uses of the VarSave and VarRestore functions.
 *
 \*-----*/

#define NEW_SOURCE_DIR "EXAMPLE\\SOURCE"
#define NEW_TARGET_DIR "EXAMPLE\\TARGET"

program

    // Print out initial directory variables
    SprintfBox (INFORMATION, "System Directories", "Source: %s\n" +
                "Target: %s ", SRCDIR, TARGETDIR);

/*-----*\
 *
 * Save the variables.
 *
 \*-----*/

    VarSave (SRCTARGETDIR);

    // Change to new directories
    SRCDIR = NEW_SOURCE_DIR;
    TARGETDIR = NEW_TARGET_DIR;
    SprintfBox (INFORMATION, "New System Directories", "New Source: %s\n" +
                "New Target: %s", SRCDIR, TARGETDIR);

/*-----*\
 *
 * Restore the old variables.
 *
 \*-----*/

    VarRestore (SRCTARGETDIR);

    SprintfBox (INFORMATION, "Old System Directories", "Old Source: %s\n" +
                "Old Target: %s", SRCDIR, TARGETDIR);

endprogram

// Source file Is5fn181.rul
```

{button ,PI("",`comment')} [Feedback](#)



Path buffer functions

The path buffer functions are available to assist you in working with strings that contain search paths. The Path String functions work on a unique temporary string variable known as the path buffer. The path buffer is defined internally within InstallShield; all the Path String functions act on the contents of the path buffer.



These functions do not support long filenames. In a 32-bit setup, call [LongPathToShortPath](#) to convert the long path to its short path equivalent before passing it to a path buffer function.

Path functions assist you in manipulating and building path strings. Once you have created a path string, you can save it to the appropriate file.

PathAdd

Adds a path to the search path in the path buffer.

PathDelete

Deletes a directory from the path buffer.

PathFind

Finds a specific path in the path buffer or any path that includes a specified name.

PathGet

Retrieves the current value of the path buffer.

PathMove

Rearranges the path buffer.

PathSet

Assigns a value to the path buffer.

{button ,PI("",`comment')} [Feedback](#)



PathAdd

Syntax

PathAdd (szDir, szRefDir, bRefDir, bPosition);

Description

The PathAdd function adds a path to the search path in the path buffer. With this function you can specify the position of the directory in relation to an existing directory in the path buffer. In addition, you can add the directory as the first or the last directory of the path buffer.

This function has no relationship to the path statement in the Autoexec.bat file or the path environment variable. It acts only on the path buffer, which helps you build, modify, and manipulate search paths. You can then add the modified path string to the Autoexec.bat file using the various Batch file functions.



This function does not support long filenames. In a 32-bit setup, call [LongPathToShortPath](#) to convert the long path to its short path equivalent before passing it to PathAdd.

Parameters

szDir

Specifies a path to add to the path buffer.

szRefDir

Specifies the path in the current path buffer relative to which the new path will be added.

bRefDir

Specifies whether or not szRefDir is a fully qualified path. Pass one of the following predefined constants in this parameter:

FULL

szRefDir is a fully qualified path (that is, it includes a drive designation and the complete path to a directory.)

PARTIAL

szRefDir is the directory name only (without drive or path information).

bPosition

Specifies the position relative to szRefDir at which szDir is to be inserted. Pass one of the following predefined constants in this parameter:

AFTER

Specifies that szDir is to be inserted after szRefDir. If szRefDir specifies a null string (""), szDir is inserted at the end of the path in the path buffer.

BEFORE

Specifies that szDir is to be inserted before szRefDir. If szRefDir specifies a null string (""), szDir is inserted at the front of the path in the path buffer.

Return values

0

Indicates that the function successfully added a directory to the path buffer.

< 0

Indicates that the function was unable to add a directory to the path buffer.

Comments

If you specify a directory in szDir that currently exists in the path buffer, InstallShield will not duplicate it, and the position of the pre-existing directory is not modified. InstallShield ignores a backslash at the end of the directory name.

{button ,JI('LANGREF.HLP>Examples','PathAdd_example')} Example

{button ,AL('Path buffer fn',0,'')} See also

{button ,PI('','comment')} Feedback



PathAdd example

```
/*-----*\
 *
 * This example illustrates the uses of the PathAdd and PathMove functions.
 *
 *-----*/

    STRING szDir, szRefDir, svString;
    BOOL   bDir, bRefDir, bPosition;

program

    PathSet("C:\\DOS;C:\\WINDOWS;C:\\TEMP");

    szDir   = "C:\\MSOFFICE";
    szRefDir = "";
    bRefDir  = FULL;
    bPosition = BEFORE;
/*-----*\
 *
 * Add C:\\MSOFFICE before C:\\DOS in the path buffer.
 *
 *-----*/
    if (PathAdd(szDir, szRefDir, bRefDir, bPosition) < 0) then

        MessageBox("PathAdd failed.", SEVERE);
        abort;
    endif;

    szDir   = "C:\\APP2";
    szRefDir = "TEMP";
    bRefDir  = PARTIAL;
    bPosition = BEFORE;
/*-----*\
 *
 * Add C:\\APP2 before the C:\\TEMP directory in the path buffer.
 *
 *-----*/
    if (PathAdd(szDir, szRefDir, bRefDir, bPosition) < 0) then

        MessageBox("PathAdd failed.", SEVERE);
        abort;
    endif;

    PathGet(svString);

    // svString will contain C:\\MSOFFICE;C:\\DOS;C:\\WINDOWS;C:\\APP2;C:\\TEMP;.
    sprintfBox(INFORMATION, "Path Get", "Path is: %s", svString);

    // Need to reset path after using PathGet
    PathSet(svString);
/*-----*\
 *
 * Reposition C:\\APP2 before C:\\DOS.
 *
 *-----*/
    if (PathMove("C:\\APP2", "C:\\DOS", FULL, FULL, BEFORE) < 0) then

        MessageBox("PathMove failed.", SEVERE);
```

```
        abort;
    endif;

/*-----*\
*
* Reposition C:\DOS after C:\TEMP.
*
/*-----*/
    if (PathMove("C:\\DOS", "TEMP", FULL, PARTIAL, AFTER) < 0) then

        MessageBox("PathMove failed.", SEVERE);
    endif;

    // Retrieve the full path from the buffer
    PathGet(svString);

    // Display the full path from the buffer.
    // svString will contain C:\MSOFFICE;C:\APP2;C:\WINDOWS;C:\TEMP;C:\DOS
    SprintfBox(INFORMATION, "Path Get", "Path is: %s", svString);

endprogram

// Source file: Is5fn116.rul
```

{button ,PI("",`comment')} Feedback



PathDelete

Syntax

PathDelete (szDir, bDir);

Description

The PathDelete function deletes a specific directory in the path buffer. You can specify the name of the directory or enter a fully qualified path.

This function has no relationship to the path statement in the Autoexec.bat file or the path environment variable. It acts only on the path buffer, which helps you build, modify, and manipulate search paths.

Call [PathGet](#) to get the contents of the path buffer; call [PathSet](#) to set contents of the path buffer.



This function does not support long filenames. In a 32-bit setup, call [LongPathToShortPath](#) to convert the long path to its short path equivalent before passing it to PathDelete.

Parameters

szDir

Specifies the path to remove from the path buffer.

bDir

Specifies whether or not szRefDir is a fully qualified path. Pass one of the following predefined constants in this parameter:

FULL

szRefDir is a fully qualified path (that is, it includes a drive designation and the complete path to a directory.)

PARTIAL

szRefDir is the directory name only (without drive or path information).

Return values

0

Indicates that the function successfully deleted a directory from the path buffer.

< 0

Indicates that the function was unable to delete a directory from the path buffer.

{button ,JI('LANGREF.HLP>Examples','PathDelete_example')} [Example](#)

{button ,AL('Path buffer fn',0,'')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



PathDelete example

```
/*-----*\
 *
 * This example illustrates the usage of the PathDelete function.
 *
 * PathDelete is called twice. The first time to delete a path in the buffer
 * by searching for the directory name only.
 *
 * The second time PathDelete is called to delete a path by searching for the
 * fully qualified directory name.
 *
 \*-----*/

    STRING svString, szDir, szTitle;
    BOOL bDir;

program

    svString = "C:\\DOS;C:\\WINDOWS;C:\\TEMP;C:\\EXAMPLE\\SOURCE";
    PathSet(svString);

    szDir = "TEMP";
    bDir = PARTIAL;
/*-----*\
 *
 * The following deletes a path with the directory name 'TEMP'.
 *
 \*-----*/
    if (PathDelete(szDir, bDir) < 0) then

        MessageBox("First call to PathDelete failed.", SEVERE);
    endif;

    // Get the full path from the buffer
    PathGet(svString);

    szTitle = "PathDelete Example";
    // The following will display the path buffer:
    // 'C:\DOS;C:\WINDOWS;C:\EXAMPLE\SOURCE'.
    sprintfBox(INFORMATION, szTitle, "Path is: %s", svString);

    // Set the path after PathGet was called.
    PathSet(svString);

    szDir = "C:\\EXAMPLE\\SOURCE";
    bDir = FULL;
/*-----*\
 *
 * The following deletes a path with the fully qualified directory name
 * 'C:\\EXAMPLE\\SOURCE'.
 *
 \*-----*/
    if (PathDelete(szDir, bDir) < 0) then

        MessageBox("Second call to PathDelete failed.", SEVERE);
    endif;

    // Get the full path from the buffer again.
    PathGet(svString);
    // The following will display the new path buffer: 'C:\DOS;C:\WINDOWS'.
```

```
    SprintfBox(INFORMATION, szTitle, "Path is: %s", svString);  
endprogram  
// Source file: Is5fn117.rul
```

```
{button ,PI("",`comment')} Feedback
```



PathFind

Syntax

PathFind (szDir, svResult, bDir, bSearch);

Description

The PathFind function searches the path buffer for a specific directory. You can specify the directory with either a fully qualified path or the directory name only.

This function has no relationship to the path statement in the Autoexec.bat file or the path environment variable. It acts only on the path buffer, which helps you build, modify, and manipulate search paths. You can then add this temporary path string to the Autoexec.bat file using the various Batch file functions.



This function does not support long filenames. In a 32-bit setup, call [LongPathToShortPath](#) to convert the long path to its short path equivalent before passing it to PathFind.

Parameters

szDir

Specifies the path to find in the path buffer.

svResult

Returns the full directory and path found in the path buffer returned by the function.

bDir

Specifies whether or not szDir contains a fully qualified or an unqualified directory name. Pass one of the following predefined constants in this parameter:

FULL

Specifies szDir contains a fully qualified directory name.

PARTIAL

Specifies szDir contains the directory name only.

bSearch

Specifies where to begin the search. Pass one of the following predefined constants in this parameter.

CONTINUE

Continues searching from where the previous search was terminated in the path buffer.

RESTART

Starts the search from the beginning of the path buffer.

Return values

0

Indicates that the function successfully searched the path buffer for a directory.

< 0

Indicates that the function was unable to successfully search the path buffer for a directory.

{button ,JI(`LANGREF.HLP>Examples',`PathFind_example')} [Example](#)

{button ,AL(`Path buffer fn',0,`,`')} [See also](#)

{button ,PI(`",`comment')} [Feedback](#)



PathFind example

```
/*-----*\
 *
 * This example illustrates the usage of the PathSet, PathFind, and PathGet
 * functions.
 *
 * PathSet is called to set the path buffer to the szString string.
 *
 * PathFind is then called to loop through this string, displaying all
 * occurrences of the szDir string.
 *
 * PathGet is called to return the path buffer into the svString string.
 *
 \*-----*/

    STRING szString, szMsg, svResult, svString, szDir;
    NUMBER nResult;
    BOOL   bDir, bSearch;

program

    // Set the string for PathSet function.
    szString = "C:\\DOS;C:\\USERS\\BIN;C:\\MSC\\BIN;";

/*-----*\
 *
 * Set the buffer to contain the string path.
 *
 \*-----*/
    if (PathSet(szString) < 0) then

        MessageBox("PathSet failed.", SEVERE);
    else

        szMsg = "PathSet set the temporary path buffer to: %s";
        sprintfBox(INFORMATION, "PathSet Example", szMsg, szString);
    endif;

    // Set PathFind variables.
    szDir   = "BIN";
    bDir    = PARTIAL;
    bSearch = RESTART;

/*-----*\
 *
 * Search the buffer to find a string. Return the result into the svResult
 * string. Loop until all string have been displayed.
 *
 \*-----*/
    nResult = PathFind(szDir, svResult, bDir, bSearch);

    // Error check PathFind.
    if (nResult < 0) then

        MessageBox("PathFind failed.", SEVERE);
        abort;
    endif;

    // Loop through the string to find all occurrences of the szDir string.
    while (nResult = 0)
```

```
        sprintfBox(INFORMATION, "PathFind example",
                  "Search for %s.\n\nFound:  %s", szDir, svResult);

        bSearch = CONTINUE;
        nResult = PathFind(szDir, svResult, PARTIAL, bSearch);
    endwhile;

/*-----*\
 *
 * Get the path from the buffer.
 *
\*-----*/
    if (PathGet(svString) < 0) then

        MessageBox("PathGet failed.", SEVERE);
    endif;

    // Display the path string.
    sprintfBox(INFORMATION, "Path Get Example" ,"Path is: %s", svString);

endprogram

// Source file: Is5fn118.rul
```

{button ,PI("",`comment')} Feedback



PathGet

Syntax

PathGet (svString);

Description

The PathGet function retrieves the path string currently stored in the temporary path string buffer. You can use Path functions to manipulate the path buffer. The PathGet function is used to retrieve the path string.

This function has no relationship to the path statement in the Autoexec.bat file or the path environment variable. It acts on the path buffer only, which helps you build, modify, and manipulate search paths. You can then add this temporary path string to the Autoexec.bat file using the various Batch file functions.



This function does not support long filenames. In a 32-bit setup, call [LongPathToShortPath](#) to convert the long path to its short path equivalent before passing it to PathGet.

Parameters

svString

Returns the contents of the path buffer.

Return values

0

Indicates that the function successfully retrieved the path currently stored in the path buffer.

< 0

Indicates that the function was unable to retrieve the path string currently stored in the temporary path string buffer.

{button ,JI('LANGREF.HLP>Examples','PathGet_example')} [Example](#)

{button ,AL('Path buffer fn',0,',')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



PathGet example

```
/*-----*\
 *
 * This example illustrates the usage of the PathSet, PathFind, and PathGet
 * functions.
 *
 * PathSet is called to set the path buffer to the szString string.
 *
 * PathFind is then called to loop through this string, displaying all
 * occurrences of the szDir string.
 *
 * PathGet is called to return the path buffer into the svString string.
 *
\*-----*/

    STRING szString, szMsg, svResult, svString, szDir;
    NUMBER nResult;
    BOOL    bDir, bSearch;

program

    // Set the string for PathSet function.
    szString = "C:\\DOS;C:\\USERS\\BIN;C:\\MSC\\BIN;";

/*-----*\
 *
 * Set the buffer to contain the string path.
 *
\*-----*/
    if (PathSet(szString) < 0) then

        MessageBox("PathSet failed.", SEVERE);
    else

        szMsg = "PathSet set the temporary path buffer to: %s";
        sprintfBox(INFORMATION, "PathSet Example", szMsg, szString);
    endif;

    // Set PathFind variables.
    szDir    = "BIN";
    bDir     = PARTIAL;
    bSearch  = RESTART;

/*-----*\
 *
 * Search the buffer to find a string. Return the result into the svResult
 * string. Loop until all string have been displayed.
 *
\*-----*/
    nResult = PathFind(szDir, svResult, bDir, bSearch);

    // Error check PathFind.
    if (nResult < 0) then

        MessageBox("PathFind failed.", SEVERE);
        abort;
    endif;

    // Loop through the string to find all occurrences of the szDir string.
    while (nResult = 0)
```

```
        sprintfBox(INFORMATION, "PathFind example",
                  "Search for %s.\n\nFound:  %s", szDir, svResult);

        bSearch = CONTINUE;
        nResult = PathFind(szDir, svResult, PARTIAL, bSearch);
    endwhile;

/*-----*\
 *
 * Get the path from the buffer.
 *
\*-----*/
    if (PathGet(svString) < 0) then

        MessageBox("PathGet failed.", SEVERE);
    endif;

    // Display the path string.
    sprintfBox(INFORMATION, "Path Get Example" ,"Path is: %s", svString);

endprogram

// Source file: Is5fn118.rul
```

{button ,PI("",`comment')} Feedback



PathMove

Syntax

PathMove (szDir, szRefDir, bDir, bRefDir, bPosition);

Description

The PathMove function repositions a directory in the path buffer to another location. You can also use this function to position the directory relative to another directory or as the first or the last item in the path string.

Call [PathGet](#) to get the contents of the path buffer; call [PathSet](#) to set contents of the path buffer.

This function has no relation to the PATH statement in the Autoexec.bat file or the PATH environment variable. It acts only on the path buffer, which helps you build, modify, and manipulate search paths. You can then add this temporary path string to Autoexec.bat using the various Batch file functions.



This function does not support long filenames. In a 32-bit setup, call [LongPathToShortPath](#) to convert the long path to its short path equivalent before passing it to PathMove.

Parameters

szDir

Specifies a full or partial path to reposition in the path buffer.

szRefDir

Specifies the path in the path buffer relative to which the path in szDir will be moved. To move the path in szDir to the beginning or end of the path in the path buffer, pass a null string ("") in this parameter.

bDir

Use this parameter to specify whether szDir contains a fully qualified or an unqualified directory name. These constants are available:

FULL

Specifies that szDir contains a fully qualified directory name.

PARTIAL

Specifies that szDir contains the directory name only.

bRefDir

Use this parameter to specify whether szRefDir contains a fully qualified or an unqualified directory name. These constants are available:

FULL

Specifies that szRefDir contains a fully qualified directory name.

PARTIAL

Specifies that szRefDir contains the directory name only.

bPosition

Specifies the position relative to szRefDir to which szDir is to be moved. Pass one of the following predefined constants in this parameter:

AFTER

Specifies that szDir is to be positioned after szRefDir. If szRefDir specifies a null string (""), szDir is positioned at the end of the path in the path buffer.

BEFORE

Specifies that szDir is to be positioned before szRefDir. If szRefDir specifies a null string (""), szDir is positioned at the front of the path in the path buffer.

Return values

0

Indicates that the function successfully repositioned a directory in the path buffer.

< 0

Indicates that the function was unable to reposition a directory in the path buffer.

{button ,JI('LANGREF.HLP>Examples','PathMove_example')} Example

{button ,AL('Path buffer fn',0,'')} See also

{button ,PI('','comment')} Feedback



PathMove example

```
/*-----*\
 *
 * This example illustrates the uses of the PathAdd and PathMove functions.
 *
 *-----*/

    STRING szDir, szRefDir, svString;
    BOOL   bDir, bRefDir, bPosition;

program

    PathSet("C:\\DOS;C:\\WINDOWS;C:\\TEMP");

    szDir    = "C:\\MSOFFICE";
    szRefDir = "";
    bRefDir  = FULL;
    bPosition = BEFORE;
/*-----*\
 *
 * Add C:\MSOFFICE before C:\DOS in the path buffer.
 *
 *-----*/
    if (PathAdd(szDir, szRefDir, bRefDir, bPosition) < 0) then

        MessageBox("PathAdd failed.", SEVERE);
        abort;
    endif;

    szDir    = "C:\\APP2";
    szRefDir = "TEMP";
    bRefDir  = PARTIAL;
    bPosition = BEFORE;
/*-----*\
 *
 * Add C:\APP2 before the C:\TEMP directory in the path buffer.
 *
 *-----*/
    if (PathAdd(szDir, szRefDir, bRefDir, bPosition) < 0) then

        MessageBox("PathAdd failed.", SEVERE);
        abort;
    endif;

    PathGet(svString);

    // svString will contain C:\MSOFFICE;C:\DOS;C:\WINDOWS;C:\APP2;C:\TEMP;.
    sprintfBox(INFORMATION, "Path Get", "Path is: %s", svString);

    // Need to reset path after using PathGet
    PathSet(svString);
/*-----*\
 *
 * Reposition C:\APP2 before C:\DOS.
 *
 *-----*/
    if (PathMove("C:\\APP2", "C:\\DOS", FULL, FULL, BEFORE) < 0) then

        MessageBox("PathMove failed.", SEVERE);
```

```
        abort;
    endif;

/*-----*\
*
* Reposition C:\DOS after C:\TEMP.
*
*-----*/
    if (PathMove("C:\\DOS", "TEMP", FULL, PARTIAL, AFTER) < 0) then

        MessageBox("PathMove failed.", SEVERE);
    endif;

    // Retrieve the full path from the buffer
    PathGet(svString);

    // Display the full path from the buffer.
    // svString will contain C:\MSOFFICE;C:\APP2;C:\WINDOWS;C:\TEMP;C:\DOS
    sprintfBox(INFORMATION, "Path Get", "Path is: %s", svString);

endprogram

// Source file: Is5fn116.rul
```

{button ,PI("",`comment')} Feedback



PathSet

Syntax

```
PathSet (szString);
```

Description

The PathSet function stores a search path string in the path buffer. You can then manipulate this buffer using the other path functions. The value of szString should be an [absolute path](#).

This function has no relation to the PATH statement in the Autoexec.bat file or the PATH environment variable. It acts only on the path buffer, which helps you build, modify, and manipulate search paths. You can then add this temporary path string to the Autoexec.bat file or PATH environment variable.



This function does not support long filenames. In a 32-bit setup, call [LongPathToShortPath](#) to convert the long path to its short path equivalent before passing it to PathSet.

Parameters

szString

Specifies a search path to store in the path buffer. This path must contain a drive letter, a colon, and a root directory (a backslash).

Return values

0

Indicates that the function successfully stored a search path string in the path buffer.

< 0

Indicates that the function was unable to store a search path string in the path buffer.

{button ,JI('LANGREF.HLP>Examples','PathSet_example')} [Example](#)

{button ,AL('Path buffer fn',0,',')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



PathSet example

```
/*-----*\
 *
 * This example illustrates the usage of the PathSet, PathFind, and PathGet
 * functions.
 *
 * PathSet is called to set the path buffer to the szString string.
 *
 * PathFind is then called to loop through this string, displaying all
 * occurrences of the szDir string.
 *
 * PathGet is called to return the path buffer into the svString string.
 *
 \*-----*/

    STRING szString, szMsg, svResult, svString, szDir;
    NUMBER nResult;
    BOOL   bDir, bSearch;

program

    // Set the string for PathSet function.
    szString = "C:\\DOS;C:\\USERS\\BIN;C:\\MSC\\BIN;";

/*-----*\
 *
 * Set the buffer to contain the string path.
 *
 \*-----*/

    if (PathSet(szString) < 0) then

        MessageBox("PathSet failed.", SEVERE);
    else

        szMsg = "PathSet set the temporary path buffer to: %s";
        sprintfBox(INFORMATION, "PathSet Example", szMsg, szString);
    endif;

    // Set PathFind variables.
    szDir   = "BIN";
    bDir    = PARTIAL;
    bSearch = RESTART;

/*-----*\
 *
 * Search the buffer to find a string. Return the result into the svResult
 * string. Loop until all string have been displayed.
 *
 \*-----*/

    nResult = PathFind(szDir, svResult, bDir, bSearch);

    // Error check PathFind.
    if (nResult < 0) then

        MessageBox("PathFind failed.", SEVERE);
        abort;
    endif;

    // Loop through the string to find all occurrences of the szDir string.
    while (nResult = 0)
```

```

        sprintfBox(INFORMATION, "PathFind example",
            "Search for %s.\n\nFound:  %s", szDir, svResult);

        bSearch = CONTINUE;
        nResult = PathFind(szDir, svResult, PARTIAL, bSearch);
    endwhile;

/*-----*\
 *
 * Get the path from the buffer.
 *
\*-----*/
    if (PathGet(svString) < 0) then

        MessageBox("PathGet failed.", SEVERE);
    endif;

    // Display the path string.
    sprintfBox(INFORMATION, "Path Get Example" ,"Path is: %s", svString);

endprogram

// Source file: Is5fn118.rul

```

{button ,PI("",`comment')} [Feedback](#)



Registry functions

The following functions allow you to access the registry, read, create and delete registry keys, and establish registry-related parameters for uninstallation.

CreateRegistrySet

Creates one of the sets of registry entries specified in the Resources pane's Registry Entries folder.

DeinstallSetReference

Specifies a reference file to be checked before the uninstallation process begins.

DeinstallStart

Creates the application uninstallation key and sets the [UninstallString] value under that key.

InstallationInfo

Creates registry keys based on a company name, product name, and product version number.

RegDBConnectRegistry

Opens a connection to a remote registry.

RegDBCreateKeyEx

Creates a key in the registry. Also allows you to associate a class object with a registry key (advanced users only).

RegDBDeleteKey

Deletes the specified key from the registry.

RegDBDeleteValue

Deletes a value from a specified registry key.

RegDBDisconnectRegistry

Closes the connection to a remote registry.

RegDBGetAppInfo

Retrieves a value from under the application information key.

RegDBGetItem

Retrieves values under the per-application paths key or the application uninstallation key.

RegDBGetKeyValueEx

Retrieves a value from under a key in the registry.

RegDBKeyExist

Checks that a registry key exists.

RegDBQueryKey

Queries a key for its subkeys and value names.

RegDBSetAppInfo

Sets a value under the application information key.

RegDBSetDefaultRoot

Sets the root key.

RegDBSetItem

Assign values under the per application paths key or the application uninstallation key.

RegDBSetKeyValueEx

Sets registry entries.

{button ,AL(`Overview the registry',0,`,`')} [See also](#)

{button ,PI(`",`comment')} [Feedback](#)



CreateRegistrySet

Syntax

```
CreateRegistrySet (szRegistrySet );
```

Description

The CreateRegistrySet function creates the registry information specified by one or all registry sets in the current media. The name of the current media is stored in the system variable MEDIA. A registry set can be associated with a file group so that a call to CreateRegistrySet creates the registry set only if the file group has been installed to the target system. For details, click [here](#).

Parameters

szRegistrySet

Specifies the name of a registry set in the current media. To create all registry sets that are defined in the current media, pass a null string ("") in this parameter.

Return values

0

Indicates that the function was successful.

< 0

Indicates that the function failed.

Comments

- n The value of the system variable MEDIA is set to 'DATA' during setup initialization. If you change the value of this variable to refer to a script-created component set, you must change the value back to 'DATA' before calling CreateRegistrySet.
- n This function should be called only after [ComponentMoveData](#) has been called.

{button ,AL(`Overview the registry',0,`,`')} [See also](#)

{button ,PI("",`comment')} [Feedback](#)



DeinstallSetReference

Syntax

DeinstallSetReference (szReferenceFile);

Description

Specifies a reference file to be checked before the uninstallation process begins. If the file is already in use, uninstallation will not proceed. The end user must close the application that is using this file and start uninstallation again. You must call [DeinstallStart](#) before calling DeinstallSetReference.



Due to certain limitations in Windows and Windows applications, it is possible that on rare occasions a file specified by DeinstallSetReference will not be seen as locked; if that happens, the intended uninstallation prevention will fail.

Parameters

szReferenceFile

Specifies the fully qualified name of the file to be checked by unInstallShield before uninstallation begins. This file can be either a program or a DLL that is used only by the application to be uninstalled.

Return values

0

Indicates that the function was successful.

< 0

Indicates that the function failed.

Comments

To specify more than one file to be checked, call this function once for each file. Never specify a file that may be in use by the operating system or another application; if you do, the end user will never be able to continue with uninstallation.

{button ,JI('LANGREF.HLP>Examples','DeinstallSetReference_example')} [Example](#)

{button ,AL('DeinstallStart;CommitSharedFiles;Disable;Enable;InstallationInfo;RegDBGetItem;RegDBSetItem;SdFinishReboot;VerUpdateFile;Prepare my setup for uninstallation;Launch unInstallShield;How NT permissions affect unIS;Uninstalling ini file entries;What gets logged for unis;The uninstallation log file;unInstallShields executable files;OverView UninstallShield;Prevent unIS from removing running application',0,'')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



DeinstallSetReference example

```
/*-----*\
 *
 * This example illustrates the uses of the InstallationInfo,
 * DeinstallStart, and DeinstallSetReference functions.
 *
 * InstallationInfo is called to create an application information key in the
 * registry.
 *
 * DeinstallStart is called to create an uninstallation key in the registry.
 * RegDBSetItem is called to set the [DisplayName] value in the uninstallation
 * key. This value is displayed in the Windows95 and WinNT 4.0 'Add/Remove
 * Programs Properties' dialog.
 *
 * DeinstallSetReference is called to specify a file that, if locked at
 * uninstallation time, prevents uninstallation from continuing.
 *
 * To run properly this example requires at least a single
 * component that installs a file group containing Notepad.exe.
 * If you use an .exe file other than Notepad.exe, be sure to
 * change the PRODUCT_KEY define accordingly.
 *
 * To see the results of DeinstallSetReference, open the Notepad.exe file
 * installed by this example and try to uninstall the example.
 */-----*
#include "sddialog.h"

#define COMPANY_NAME      "ExampleCompany"
#define PRODUCT_NAME      "ExampleApp2"
#define PRODUCT_VERSION   "2.0"
#define PRODUCT_KEY       "Notepad.exe"
#define DEINST_KEY        "ExApp2Uninstall"

prototype HandleComponentError(NUMBER);

STRING  szCompany, szProduct, szVersion, szProductKey;
STRING  svDir, svLogFile, szKey, svPath;
NUMBER  nvDisk, nResult;

program

    // Disable Back button since it is not needed in this example.
    Disable(BACKBUTTON);

    // Initialize variables.
    szCompany    = COMPANY_NAME;
    szProduct    = PRODUCT_NAME;
    szVersion    = PRODUCT_VERSION;
    szProductKey = PRODUCT_KEY;

/*-----*\
 * Call InstallationInfo, which creates the application information key using
 * the values given to it. InstallationInfo also provides information to
 * create the per application paths key.
 */-----*
    InstallationInfo(szCompany, szProduct, szVersion, szProductKey);

    // The Welcome Dialog uses the product name value set by InstallationInfo.
    Welcome("", 0);
```

```

TARGETDIR = PROGRAMFILES ^ COMPANY_NAME;
AskDestPath("Destination Location",
            "Select a destination location.",
            TARGETDIR , 0 );
szKey = DEINST_KEY;

/*-----*\
*
* DeinstallStart initializes the uninstall log file using the values provided
* by InstallationInfo. DeinstallStart also creates the application
* uninstallation key using the values given to it.
*
* NOTE: Call DeinstallStart immediately after InstallationInfo so that there
*       are no intervening dialogs or events that could cause an exit or
*       abort. If an exit or abort occurs before DeinstallStart is called,
*       certain registry entries may not be removed because the uninstall
*       log file has not yet been initialized.
*
\*-----*/
DeinstallStart(TARGETDIR, svLogFile, szKey, 0);
RegDBSetItem(REGDB_UNINSTALL_NAME, PRODUCT_NAME);

// Specify a file that, if locked at uninstallation time will prevent
// uninstallation from continuing.
DeinstallSetReference(TARGETDIR ^ PRODUCT_KEY);

// Transfer files and handle any errors.
SdShowMsg("Installing program files...", TRUE);
nResult = ComponentMoveData ( MEDIA , nvDisk , 0 );
HandleComponentError(nResult);
Delay(1);
SdShowMsg("Installing program files...", FALSE);

// Add .exe icon to Start Programs menu.
SdShowMsg("Adding icon to Start Programs menu...", TRUE);
svPath = TARGETDIR ^ PRODUCT_KEY;
LongPathToQuote(svPath, TRUE);
AddFolderIcon(FOLDER_PROGRAMS, PRODUCT_NAME, svPath,
             "", "", 0, "", REPLACE);
Delay(1);
SdShowMsg("Adding icon to Start Programs menu...", FALSE);

endprogram

/*-----*\
*
* Function:  HandleComponentError
*
* Purpose:  This function evaluates the value returned by a Component...
*           function and, if the value is less than zero, displays the error
*           number and aborts the setup.
*
\*-----*/
function HandleComponentError( nResult )

    NUMBER  nvError;
    STRING  svMedia, svComponent, svFileGroup, svFile;

begin
    if(nResult < 0) then
        ComponentError( svMedia, svComponent, svFileGroup, svFile, nvError );
        sprintfBox( INFORMATION, "Data Transfer Error Information",
                  "ComponentError returned the following data transfer error.\n" +

```

```
        "Setup will now abort.\n\n" +  
        "Media Name: %s\nComponent: %s\nFile Group: %s\n" +  
        "File: %s\nError Number: %ld",  
        svMedia, svComponent, svFileGroup, svFile, nvError );  
    abort;  
endif;  
end;  
  
#include "sddialog.rul"  
  
// Source file: Is5fn202.rul
```

{button ,PI("",`comment')} Feedback



DeinstallStart

Syntax

```
DeinstallStart (szDefLogPath, svLogFile, szKey, IStyle);
```

Description

The DeinstallStart function enables unInstallShield functionality. DeinstallStart creates the application uninstallation key and initializes the uninstall log file. Depending on the value of the second parameter, it either creates a log file or opens an existing log file to which it will append. It also creates the [UninstallString] value under the application uninstallation key.



You must call InstallationInfo before calling DeinstallStart.

You provide the uninstallation log file path in the first parameter. If the second parameter contains a null string (""), InstallShield provides the log file filename, resulting in the second half (following the -f switch) of the [UninstallString] value. If you provide a filename in the second parameter and that filename exists at the location specified in the first parameter and is a valid uninstallation log file, InstallShield will append uninstallation log file information to the specified log file.

When DeinstallStart is complete, the fully qualified name of the uninstallation log file is assigned to the variable in the second parameter. The first half of the [UninstallString] value, the path and filename of the unInstallShield executable file, is provided by InstallShield, which automatically transfers InstallShield's uninstallation executable file to the target system.

DeinstallStart is a special registry-related function, designed to work with certain predefined registry keys. Refer to [special registry-related functions](#) for more information.

DeinstallStart should be executed only once in a setup script. However, if you are launching multiple scripts using [DoInstall](#), then you can in effect call DeinstallStart more than once in a setup because each individual script can call DeinstallStart once.



If you call DeinstallStart in your setup script, you should do so immediately after calling InstallationInfo. If the setup aborts or is exited by the user after the call to InstallationInfo but before the call to DeinstallStart, the registry entries created by InstallationInfo will not be removed automatically since the uninstaller is not yet initialized.

Parameters

szDefLogPath

Specifies the fully qualified name of the folder in which the log file is to reside. Do not include the filename. InstallShield will create the filename and append it to the path you provide. If the path does not exist, it is created if possible. If you are appending to an existing log file, the path must match exactly the path used in the setup that created or most recently appended to the existing log file.

svLogFile

Specifies a filename for the log file. If svLogFile is a null string (""), DeinstallStart uses the filename Deisl1n.isu, where n is the lowest one-up number that makes the filename unique. If you provide a filename in the second parameter variable and that filename exists in the location specified in the first parameter and is a valid uninstallation log file, InstallShield will append uninstallation log file information to the specified log file. When DeinstallStart is complete, the fully qualified name of the uninstallation log file is assigned to the variable in the variable passed as this parameter.

szKey

Specifies the name of the application uninstallation key to be created.

IStyle

Pass zero in this parameter. No other value is allowed.

Return values

0

Indicates that the function successfully started the logging process.

< 0

Indicates that the function was unable to start the logging process.

Comments

The RegDBSetItem and RegDBGetItem functions set and retrieve the uninstallation icon name (the [DisplayName] value) under the application uninstallation key, which is created by calling the DeinstallStart function. Therefore, depending on the values to be set or retrieved, RegDBSetItem and RegDBGetItem require that DeinstallStart be called before they can be used.

DeinstallStart results in the creation of the application uninstallation key under Windows 95 and Windows NT 4.0 New Shell. DeinstallStart also provides the [UninstallString] value written under the application uninstallation key. Windows 3.1 using Program Manager does not use the application uninstallation key, however. Therefore, to provide uninstallation functionality you obtain the expression stored in the registry as the [UninstallString] value and use it as the command associated with an uninstallation icon in the program group. The uninstall executable filename comes from the UNINST system variable. The second parameter of DeinstallStart provides the log file filename used as the -f switch argument in the command line expression for the uninstallation icon. By concatenating the two values, you create the uninstallation icon command expression. The following example assigns a command expression to szCommand for use in an uninstallation icon:

```
szCommand = UNINST + " -f" + svLogFile;
```

Before calling any of the functions that use the SHAREDFILE or LOCKEDFILE option, and before calling CommitSharedFiles or SdFinishReboot, the application information key must be created using InstallationInfo, and the application uninstallation key must be created using the DeinstallStart function.

Enable(LOGGING) enables the logging of uninstallation information. It is enabled by default; you would need to call Enable(LOGGING) only to reverse the effect of a previous call to Disable(LOGGING).

{button ,JI('LANGREF.HLP>Examples','DeinstallStart_example')} [Example](#)

{button ,AL('DeinstallSetReference;CommitSharedFiles;Disable;Enable;InstallationInfo;RegDBGetItem;RegDBSetItem;SdFinishReboot;VerUpdateFile;Prepare my setup for uninstallation;Launch unInstallShield;How NT permissions affect unis;Uninstalling ini file entries;What gets logged for uninstallation;The uninstallation log file;unInstallShields executable files;OverView UninstallShield;Prevent unIS from removing running application;Launching unis under the PM shell',0,',')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



DeinstallStart example

```
/*-----*\
 *
 * This example illustrates the uses of the InstallationInfo,
 * DeinstallStart, and DeinstallSetReference functions.
 *
 * InstallationInfo is called to create an application information key in the
 * registry.
 *
 * DeinstallStart is called to create an uninstallation key in the registry.
 * RegDBSetItem is called to set the [DisplayName] value in the uninstallation
 * key. This value is displayed in the Windows95 and WinNT 4.0 'Add/Remove
 * Programs Properties' dialog.
 *
 * DeinstallSetReference is called to specify a file that, if locked at
 * uninstallation time, prevents uninstallation from continuing.
 *
 * To run properly this example requires at least a single
 * component that installs a file group containing Notepad.exe.
 * If you use an .exe file other than Notepad.exe, be sure to
 * change the PRODUCT_KEY define accordingly.
 *
 * To see the results of DeinstallSetReference, open the Notepad.exe file
 * installed by this example and try to uninstall the example.
 */-----*
#include "sddialog.h"

#define COMPANY_NAME          "ExampleCompany"
#define PRODUCT_NAME          "ExampleApp2"
#define PRODUCT_VERSION       "2.0"
#define PRODUCT_KEY           "Notepad.exe"
#define DEINST_KEY            "ExApp2Uninstall"

prototype HandleComponentError(NUMBER);

STRING  szCompany, szProduct, szVersion, szProductKey;
STRING  svDir, svLogFile, szKey, svPath;
NUMBER  nvDisk, nResult;

program

    // Disable Back button since it is not needed in this example.
    Disable(BACKBUTTON);

    // Initialize variables.
    szCompany    = COMPANY_NAME;
    szProduct    = PRODUCT_NAME;
    szVersion    = PRODUCT_VERSION;
    szProductKey = PRODUCT_KEY;

/*-----*\
 * Call InstallationInfo, which creates the application information key using
 * the values given to it. InstallationInfo also provides information to
 * create the per application paths key.
 */-----*
    InstallationInfo(szCompany, szProduct, szVersion, szProductKey);

    // The Welcome Dialog uses the product name value set by InstallationInfo.
    Welcome("", 0);
```

```

TARGETDIR = PROGRAMFILES ^ COMPANY_NAME;
AskDestPath("Destination Location",
            "Select a destination location.",
            TARGETDIR , 0 );
szKey = DEINST_KEY;

/*-----*\
*
* DeinstallStart initializes the uninstall log file using the values provided
* by InstallationInfo. DeinstallStart also creates the application
* uninstallation key using the values given to it.
*
* NOTE: Call DeinstallStart immediately after InstallationInfo so that there
*       are no intervening dialogs or events that could cause an exit or
*       abort. If an exit or abort occurs before DeinstallStart is called,
*       certain registry entries may not be removed because the uninstall
*       log file has not yet been initialized.
*
\*-----*/
DeinstallStart(TARGETDIR, svLogFile, szKey, 0);
RegDBSetItem(REGDB_UNINSTALL_NAME, PRODUCT_NAME);

// Specify a file that, if locked at uninstallation time will prevent
// uninstallation from continuing.
DeinstallSetReference(TARGETDIR ^ PRODUCT_KEY);

// Transfer files and handle any errors.
SdShowMsg("Installing program files...", TRUE);
nResult = ComponentMoveData ( MEDIA , nvDisk , 0 );
HandleComponentError(nResult);
Delay(1);
SdShowMsg("Installing program files...", FALSE);

// Add .exe icon to Start Programs menu.
SdShowMsg("Adding icon to Start Programs menu...", TRUE);
svPath = TARGETDIR ^ PRODUCT_KEY;
LongPathToQuote(svPath, TRUE);
AddFolderIcon(FOLDER_PROGRAMS, PRODUCT_NAME, svPath,
              "", "", 0, "", REPLACE);
Delay(1);
SdShowMsg("Adding icon to Start Programs menu...", FALSE);

endprogram

/*-----*\
*
* Function: HandleComponentError
*
* Purpose: This function evaluates the value returned by a Component...
*          function and, if the value is less than zero, displays the error
*          number and aborts the setup.
*
\*-----*/
function HandleComponentError( nResult )

    NUMBER   nvError;
    STRING   svMedia, svComponent, svFileGroup, svFile;

begin
    if(nResult < 0) then
        ComponentError( svMedia, svComponent, svFileGroup, svFile, nvError );
        SprintfBox( INFORMATION, "Data Transfer Error Information",
                    "ComponentError returned the following data transfer error.\n" +

```

```
        "Setup will now abort.\n\n" +  
        "Media Name: %s\nComponent: %s\nFile Group: %s\n" +  
        "File: %s\nError Number: %ld",  
        svMedia, svComponent, svFileGroup, svFile, nvError );  
    abort;  
endif;  
end;  
  
#include "sddialog.rul"  
  
// Source file: Is5fn202.rul
```

{button ,PI("",`comment')} Feedback



InstallationInfo

Syntax

InstallationInfo (szCompany, szProduct, szVersion, szProductKey);

Description

The InstallationInfo function specifies a company name, a product name, a product version number, and an application executable filename. The information you specify is used to create an application information key and a per application paths key for the program you are installing. The application information key is created immediately as a result of calling InstallationInfo. The per application paths key is not created until a subsequent call to RegDBSetItem sets a [Path] or [DefaultPath] value under that key.

InstallationInfo provides the product name for display in the Welcome dialog box. InstallationInfo provides the company name, product name, and version number that DeinstallStart uses to initialize the uninstallation log file. DeinstallStart will fail if InstallationInfo is not called before it in the script.

Call InstallationInfo only once in a setup. If you are launching multiple installations using DoInstall, each installation can of course have its own call to InstallationInfo.

InstallationInfo is a special registry-related function, designed to work with certain predefined registry keys. Refer to [special registry-related functions](#) for more information.

Parameters

szCompany

Specifies the company name. InstallShield uses szCompany to create a \<company> key under the [HKEY_LOCAL_MACHINE]\Software key in the registry.

szProduct

Specifies the name of the product to be installed. InstallShield uses szProduct to create a \<product> key under the [HKEY_LOCAL_MACHINE]\Software\<company> key in the registry. The value in szProduct is also inserted into the first paragraph of message text in the [Welcome](#) dialog box.

szVersion

Specifies the version number of the product. InstallShield uses szVersion to create a \<version> key under the [HKEY_LOCAL_MACHINE]\Software\<company>\<product> key in the registry. Together, the \<company> key (szCompany), the \<product> key (szProduct), and the \<version> key (szVersion) are referred to as the application information key. The application information key is created immediately upon calling InstallationInfo.

szProductKey

Specifies the name of the application's main executable file. If your product uses several executables, specify the executable that best represents the product. InstallShield uses szProductKey to create a per application paths key under the [HKEY_LOCAL_MACHINE]\Software\Microsoft\Windows\CurrentVersion\App Paths key. The per application paths key is not actually created in the registry until you call RegDBSetItem to create a value name and value pair under the key.

Return values

0

Indicates that the function was successful.

< 0

Indicates that the function was unable to use the parameters as set. Verify that you used proper syntax. Null string ("")s are not allowed in any parameter within this function.

Comments

InstallationInfo, the Registry, and Windows 95

The RegDBSetAppInfo and RegDBGetAppInfo functions set and retrieve values under the application information key created using the first three parameters of InstallationInfo. Therefore, the RegDBSetAppInfo and

RegDBGetAppInfo functions require that InstallationInfo be called before they can be used.

RegDBSetItem results in the creation of the per application paths key specified in the parameter szProductKey and sets values under that key. RegDBGetItem retrieves values under the per application paths key. These functions can also set and retrieve the uninstallation icon name value under the application uninstallation key, which is created by calling the DeinstallStart function. Therefore, depending on the values to be set or retrieved, RegDBSetItem and RegDBGetItem require that InstallationInfo, or InstallationInfo and DeinstallStart be called before they can be used. Remember that the per application paths key specified by the values passed to InstallationInfo is not actually created until RegDBSetItem is called to set a value under that key.

Before calling any of the functions that use the SHAREDFILE or LOCKEDFILE option and before calling CommitSharedFiles or SdFinishReboot, the application information key must be created using InstallationInfo, and uninstallation information must be set using the DeinstallStart function.

Enable(LOGGING) enables the logging of registry keys and values set by InstallationInfo for removal at uninstallation. It is enabled by default; you would need to call Enable(LOGGING) only to reverse the effect of a previous call to Disable(LOGGING).

InstallationInfo, Windows 3.1, and Windows NT with the Program Manager Shell

Windows 3.1 uses the Program Manager and InstallationInfo to accomplish a number of tasks that Windows 95 accomplishes using the registry (see information above). InstallationInfo provides the product name, version number, and company name for use in the [Application] section of Setup.log files. If InstallationInfo is not called, this section is not created in Setup.log files.

{button ,JI('LANGREF.HLP>Examples','InstallationInfo_example')}

[Example](#)

{button ,AL('CommitSharedFiles;Enable;DeinstallStart;Disable;RegDBGetAppInfo;RegDBGetItem;RegDBSetAppInfo;RegDBSetItem;SdFinishReboot;Prepare my setup for uninstallation',0,'')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



InstallationInfo example

```
/*-----*\
 *
 * This example illustrates the uses of the InstallationInfo,
 * DeinstallStart, and DeinstallSetReference functions.
 *
 * InstallationInfo is called to create an application information key in the
 * registry.
 *
 * DeinstallStart is called to create an uninstallation key in the registry.
 * RegDBSetItem is called to set the [DisplayName] value in the uninstallation
 * key. This value is displayed in the Windows95 and WinNT 4.0 'Add/Remove
 * Programs Properties' dialog.
 *
 * DeinstallSetReference is called to specify a file that, if locked at
 * uninstallation time, prevents uninstallation from continuing.
 *
 * To run properly this example requires at least a single
 * component that installs a file group containing Notepad.exe.
 * If you use an .exe file other than Notepad.exe, be sure to
 * change the PRODUCT_KEY define accordingly.
 *
 * To see the results of DeinstallSetReference, open the Notepad.exe file
 * installed by this example and try to uninstall the example.
 */-----*
#include "sddialog.h"

#define COMPANY_NAME      "ExampleCompany"
#define PRODUCT_NAME      "ExampleApp2"
#define PRODUCT_VERSION   "2.0"
#define PRODUCT_KEY       "Notepad.exe"
#define DEINST_KEY        "ExApp2Uninstall"

prototype HandleComponentError(NUMBER);

STRING  szCompany, szProduct, szVersion, szProductKey;
STRING  svDir, svLogFile, szKey, svPath;
NUMBER  nvDisk, nResult;

program

    // Disable Back button since it is not needed in this example.
    Disable(BACKBUTTON);

    // Initialize variables.
    szCompany      = COMPANY_NAME;
    szProduct      = PRODUCT_NAME;
    szVersion      = PRODUCT_VERSION;
    szProductKey   = PRODUCT_KEY;

/*-----*\
 * Call InstallationInfo, which creates the application information key using
 * the values given to it. InstallationInfo also provides information to
 * create the per application paths key.
 */-----*
    InstallationInfo(szCompany, szProduct, szVersion, szProductKey);

    // The Welcome Dialog uses the product name value set by InstallationInfo.
    Welcome("", 0);
```

```

TARGETDIR = PROGRAMFILES ^ COMPANY_NAME;
AskDestPath("Destination Location",
            "Select a destination location.",
            TARGETDIR , 0 );
szKey = DEINST_KEY;

/*-----*\
*
* DeinstallStart initializes the uninstall log file using the values provided
* by InstallationInfo. DeinstallStart also creates the application
* uninstallation key using the values given to it.
*
* NOTE: Call DeinstallStart immediately after InstallationInfo so that there
*       are no intervening dialogs or events that could cause an exit or
*       abort. If an exit or abort occurs before DeinstallStart is called,
*       certain registry entries may not be removed because the uninstall
*       log file has not yet been initialized.
*
\*-----*/
DeinstallStart(TARGETDIR, svLogFile, szKey, 0);
RegDBSetItem(REGDB_UNINSTALL_NAME, PRODUCT_NAME);

// Specify a file that, if locked at uninstallation time will prevent
// uninstallation from continuing.
DeinstallSetReference(TARGETDIR ^ PRODUCT_KEY);

// Transfer files and handle any errors.
SdShowMsg("Installing program files...", TRUE);
nResult = ComponentMoveData ( MEDIA , nvDisk , 0 );
HandleComponentError(nResult);
Delay(1);
SdShowMsg("Installing program files...", FALSE);

// Add .exe icon to Start Programs menu.
SdShowMsg("Adding icon to Start Programs menu...", TRUE);
svPath = TARGETDIR ^ PRODUCT_KEY;
LongPathToQuote(svPath, TRUE);
AddFolderIcon(FOLDER_PROGRAMS, PRODUCT_NAME, svPath,
             "", "", 0, "", REPLACE);
Delay(1);
SdShowMsg("Adding icon to Start Programs menu...", FALSE);

endprogram

/*-----*\
*
* Function:  HandleComponentError
*
* Purpose:  This function evaluates the value returned by a Component...
*           function and, if the value is less than zero, displays the error
*           number and aborts the setup.
*
\*-----*/
function HandleComponentError( nResult )

    NUMBER  nvError;
    STRING  svMedia, svComponent, svFileGroup, svFile;

begin
    if(nResult < 0) then
        ComponentError( svMedia, svComponent, svFileGroup, svFile, nvError );
        sprintfBox( INFORMATION, "Data Transfer Error Information",
                  "ComponentError returned the following data transfer error.\n" +

```

```
        "Setup will now abort.\n\n" +
        "Media Name: %s\nComponent: %s\nFile Group: %s\n" +
        "File: %s\nError Number: %ld",
        svMedia, svComponent, svFileGroup, svFile, nvError );
    abort;
endif;
end;

#include "sddialog.rul"

// Source file: Is5fn202.rul
```

{button ,PI("",`comment')} [Feedback](#)



RegDBCreateKeyEx

Syntax

```
RegDBCreateKeyEx (szKey, szClass);
```

Description

The RegDBCreateKeyEx function creates a key in the registry. You can also associate a class object with the newly created key (advanced users only). The newly created key does not have a value associated with it. Unless you specify otherwise, InstallShield creates the key as a subkey of HKEY_CLASSES_ROOT. You can use [RegDBSetDefaultRoot](#) to specify a different root key.

Separate different levels in a key-subkey expression with a double backslash (\\). InstallShield will create all levels immediately if they do not already exist.

Keys created with RegDBCreateKeyEx are recorded by unInstallShield for uninstallation when logging is enabled. However, remember that in multi-key expressions such as Key1\Key2\Key3, RegDBCreateKeyEx recognizes Key3 as the key of concern for that function call, just as the DOS command DIR C:\Windows\System lists the files in the System directory and not the Windows directory. Therefore, Key3 would be logged for uninstallation, but Key1 and Key2 would not.

To ensure correct uninstallation logging while creating a key and subkeys with RegDBCreateKeyEx, first create the parent key (with logging enabled). After the parent or top-level key in the key chain is created, then create the subkeys under the parent key. All subkeys under the individually created parent key will be uninstalled when the parent key is uninstalled.

For example, to ensure that Key1 and all its subkeys will be uninstalled, first create Key1 using RegDBCreateKeyEx while logging is enabled. Then you can create Key2, Key3, etc., in a single function call or in individual function calls. When Key1 is uninstalled, all subkeys under it will be uninstalled.

Remember that when a key is uninstalled, all its subkeys are also uninstalled. Therefore, if you use RegDBCreateKeyEx to create a key or keys under a key that is already logged for uninstallation, then the keys you create will be uninstalled when the higher-level key is uninstalled, regardless of whether logging is enabled when you create your keys and regardless of the order in which you create your keys.



If the key you are creating with RegDBCreateKeyEx already exists and you have not disabled logging, the key—which other applications use—will be logged for uninstallation. When unInstallShield uninstalls your application, the key will be uninstalled, causing problems for those applications that use the key. To avoid this problem, test for the existence of the key using [RegDBKeyExist](#) before creating it. If the key already exists, use RegDBCreateKeyEx to create a subkey unique to your application. Then, when uninstallation takes place, only the subkey will be deleted. If you do not wish to test for the existence of the key first, you can use the [Disable](#) function to disable logging while you create the key. Enable logging after the key is created. Remember, however, that the key will not be uninstalled with your application.

RegDBCreateKeyEx is a general registry-related function, designed to work with all registry keys, including those handled by the special registry-related functions. Refer to [special registry-related functions](#) for more information.

Parameters

szKey

Specifies the name of the key to create under one of the four root keys. Separate different levels in the subkey with a double backslash (\\).

szClass

Specifies the class name to associate with this key.

Return values

0

Indicates that the function successfully created the subkey.

< 0

Indicates that the function was unable to create the subkey.

Comment

Windows NT 4.0 does not allow the creation of a key directly under HKEY_LOCAL_MACHINE.

```
{button ,JI('LANGREF.HLP>Examples','RegDBCreateKeyEx_example')}
```

[Example](#)

```
{button ,AL('Disable;Enable;RegDBSetDefaultRoot;RegDBDeleteKey;RegDBGetKeyValueEx;RegDBKeyExist;RegDBSetKeyValueEx;Prepare my setup for uninstallation;Overview the registry',0,',')}
```

[See also](#)

```
{button ,PI('','comment')} Feedback
```



RegDBCreateKeyEx example

```
/*-----*\
*
* This example illustrates the RegDBCreateKeyEx and RegDBKeyExist functions.
*
* RegDBCreateKey is first called to create a subkey with no class value in
* the HKEY_CLASSES_ROOT key. RegDBKeyExist is then called to check if the
* key was created.
*
* RegDBCreateKey is called again to create a multi-level subkey with a class
* value associated with it under HKEY_CLASSES_ROOT. RegDBKeyExist is
* again called to check the existence of the new key.
*
/*-----*/

    STRING szKey, szClass, szKeyRoot, szTitle, szMsg, svLogFile;
    NUMBER nResult1, nResult2;

program

/*-----*\
*
* Create a key with no class value.
*
/*-----*/

    szKey = "CreateKeyExample";
    szClass = "";

    szTitle = "RegDBCreateKeyEx & RegDBKeyExist";

    if (RegDBCreateKeyEx(szKey, szClass) < 0) then
        MessageBox("First call to RegDBCreateKeyEx failed.", SEVERE);
        abort;
    else
        sprintfBox(INFORMATION, szTitle, "Successfully created: %s", szKey);

/*-----*\
*
* Check to see if the key just created exists.
*
/*-----*/

    if (RegDBKeyExist(szKey) < 0) then
        MessageBox("First call to RegDBKeyExist failed.", SEVERE);
    else
        sprintfBox(INFORMATION, szTitle, "%s exists.", szKey);
    endif;
endif;

    if (RegDBDeleteKey(szKey) < 0) then

        MessageBox("RegDBDeleteKey failed.", SEVERE);
    endif;

/*-----*\
*
* Create a key with more than one sublevel and a class value.
*
/*-----*/
```

```

szKey      = "ShareWare\\Games\\CoolChess";
szClass    = "LastPlayed";
szKeyRoot  = "ShareWare";

if (RegDBCreateKeyEx(szKey, szClass) < 0) then
    MessageBox("Second call to RegDBCreateKeyEx failed.", SEVERE);
    abort;
else
    sprintfBox(INFORMATION, szTitle, "Successfully created: %s", szKey);

/*-----*\
*
* Check if the newly created multi-level key exists.
*
*\-----*/

    if (RegDBKeyExist(szKeyRoot) < 0) then
        MessageBox("Second call to RegDBKeyExist failed.", SEVERE);
    else
        sprintfBox(INFORMATION, szTitle, "%s exists.", szKey);
    endif;
endif;

if (RegDBDeleteKey(szKey) < 0) then
    MessageBox("RegDBDeleteKey failed.", SEVERE);
endif;

endprogram

// Source file: Is5fn237.rul

```

{button ,PI("",`comment')} [Feedback](#)



RegDBDeleteKey

Syntax

RegDBDeleteKey (szSubKey);

Description

The RegDBDeleteKey function deletes a specific key and its associated value from the registry. All subkeys of the deleted key are also deleted, along with their associated values.

InstallShield assumes the key specified in szSubKey is a subkey of HKEY_CLASSES_ROOT. You can use [RegDBSetDefaultRoot](#) to specify another root key.

RegDBDeleteKey is a general registry-related function, designed to work with all registry keys, including those handled by the special registry-related functions. Refer to [special registry-related functions](#) for more information.

Parameters

szSubKey

Specifies the name of the key to delete. Separate different levels in the subkey with a double backslash (\\).

Return values

0

Indicates that the function successfully deleted the key.

< 0

Indicates that the function was unable to delete the key.

{button ,JI('LANGREF.HLP>Examples','RegDBDeleteKey_example')} [Example](#)

{button ,AL('RegDBCreateKeyEx;RegDBDeleteValue;RegDBKeyExist;RegDBSetDefaultRoot;Overview the registry',0,'')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



RegDBDeleteKey example



Almost all of the examples provided with the registry functions use this function to delete a key. Please refer to those examples for more information.

```
/*-----*\
*
* This example illustrates the RegDBDeleteKey function.
*
*-----*/

STRING szKey, szClass, szKeyRoot, szTitle, szMsg, svLogFile;
NUMBER nResult1, nResult2;

program

    szTitle = "RegDBDeleteKey Example";

    // Create a key with no class value.
    szKey   = "DeleteMeKey";
    szClass = "";

    if (RegDBCreateKeyEx(szKey, szClass) < 0) then

        MessageBox("RegDBCreateKeyEx failed.", SEVERE);
        abort;
    else

        sprintfBox(INFORMATION, szTitle, "%s successfully created.", szKey);
    endif;

/*-----*\
*
* Call RegDBDeleteKey to delete the key just created.
*
*-----*/

    if (RegDBDeleteKey(szKey) < 0) then
        MessageBox("RegDBDeleteKey failed.", SEVERE);
    else
        sprintfBox(INFORMATION, szTitle, "%s successfully deleted.", szKey);
    endif;

endprogram

// Source file: Is5fn238.rul
```

{button ,PI("",`comment')} [Feedback](#)



RegDBDeleteValue

Syntax

RegDBDeleteValue (szSubKey, szValue);

Description

The RegDBDeleteValue function deletes a value from a specific key in the registry. InstallShield assumes that the key specified in szSubKey is a subkey of HKEY_CLASSES_ROOT. You must use [RegDBSetDefaultRoot](#) to specify another root key.

RegDBDeleteKey is a general registry-related function, designed to work with all registry keys, including those handled by the special registry-related functions. Refer to [special registry-related functions](#) for more information.

Parameters

szSubKey

Specifies the name of the registry key that contains the value name to delete. Separate different levels in the subkey with a double backslash (\\).

szValue

Enter the name of the value you want to delete.

Return values

0

Indicates that the function successfully deleted the value.

< 0

Indicates that the function was unable to delete the value.

{button ,JI('LANGREF.HLP>Examples','RegDBDeleteValue_example')} [Example](#)

{button ,AL('RegDBDeleteKey;RegDBGetKeyValueEx;RegDBSetDefaultRoot;Overview the registry',0,','')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



RegDBDeleteValue example

```
/*-----*\
*
* This example illustrates the RegDBDeleteValue function.
*
* RegDBDeleteValue is called to delete the value name "Cursive" from the
* registry key:
* "HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Fonts".
*
* NOTE: In order for this script to run properly, it is necessary for the
*       preprocessor constants to be set to a valid subkey and value on the
*       target system.
*
*\-----*/

#define SUBKEY "\\SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Fonts"
#define VALUE  "Cursive"

    STRING szSubKey, szValue, szTitle;
    NUMBER nReturn;

program

    szTitle = "RegDBDeleteValue Example";

    // Set the root key.
    RegDBSetDefaultRoot(HKEY_LOCAL_MACHINE);

    // Set the name of the subkey.
    szSubKey = SUBKEY;
    szValue  = VALUE;

    nReturn = RegDBDeleteValue(szSubKey, szValue);

    if (nReturn < 0) then
        MessageBox("RegDBDeleteValue failed.", SEVERE);
    else
        sprintfBox(INFORMATION, szTitle, "%s successfully deleted.", szValue);
    endif;

endprogram

// Source file: Is5fn239.rul
```

{button ,PI("",`comment')} [Feedback](#)



RegDBGetAppInfo

Syntax

```
RegDBGetAppInfo (szName, nvType, svValue, nvSize);
```

Description

The RegDBGetAppInfo function retrieves from the registry the value of a particular value name under the application information key of your main application. The application information key is created by InstallShield as a result of calling [InstallationInfo](#). You must call InstallationInfo to create an application information key before calling RegDBGetAppInfo.

RegDBGetAppInfo is a special registry-related function, designed to work with certain predefined registry keys. Refer to [special registry-related functions](#) for more information.

Parameters

szName

Specifies the value name of the value to retrieve.

nvType

Returns the type of data, identified by one of these constants:

REGDB_STRING

String variable, no newline characters allowed.

REGDB_STRING_EXPAND

String variable holding an expandable environment variable expression, such as "%MYPATH%".

REGDB_STRING_MULTI

String variable, newline characters allowed.

REGDB_NUMBER

Number expressed as a string and passed in string variable.

REGDB_BINARY

Binary data stored in a string.

svValue

Returns the value of the value name specified in szName.

nvSize

Returns the size, in bytes, of the return value.

Return values

x

Where x is the actual number of bytes returned in svValue.

< 0

Indicates that the function was unable to retrieve the value.

{button ,JI('LANGREF.HLP>Examples',`RegDBGetAppInfo_example`)} [Example](#)

{button ,AL(`InstallationInfo;RegDBSetAppInfo;Overview the registry',0,`,`)} [See also](#)

{button ,PI(`",`comment`)} [Feedback](#)



RegDBGetAppInfo example

```
/*-----*\
 *
 * This example illustrates the RegDBSetAppInfo and RegDBGetAppInfo functions.
 *
 * Before calling either of these functions, the application information must
 * be set using the InstallationInfo function.
 *
 *-----*/

#define COMPANY_NAME           "Example_Company"
#define PRODUCT_NAME          "Example_App"
#define PRODUCT_VERSION       "5.0"
#define PRODUCT_KEY           "EXAMPLE.EXE"
#define DEINSTALL_KEY         "Example_DeinstKey"
#define UNINSTALL_NAME        "Example_App_5.0"
#define DEFAULT_LOG_PATH      "EXAMPLE"

    STRING  szStrName, szStrValue, svStrValue, szTitle, szMsg, svLogFile;
    NUMBER  nvSize, nvType;

program

    // Set the root key.
    RegDBSetDefaultRoot(HKEY_LOCAL_MACHINE);

    // Set the name to be used with REGDB_STRING.
    szStrName = "ExampleStringValue";
    szStrValue = "ExampleStringSetting";

    szTitle   = "RegDBGetAppInfo";

    // Set up the application information prior to using RegDBSetAppInfo
    // and RegDBGetAppInfo.
    InstallationInfo(COMPANY_NAME, PRODUCT_NAME, PRODUCT_VERSION, PRODUCT_KEY);
    DeinstallStart(DEFAULT_LOG_PATH, svLogFile, DEINSTALL_KEY, 0);

    // Set value of type REGDB_STRING
    if (RegDBSetAppInfo(szStrName, REGDB_STRING, szStrValue, -1) < 0) then
        MessageBox("Failed to set key and value of REGDB_STRING type.", SEVERE);
        abort;
    endif;

    // RegDBGetAppInfo is called to return the values and compare all the setup
    // parameters.
    if (RegDBGetAppInfo(szStrName, nvType, svStrValue, nvSize) < 0) then
        MessageBox("Failed to get application information value.", SEVERE);
        abort;
    else

        // Check to see if the value retrieved is the same as the value set.
        if (nvType != REGDB_STRING) then
            MessageBox("Type comparison failed.", WARNING);
        endif;

        if (szStrValue != svStrValue) then
            MessageBox ("Sub key value comparison Failed.", WARNING);
        else
            szMsg = "Set values: %s = %s\n\nReturn values: %s = %s";

```

```
        SprintfBox(INFORMATION, szTitle, szMsg, szStrName, szStrValue,
                  szStrName, svStrValue);
endif;

if (nvSize != StrLength(szStrValue) + 1) then
    MessageBox ("Size Comparison failed.", WARNING);
else

    szMsg = "Size in bytes entered: %d\n\nSize in bytes returned: %d";
    SprintfBox(INFORMATION, szTitle, szMsg,
              nvSize, StrLength(szStrValue) + 1);
endif;
endif;

endprogram

// Source file: Is5fn240.rul
```

{button ,PI("",`comment')} [Feedback](#)



RegDBGetItem

Syntax

```
RegDBGetItem (nItem, svValue);
```

Description

The RegDBGetItem function retrieves values under the per-application paths key or the application uninstallation key, depending on the value of nItem.

RegDBGetItem is a special registry-related function, designed to work with certain predefined registry keys. Refer to [special registry-related functions](#) for more information.

Parameters

nItem

Specifies which item to retrieve. Pass one of the following predefined constants in this parameter:

REGDB_APPPATH

The value of [Path] under the per application paths key.

REGDB_APPPATH_DEFAULT

The value of [DefaultPath] under the per application paths key.

REGDB_UNINSTALL_NAME

The value of [DisplayName] under the uninstallation key. When this constant is used, svValue specifies the application name shown in the list of uninstallable applications in Control Panel.

svValue

Returns the value of the item.

Return values

0

Indicates that the function successfully retrieved the value of the item.

< 0

Indicates that the function was unable to retrieve the value of the item. Verify that you used the InstallationInfo function before using this function.

Comments

- Before calling the RegDBGetItem function using the REGDB_APPPATH or REGDB_APPPATH_DEFAULT options for nItem, you must create the per application paths key using the InstallationInfo function.
- Before calling the RegDBGetItem function using the REGDB_UNINSTALL_NAME option for nItem, you must use the DeinstallStart function to create the application uninstallation key, and to assign a value to [DisplayName] under the application uninstallation key.

{button ,JI('LANGREF.HLP>Examples','RegDBGetItem_example')} [Example](#)

{button ,AL('DeinstallStart;InstallationInfo;RegDBSetItem;Overview the registry',0,'')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



RegDBGetItem example

```
/*-----*\
 *
 * The following example illustrates the RegDBSetItem and RegDBGetItem
 * functions.
 *
 *-----*/

#define COMPANY_NAME      "ExampleCompany"
#define PRODUCT_NAME     "ExampleProduct"
#define VERSION_NUMBER   "5.00.00"
#define PRODUCT_KEY      "EXAMPLE.EXE"
#define DEINST_KEY       "ExampleDeinstKey"
#define APP_DEF_LOG_PATH "C:\\EXAMPLE\\TEMP"

#define APP_PATH          "C:\\EXAMPLE"
#define APP_DEF_PATH     "C:\\EXAMPLE\\TARGET"
#define UNINSTALL_NAME   "ExampleUninstallName"

    STRING  szAppPath, szAppDefPath, szUninstallName, svLogFile;
    STRING  svValue, szTitle;

program

    // Set the root key.
    RegDBSetDefaultRoot(HKEY_LOCAL_MACHINE);

    // Set installation and uninstallation information in
    // order to call RegDBSetItem and RegDBGetItem.

    InstallationInfo(COMPANY_NAME, PRODUCT_NAME, VERSION_NUMBER, PRODUCT_KEY);
    DeinstallStart(APP_DEF_LOG_PATH, svLogFile, DEINST_KEY, 0);

    szTitle          = "RegDBSetItem";
    szAppPath        = APP_PATH;
    szAppDefPath     = APP_DEF_PATH;
    szUninstallName = UNINSTALL_NAME;

/*-----*\
 *
 * Call RegDBSetItem to set the value of the application path key in the
 * registry to the value of szAppPath.
 *
 *-----*/

    if (RegDBSetItem(REGDB_APPPATH, szAppPath) < 0) then
        MessageBox("Unable to set application path key.", SEVERE);
    else
        sprintfBox(INFORMATION, szTitle, "RegDBSetItem set the application " +
            "path key to %s.", szAppPath);
    endif;

/*-----*\
 *
 * Call RegDBSetItem to set the value of the application default path key in
 * the registry to the value of szAppDefPath.
 *
 *-----*/

    if (RegDBSetItem(REGDB_APPPATH_DEFAULT, szAppDefPath) < 0) then
```

```

    MessageBox("Unable to set applicatoin default path key.", SEVERE);
else
    sprintfBox(INFORMATION, szTitle, "RegDBSetItem set the application " +
              "default path key to %s.", szAppDefPath);
endif;

/*-----*\
*
* Call RegDBSetItem to set the value of the uninstall name key in the
* registry to the value of szUninstallName.
*
\*-----*/

if (RegDBSetItem(REGDB_UNINSTALL_NAME, szUninstallName) < 0) then
    MessageBox("Unable to set uninstall name key.", SEVERE);
else
    sprintfBox(INFORMATION, szTitle, "RegDBSetItem set the uninstall " +
              "name key to %s.", szUninstallName);
endif;

szTitle = "RegDBGetItem";

/*-----*\
*
* Call RegDBGetItem to get the value of the application path key in the
* registry.
*
\*-----*/

if (RegDBGetItem(REGDB_APPPATH, svValue) < 0) then
    MessageBox("Unable to get value of application path key.", SEVERE);
else
    sprintfBox(INFORMATION, szTitle, "RegDBGetItem retrieved the value " +
              "of the application path key: %s.", svValue);
endif;

/*-----*\
*
* Call RegDBGetItem to get the value of the application default path key in
* the registry.
*
\*-----*/

if (RegDBGetItem(REGDB_APPPATH_DEFAULT, svValue) < 0) then
    MessageBox("Unable to get application default path key", SEVERE);
else
    sprintfBox(INFORMATION, szTitle, "RegDBGetItem retrieved the value " +
              "of the application default path key: %s.", svValue);
endif;

/*-----*\
*
* Call RegDBGetItem to get the value of the uninstall name key in the
* registry.
*
\*-----*/

if (RegDBGetItem(REGDB_UNINSTALL_NAME, svValue) < 0) then
    MessageBox("Unable to get application uninstall name key.", SEVERE);
else
    sprintfBox(INFORMATION, szTitle, "RegDBGetItem retrieved the value " +
              "of the uninstalation name key: %s.", svValue);
endif;

```

endprogram

// Source file: Is5fn241.rul

{button ,PI("",`comment')} Feedback



RegDBGetKeyValueEx

Syntax

```
RegDBGetKeyValueEx (szKey, szName, nvType, svValue, nvSize);
```

Description

The RegDBGetKeyValueEx function retrieves the value of a particular value name under a specified key in the registry. By default, InstallShield assumes this key is a subkey of HKEY_CLASSES_ROOT. You can use [RegDBSetDefaultRoot](#) to specify another root key.

RegDBGetKeyValueEx is a general registry-related function, designed to work with all registry keys, including those handled by the special registry-related functions. Refer to [special registry-related functions](#) for more information.

Parameters

szKey

Specifies the name of the key whose value is to be retrieved. Separate different levels in the subkey with a double backslash (\\).

szName

Specifies the value name under szKey of the value to retrieve. To retrieve the default value of the key, pass a null string ("").

nvType

Returns the type of data, identified by one of the following constants. You do not pass these constants in the parameter nvType. Rather, you test the value returned in the parameter nvType from the call to RegDBGetKeyValueEx using the constants.

REGDB_STRING

String variable, no newline characters allowed

REGDB_STRING_EXPAND

String variable holding an expandable environment variable expression such as "%MYPATH%".

REGDB_STRING_MULTI

String variable, newline characters allowed.

REGDB_NUMBER

Number expressed as a string and passed in string variable.

REGDB_BINARY

Binary data stored in a string.

svValue

Points to the string containing the data. Numbers are converted in the string.

nvSize

Returns size, in bytes, of the data you are retrieving.

Return values

0

Indicates that the function successfully retrieved the value.

< 0

Indicates that the function was unable to retrieve the value.

Comments

On a Windows NT platform, when data type REGDB_STRING_MULTI is retrieved, use StrGetTokens with null

string ("") to parse the multiple null terminated strings into a list of strings. That is, if svValue has the resulting multiple strings after a call to RegDBGetKeyValueEx, StrGetTokens(listID, svValue, "") can be used to parse the strings and put them in a string list pointed by listID.

{button ,JI('LANGREF.HLP>Examples','RegDBGetKeyValueEx_example')}

Example

{button ,AL('RegDBCreateKeyEx;RegDBDeleteKey;RegDBKeyExist;RegDBSetKeyValueEx;Overview the registry',0,'')} See also

{button ,PI('','comment')} Feedback



RegDBGetKeyValueEx example

```
/*-----*\
 *
 * This example illustrates the RegDBSetKeyValueEx and RegDBGetKeyValueEx functions.
 *
 * RegDBCreateKeyEx is called to create a test subkey in the HKEY_CLASSES_ROOT
 * key. The value of a key is set in integer form using the REGDB_NUMBER
 * option of the RegDBSetKeyValueEx function. After this value is set, it is
 * retrieved using the RegDBGetKeyValueEx function and verified.
 *
 *-----*/

    STRING szKey, szNumName, szNumValue, svNumValue, szTitle, szMsg;
    NUMBER nType, nSize, nvType, nvSize;

program

    // Create a key to test.
    szKey      = "TestKey";
    if (RegDBCreateKeyEx(szKey, "") < 0) then
        MessageBox("RegDBCreateKeyEx failed.", SEVERE);
        abort;
    endif;

    szNumName  = "TestValue";
    szNumValue = "12345";
    nType      = REGDB_NUMBER;
    nSize      = -1;
    szTitle    = "RegDBSetKeyValueEx & RegDBGetKeyValueEx";

/*-----*\
 *
 * RegDBSetKeyValueEx is called to set a key name and a value associated with
 * it.
 *
 *-----*/

    if (RegDBSetKeyValueEx(szKey, szNumName, nType, szNumValue,
        nSize) < 0) then
        MessageBox("RegDBSetKeyValueEx failed.", SEVERE);
        abort;
    else
        // Display what RegDBSetKeyValueEx had done.
        szMsg = "%s set to: %s";
        sprintfBox(INFORMATION, szTitle, szMsg, szNumName, szNumValue);
    endif;

/*-----*\
 *
 * RegDBGetKeyValueEx is called to retrieve key value information.
 *
 *-----*/

    if (RegDBGetKeyValueEx(szKey, szNumName, nvType, svNumValue,
        nvSize) < 0) then
        MessageBox("RegDBGetKeyValueEx failed.", SEVERE);
    else

        // Check to see if the value returned is the same as the value set.
        if (nvType != REGDB_NUMBER) then
```

```
        MessageBox("Type comparison failed.", SEVERE);
    endif;

    if (svNumValue != szNumValue) then
        MessageBox("Subkey value comparison failed.", SEVERE);
    endif;

    // Display what RegDBGetKeyValueEx retrieved.
    szMsg = "%s has value: %s\n\nThis data is %d bytes.";
    sprintfBox(INFORMATION, szTitle, szMsg, szNumName, svNumValue, nvSize);
endif;

// Delete the created test key.
if (RegDBDeleteKey(szKey) < 0) then
    MessageBox("RegDBDeleteKey failed.", SEVERE);
endif;

endprogram

// Source file: Is5fn242.rul
```

{button ,PI("",`comment')} Feedback



RegDBKeyExist

Syntax

RegDBKeyExist (szSubKey);

Description

The RegDBKeyExist function checks for the existence of a specific key in the registry. By default, InstallShield assumes this key is a subkey of HKEY_CLASSES_ROOT. If you want to use a different main key, use [RegDBSetDefaultRoot](#) to specify another root key.

RegDBKeyExist is a general registry-related function, designed to work with all registry keys, including those handled by the special registry-related functions. Refer to [special registry-related functions](#) for more information.

Parameters

szSubKey

Specifies the name of the key to find. You do not have to include the HKEY_CLASSES_ROOT key (or another root key you specified) in this parameter. Separate different levels in the subkey with a double backslash (\\).

Return values

1

Indicates that the function found the key name in the registry.

< 0

Indicates that the function was unable to find the key name in the registry.

{button ,JI('LANGREF.HLP>Examples','RegDBKeyExist_example')} [Example](#)

{button ,AL('RegDBDeleteKey;RegDBSetKeyValueEx;Overview the registry',0,'')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



RegDBKeyExist example



Almost all of the examples provided with registry functions make use of this function. Please refer to those examples for more information.

```
/*-----*\
*
* This example illustrates the RegDBCreateKeyEx and RegDBKeyExist functions.
*
* RegDBCreateKey is first called to create a subkey with no class value in
* the HKEY_CLASSES_ROOT key. RegDBKeyExist is then called to check if the
* key was created.
*
* RegDBCreateKey is called again to create a multi-level subkey with a class
* value associated with it under HKEY_CLASSES_ROOT. RegDBKeyExist is
* again called to check the existence of the new key.
*
/*-----*/

    STRING szKey, szClass, szKeyRoot, szTitle, szMsg, svLogFile;
    NUMBER nResult1, nResult2;

program

/*-----*\
*
* Create a key with no class value.
*
/*-----*/

    szKey   = "CreateKeyExample";
    szClass = "";

    szTitle = "RegDBCreateKeyEx & RegDBKeyExist";

    if (RegDBCreateKeyEx(szKey, szClass) < 0) then
        MessageBox("First call to RegDBCreateKeyEx failed.", SEVERE);
        abort;
    else
        sprintfBox(INFORMATION, szTitle, "Successfully created: %s", szKey);
/*-----*\
*
* Check to see if the key just created exists.
*
/*-----*/

    if (RegDBKeyExist(szKey) < 0) then
        MessageBox("First call to RegDBKeyExist failed.", SEVERE);
    else
        sprintfBox(INFORMATION, szTitle, "%s exists.", szKey);
    endif;
endif;

    if (RegDBDeleteKey(szKey) < 0) then

        MessageBox("RegDBDeleteKey failed.", SEVERE);
    endif;

/*-----*\
```

```

*
* Create a key with more than one sublevel and a class value.
*
/*-----*/

szKey      = "ShareWare\\Games\\CoolChess";
szClass    = "LastPlayed";
szKeyRoot  = "ShareWare";

if (RegDBCreateKeyEx(szKey, szClass) < 0) then
    MessageBox("Second call to RegDBCreateKeyEx failed.", SEVERE);
    abort;
else
    sprintfBox(INFORMATION, szTitle, "Successfully created: %s", szKey);
/*-----*\
*
* Check if the newly created multi-level key exists.
*
/*-----*/

    if (RegDBKeyExist(szKeyRoot) < 0) then
        MessageBox("Second call to RegDBKeyExist failed.", SEVERE);
    else
        sprintfBox(INFORMATION, szTitle, "%s exists.", szKey);
    endif;
endif;

if (RegDBDeleteKey(szKey) < 0) then
    MessageBox("RegDBDeleteKey failed.", SEVERE);
endif;

endprogram

// Source file: Is5fn237.rul

```

{button ,PI("",`comment')} Feedback



RegDBQueryKey

Syntax

RegDBQueryKey (szSubKey, nItem, listResults);

Description

The RegDBQueryKey function allows users to query a key for its subkeys and value names. The keys can be enumerated dynamically at run time using this function.

RegDBQueryKey is a general registry-related function, designed to work with all registry keys, including those handled by the special registry-related functions. Refer to [special registry-related functions](#) for more information.

Parameters

szSubKey

Specifies subkey(s) under one of the root keys that were set by a previous call to [RegDBSetDefaultRoot](#). Use backslashes to specify deeper levels of subkeys. To retrieve the root key, pass a null string ("").

nItem

Specifies which items should be placed in the list. Pass one of the following predefined constants in this parameter:

REGDB_KEYS

The string list returned in listResults will contain a list of all the subkeys under this key.

REGDB_NAMES

The string list returned in listResults will contain the names of all named values for this key. This constant is not valid under Windows 3.1.

listResults

Returns the results of the query in a string list.

Return values

0

Indicates function was successful.

< 0

Indicates function failed.

Comments

All keys in Windows 3.1, Windows NT and Windows 95 can have subkeys. Windows 95 and Windows NT can also allow a single key to have a set of "named" values. In contrast, Windows 3.1 allows only one default value per key.

{button ,JI('LANGREF.HLP>Examples',`RegDBQueryKey_example`)} [Example](#)

{button ,AL(`RegDBKeyExist;RegDBSetDefaultRoot;Overview the registry',0,`,`)} [See also](#)

{button ,PI("",`comment`)} [Feedback](#)



RegDBQueryKey example

```
/*-----*\
 *
 * This example illustrates the RegDBQueryKey function.
 *
 * RegDBQueryKey is first called to query the subkeys under the SUBKEY1 key.
 * The returned list is displayed in a dialog box.
 *
 * RegDBQueryKey is called a second time to query the subkeys under the
 * SUBKEY2 key. This list is also displayed in a dialog box.
 *
 \*-----*/

#define KEY "SOFTWARE"
#define KEY2 "SOFTWARE\\Microsoft"

#include "Sddialog.h"

STRING szSubKey, szMsg, szTitle;
NUMBER nReturn, nItem;
LIST szSubKeyList, szNameList;

program

szTitle = "RegDBQueryKey Example";

szSubKeyList = ListCreate(STRINGLIST);
szNameList = ListCreate(STRINGLIST);

if ((szNameList = LIST_NULL) || (szSubKeyList = LIST_NULL)) then
    MessageBox ("Unable to create necessary lists!", SEVERE);
    abort;
endif;

RegDBSetDefaultRoot(HKEY_LOCAL_MACHINE);
szSubKey = KEY;
nItem = REGDB_KEYS;

nReturn = RegDBQueryKey(szSubKey, nItem, szSubKeyList);

if (nReturn < 0) then
    MessageBox("First call to RegDBQueryKey failed.", SEVERE);
else
    szMsg = "Subkeys under " + KEY + " key:";
    SdShowInfoList(szTitle, szMsg, szSubKeyList);
endif;

szSubKey = KEY2;
nItem = REGDB_NAMES;
nReturn = RegDBQueryKey(szSubKey, nItem, szNameList);

if (nReturn < 0) then
    MessageBox("Second call to RegDBQueryKey failed.", SEVERE);
else
    szMsg = "Named values under " + KEY2 + " key";
    SdShowInfoList(szTitle, szMsg, szNameList);
endif;

ListDestroy (szNameList);
ListDestroy (szSubKeyList);
```

```
endprogram

#define SD_SINGLE_DIALOGS 1
#define SD_SHOWINFOLIST 1

#include "Sddialog.rul"

// Source file: Is5fn244.rul
```

{button ,PI("",`comment')} [Feedback](#)



RegDBSetAppInfo

Syntax

RegDBSetAppInfo (szName, nType, szValue, nSize);

Description

The RegDBSetAppInfo function sets the value of a particular value name under the application information key in the registry. You must call InstallationInfo to create an application information key before calling RegDBSetAppInfo.

RegDBSetAppInfo is a special registry-related function, designed to work with certain predefined registry keys. Refer to [special registry-related functions](#) for more information.

Parameters

szName

Specifies the value name whose information is to be set.

nType

Specifies the type of data you are setting. Pass one of the following predefined constants in this parameter:

REGDB_STRING

String variable, no newline characters allowed.

REGDB_STRING_EXPAND

String variable holding an expandable environment variable expression, such as "%MYPATH%".

REGDB_STRING_MULTI

String variable, newline characters allowed.

REGDB_NUMBER

Number expressed as a string and passed in string variable.

REGDB_BINARY

Binary data stored in a string.

szValue

Specifies the value to set for the value name.

nSize

Specifies the size, in bytes, of the data passed to RegDBSetAppInfo. Pass -1 in this parameter to indicate that InstallShield should determine the size of the data.

Return values

0

Indicates that the function successfully assigned the value to the value name.

< 0

Indicates that the function was unable to assign the value.

{button ,JI('LANGREF.HLP>Examples','RegDBSetAppInfo_example')} [Example](#)

{button ,AL('InstallationInfo;RegDBGetAppInfo;RegDBSetKeyValueEx;Overview the registry',0,','')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



RegDBSetAppInfo example

```
/*-----*\
 *
 * This example illustrates the RegDBSetAppInfo and RegDBGetAppInfo functions.
 *
 * Before calling either of these functions, the application information must
 * be set using the InstallationInfo function.
 *
 *-----*/

#define COMPANY_NAME           "Example_Company"
#define PRODUCT_NAME          "Example_App"
#define PRODUCT_VERSION       "5.0"
#define PRODUCT_KEY           "EXAMPLE.EXE"
#define DEINSTALL_KEY        "Example_DeinstKey"
#define UNINSTALL_NAME        "Example_App_5.0"
#define DEFAULT_LOG_PATH      "EXAMPLE"

    STRING  szStrName, szStrValue, svStrValue, szTitle, szMsg, svLogFile;
    NUMBER  nvSize, nvType;

program

    // Set the root key.
    RegDBSetDefaultRoot(HKEY_LOCAL_MACHINE);

    // Set the name to be used with REGDB_STRING.
    szStrName = "ExampleStringValue";
    szStrValue = "ExampleStringSetting";

    szTitle   = "RegDBGetAppInfo";

    // Set up the application information prior to using RegDBSetAppInfo
    // and RegDBGetAppInfo.
    InstallationInfo(COMPANY_NAME, PRODUCT_NAME, PRODUCT_VERSION, PRODUCT_KEY);
    DeinstallStart(DEFAULT_LOG_PATH, svLogFile, DEINSTALL_KEY, 0);

    // Set value of type REGDB_STRING
    if (RegDBSetAppInfo(szStrName, REGDB_STRING, szStrValue, -1) < 0) then
        MessageBox("Failed to set key and value of REGDB_STRING type.", SEVERE);
        abort;
    endif;

    // RegDBGetAppInfo is called to return the values and compare all the setup
    // parameters.
    if (RegDBGetAppInfo(szStrName, nvType, svStrValue, nvSize) < 0) then
        MessageBox("Failed to get application information value.", SEVERE);
        abort;
    else

        // Check to see if the value retrieved is the same as the value set.
        if (nvType != REGDB_STRING) then
            MessageBox("Type comparison failed.", WARNING);
        endif;

        if (szStrValue != svStrValue) then
            MessageBox ("Sub key value comparison Failed.", WARNING);
        else
            szMsg = "Set values: %s = %s\n\nReturn values: %s = %s";

```

```
        SprintfBox(INFORMATION, szTitle, szMsg, szStrName, szStrValue,
                  szStrName, svStrValue);
endif;

if (nvSize != StrLength(szStrValue) + 1) then
    MessageBox ("Size Comparison failed.", WARNING);
else

    szMsg = "Size in bytes entered: %d\n\nSize in bytes returned: %d";
    SprintfBox(INFORMATION, szTitle, szMsg,
              nvSize, StrLength(szStrValue) + 1);
endif;
endif;

endprogram

// Source file: Is5fn240.rul
```

{button ,PI("",`comment')} [Feedback](#)



RegDBSetDefaultRoot

Syntax

RegDBSetDefaultRoot (nRootKey);

Description

The RegDBSetDefaultRoot function sets a different root key to be used by other registry functions. Most InstallShield registry functions work on the HKEY_CLASSES_ROOT as the default base of the registry tree. Using this function, you can specify another key, such as HKEY_LOCAL_MACHINE or HKEY_CURRENT_USER or HKEY_USERS, as the root key.

RegDBSetDefaultRoot is a general registry-related function, designed to work with all registry keys, including those handled by the special registry-related functions. Refer to [special registry-related functions](#) for more information.

You cannot use RegDBSetDefaultRoot to change the root key of keys created or handled using the special registry-related functions.

Parameters

nRootKey

Specifies the name of the key to set as the root key.

Return values

0

Indicates that the function successfully set the key.

< 0

Indicates that the function was unable to set the key.

Comments

- n This function is meant only for 32-bit operating systems. On 16-bit Windows, it will always return -1. Only HKEY_CLASSES_ROOT is available under 16-bit windows, which InstallShield always uses by default for 16-bit Windows registry functions.
- n Windows NT 4.0 does not allow the creation of a key directly under HKEY_LOCAL_MACHINE.

{button ,JI('LANGREF.HLP>Examples','RegDBSetDefaultRoot_example')} [Example](#)

{button ,AL('RegDBCreateKeyEx;RegDBDeleteKey;RegDBGetKeyValueEx;RegDBKeyExist;RegDBSetKeyValueEx;Overview the registry',0,','')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



RegDBSetDefaultRoot example

```
/*-----*\
 *
 * This example illustrates the RegDBSetDefaultRoot function.
 *
 * RegDBDefaultRoot is called to set the default root key for the
 * RegDBCreateKeyEx function.
 *
 * NOTE: RegDBSetDefaultRoot and therefore this script will not run properly
 *       on a 16-bit Windows system.
 *
 *-----*/

    STRING szKey, szClass, szMsg, szTitle;
    NUMBER nRootKey;

program

    // Create a subkey in the HKEY_CLASSES_ROOT key (default).
    // Note: You can not create a key directly under
    //       HKEY_LOCAL_MACHINE on Windows NT 4.0.
    szKey   = "Test Key";
    szClass = "";
    szTitle = "RegDBSetDefaultRoot Example";

    if (RegDBCreateKeyEx(szKey, szClass) < 0) then
        MessageBox("RegDBCreateKeyEx failed.", SEVERE);
        abort;
    else
        szMsg = "Successfully created %s in HKEY_CLASSES_ROOT.";
        sprintfBox(INFORMATION, szTitle, szMsg, szKey);
    endif;

/*-----*\
 *
 * Set the root key to HKEY_LOCAL_MACHINE.
 *
 *-----*/

    nRootKey = HKEY_LOCAL_MACHINE;
    if (RegDBSetDefaultRoot(nRootKey) < 0) then
        MessageBox("First call to RegDBSetDefaultRoot failed.", SEVERE);
    else
        MessageBox("Root key successfully set to HKEY_LOCAL_MACHINE.", INFORMATION);
    endif;

    // Create a subkey in the HKEY_LOCAL_MACHINE key.
    if (RegDBCreateKeyEx(szKey, szClass) < 0) then
        MessageBox("RegDBCreateKeyEx failed.", SEVERE);
        abort;
    else

        szMsg = "Successfully created %s in HKEY_LOCAL_MACHINE";
        sprintfBox(INFORMATION, szTitle, szMsg, szKey);
    endif;

    // Delete the example subkey in HKEY_LOCAL_MACHINE.
    if (RegDBDeleteKey(szKey) < 0) then
        MessageBox("RegDBDeleteKey failed.", SEVERE);
    else
```




RegDBSetItem

Syntax

RegDBSetItem (nItem, szValue);

Description

The RegDBSetItem function assigns values under the per application paths key or the application uninstallation key, depending on the value of nItem. Calling RegDBSetItem with either the REGDB_APPPATH or the REGDB_APPPATH_DEFAULT option results in the creation of the per application paths key ([InstallationInfo](#) provides only the information used to create it).

RegDBSetItem is a special registry-related function, designed to work with certain predefined registry keys. Refer to [special registry-related functions](#) for more information.

Parameters

nItem

Specifies the item to set. Pass one of the following predefined constants in this parameter:

REGDB_APPPATH

The value of [Path] under the per-application paths key.

REGDB_APPPATH_DEFAULT

The value of [DefaultPath] under the per-application paths key.

REGDB_UNINSTALL_NAME

The value of [DisplayName] under the uninstallation key. When this constant is used, szValue specifies the application name shown in the list of uninstallable applications in Control Panel.

szValue

Specifies the value to assign to the specified item.

Return values

0

Indicates that the function successfully set the value.

< 0

Indicates that the function failed. The most common cause of failure is that InstallationInfo function was not called previously. This function cannot be executed until you call InstallationInfo.

Comments

- Before calling the RegDBSetItem function using the REGDB_APPPATH or the REGDB_APPPATH_DEFAULT option (which will result in the creation of the per application paths key as well as write a value under it), you must call the InstallationInfo function, which provides information used to create the key.
- Before calling the RegDBSetItem function using the REGDB_UNINSTALL_NAME option to assign a value to [DisplayName] under the application uninstallation key, you must call the DeinstallStart function to create the key.

{button ,JI('LANGREF.HLP>Examples','RegDBSetItem_example')} [Example](#)

{button ,AL('DeinstallStart;InstallationInfo;RegDBGetItem;Prepare my setup for uninstallation;Overview the registry',0,'')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



RegDBSetItem example

```
/*-----*\
 *
 * The following example illustrates the RegDBSetItem and RegDBGetItem
 * functions.
 *
 *-----*/

#define COMPANY_NAME      "ExampleCompany"
#define PRODUCT_NAME     "ExampleProduct"
#define VERSION_NUMBER   "5.00.00"
#define PRODUCT_KEY      "EXAMPLE.EXE"
#define DEINST_KEY       "ExampleDeinstKey"
#define APP_DEF_LOG_PATH "C:\\EXAMPLE\\TEMP"

#define APP_PATH          "C:\\EXAMPLE"
#define APP_DEF_PATH      "C:\\EXAMPLE\\TARGET"
#define UNINSTALL_NAME   "ExampleUninstallName"

    STRING  szAppPath, szAppDefPath, szUninstallName, svLogFile;
    STRING  svValue, szTitle;

program

    // Set the root key.
    RegDBSetDefaultRoot(HKEY_LOCAL_MACHINE);

    // Set installation and uninstallation information in
    // order to call RegDBSetItem and RegDBGetItem.

    InstallationInfo(COMPANY_NAME, PRODUCT_NAME, VERSION_NUMBER, PRODUCT_KEY);
    DeinstallStart(APP_DEF_LOG_PATH, svLogFile, DEINST_KEY, 0);

    szTitle          = "RegDBSetItem";
    szAppPath        = APP_PATH;
    szAppDefPath     = APP_DEF_PATH;
    szUninstallName = UNINSTALL_NAME;

/*-----*\
 *
 * Call RegDBSetItem to set the value of the application path key in the
 * registry to the value of szAppPath.
 *
 *-----*/

    if (RegDBSetItem(REGDB_APPPATH, szAppPath) < 0) then
        MessageBox("Unable to set application path key.", SEVERE);
    else
        sprintf(szTitle, "RegDBSetItem set the application " +
                "path key to %s.", szAppPath);
    endif;

/*-----*\
 *
 * Call RegDBSetItem to set the value of the application default path key in
 * the registry to the value of szAppDefPath.
 *
 *-----*/

    if (RegDBSetItem(REGDB_APPPATH_DEFAULT, szAppDefPath) < 0) then
```

```

    MessageBox("Unable to set applicatoin default path key.", SEVERE);
else
    sprintfBox(INFORMATION, szTitle, "RegDBSetItem set the application " +
              "default path key to %s.", szAppDefPath);
endif;

/*-----*\
*
* Call RegDBSetItem to set the value of the uninstall name key in the
* registry to the value of szUninstallName.
*
\*-----*/

if (RegDBSetItem(REGDB_UNINSTALL_NAME, szUninstallName) < 0) then
    MessageBox("Unable to set uninstall name key.", SEVERE);
else
    sprintfBox(INFORMATION, szTitle, "RegDBSetItem set the uninstall " +
              "name key to %s.", szUninstallName);
endif;

szTitle = "RegDBGetItem";

/*-----*\
*
* Call RegDBGetItem to get the value of the application path key in the
* registry.
*
\*-----*/

if (RegDBGetItem(REGDB_APPPATH, svValue) < 0) then
    MessageBox("Unable to get value of application path key.", SEVERE);
else
    sprintfBox(INFORMATION, szTitle, "RegDBGetItem retrieved the value " +
              "of the application path key: %s.", svValue);
endif;

/*-----*\
*
* Call RegDBGetItem to get the value of the application default path key in
* the registry.
*
\*-----*/

if (RegDBGetItem(REGDB_APPPATH_DEFAULT, svValue) < 0) then
    MessageBox("Unable to get application default path key", SEVERE);
else
    sprintfBox(INFORMATION, szTitle, "RegDBGetItem retrieved the value " +
              "of the application default path key: %s.", svValue);
endif;

/*-----*\
*
* Call RegDBGetItem to get the value of the uninstall name key in the
* registry.
*
\*-----*/

if (RegDBGetItem(REGDB_UNINSTALL_NAME, svValue) < 0) then
    MessageBox("Unable to get application uninstall name key.", SEVERE);
else
    sprintfBox(INFORMATION, szTitle, "RegDBGetItem retrieved the value " +
              "of the uninstalation name key: %s.", svValue);
endif;

```

endprogram

// Source file: Is5fn241.rul

{button ,PI("",`comment')} Feedback



RegDBSetKeyValueEx

Syntax

```
RegDBSetKeyValueEx (szKey, szName, nType, szValue, nSize);
```

Description

The RegDBSetKeyValueEx function sets the value of a specified value name under a key in the registry. If the key does not already exist, RegDBSetKeyValueEx will create it for you. However, the newly created key will not be logged for uninstallation unless it is a subkey of a key already logged for uninstallation. Keys are logged by unInstallShield for uninstallation as follow:

- n When they are created using [RegDBCreateKeyEx](#).
- n While logging is enabled.
- n When they are created automatically by InstallShield during the installation process.
- n When they are created as a result of calling one of the special registry-related functions.

If the value name does not already exist, RegDBSetKeyValueEx creates it. If the value data already exists, RegDBSetKeyValueEx overwrites it. InstallShield assumes the value name in szKey is a subkey of the HKEY_CLASSES_ROOT key. If you want to use a different main key, use the RegDBSetDefaultRoot function to set the main root key.

RegDBSetKeyValueEx is a general registry-related function, designed to work with all registry keys, including those handled by the special registry-related functions. Refer to [special registry-related functions](#) for more information.

Parameters

szKey

Specifies the name of the key to set; the key must already have been created with RegDBCreateKeyEx. You do not have to include the HKEY_CLASSES_ROOT key (or another specified key) in this parameter. Separate different levels in the subkey with a double backslash (\\).

szName

Specifies the value name for the value data to set be set. To set the default value of the key specified in szKey, pass a null string ("") in this parameter.

nType

Specifies the type of data to be set. Pass one of the following predefined constants in this parameter:

REGDB_STRING

A string variable, no newline characters allowed.

REGDB_STRING_EXPAND

A string variable holding an expandable environment variable expression, such as "%MYPATH%".

REGDB_STRING_MULTI

A string variable, newline characters allowed.

REGDB_NUMBER

A number expressed as a string and passed in string variable. Creates data of type DWORD. When nType is REGDB_NUMBER, passing a decimal string in szValue results in a numeric value being stored in the registry. This numeric value is then displayed as a hexadecimal number followed by its decimal equivalent in parentheses. You cannot pass a hexadecimal number in the string in szValue—it *must* be a decimal number.

REGDB_BINARY

Binary data stored in a string.

szValue

Specifies the value to associate with the value name. All values must be passed as string variables. Numbers must be expressed as strings (InstallShield converts them to numbers internally).

nSize

Specifies the size, in bytes, of the data to be set. You can enter -1 in this parameter when nType is REGDB_STRING, REGDB_STRING_EXPAND, or REGDB_NUMBER, and InstallShield will set the size for you. However, with REGDB_BINARY and REGDB_STRING_MULTI, you must always specify the number of bytes of binary data you are storing.

Return values**0**

Indicates that the function successfully set the key.

< 0

Indicates that the function was unable to set the key.

Comments

Under 16-bit Windows, specify the name of the key in szKey along with the data name for the value you are setting. In other words, szName must be a null string ("") and nType must always be REGDB_STRING under 16-bit Windows.

{button ,JI('LANGREF.HLP>Examples','RegDBSetKeyValueEx_example')} [Example](#)

{button ,AL('RegDBCreateKeyEx;RegDBDeleteKey;RegDBGetKeyValueEx;RegDBKeyExist;RegDBSetDefaultRoot;Overview the registry',0,'')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



RegDBSetKeyValueEx example

```
/*-----*\
 *
 * This example illustrates the RegDBSetKeyValueEx and RegDBGetKeyValueEx functions.
 *
 * RegDBCreateKeyEx is called to create a test subkey in the HKEY_CLASSES_ROOT
 * key. The value of a key is set in integer form using the REGDB_NUMBER
 * option of the RegDBSetKeyValueEx function. After this value is set, it is
 * retrieved using the RegDBGetKeyValueEx function and verified.
 *
 *-----*/

    STRING szKey, szNumName, szNumValue, svNumValue, szTitle, szMsg;
    NUMBER nType, nSize, nvType, nvSize;

program

    // Create a key to test.
    szKey      = "TestKey";
    if (RegDBCreateKeyEx(szKey, "") < 0) then
        MessageBox("RegDBCreateKeyEx failed.", SEVERE);
        abort;
    endif;

    szNumName  = "TestValue";
    szNumValue = "12345";
    nType      = REGDB_NUMBER;
    nSize      = -1;
    szTitle    = "RegDBSetKeyValueEx & RegDBGetKeyValueEx";

/*-----*\
 *
 * RegDBSetKeyValueEx is called to set a key name and a value associated with
 * it.
 *
 *-----*/

    if (RegDBSetKeyValueEx(szKey, szNumName, nType, szNumValue,
        nSize) < 0) then
        MessageBox("RegDBSetKeyValueEx failed.", SEVERE);
        abort;
    else
        // Display what RegDBSetKeyValueEx had done.
        szMsg = "%s set to: %s";
        sprintfBox(INFORMATION, szTitle, szMsg, szNumName, szNumValue);
    endif;

/*-----*\
 *
 * RegDBGetKeyValueEx is called to retrieve key value information.
 *
 *-----*/

    if (RegDBGetKeyValueEx(szKey, szNumName, nvType, svNumValue,
        nvSize) < 0) then
        MessageBox("RegDBGetKeyValueEx failed.", SEVERE);
    else

        // Check to see if the value returned is the same as the value set.
        if (nvType != REGDB_NUMBER) then
```

```
        MessageBox("Type comparison failed.", SEVERE);
    endif;

    if (svNumValue != szNumValue) then
        MessageBox("Subkey value comparison failed.", SEVERE);
    endif;

    // Display what RegDBGetKeyValueEx retrieved.
    szMsg = "%s has value: %s\n\nThis data is %d bytes.";
    sprintfBox(INFORMATION, szTitle, szMsg, szNumName, svNumValue, nvSize);
endif;

// Delete the created test key.
if (RegDBDeleteKey(szKey) < 0) then
    MessageBox("RegDBDeleteKey failed.", SEVERE);
endif;

endprogram

// Source file: Is5fn242.rul
```

{button ,PI("",`comment')} Feedback



Sd dialog box functions

InstallShield provides a number of script dialog (Sd) box functions you can customize and display. Sd dialog boxes are created using special script definition functions that create a dialog box with custom input. The dialog boxes then automatically return values to the script based on the selected action.

Sd dialog boxes that have a Cancel button do not return a CANCEL value when it is selected. Instead, the default exit handler is called. For helpful tips when using Sd dialog boxes, click [here](#).



When using SdBitmap or any Sd dialog box, remember to add `#include "Sddialog.h"` before the program block and `#include "Sddialog.rul"` after the endprogram statement. If you are using the Setup.rul generated by the Project Wizard, these lines are already included for you.

The table below lists all available Sd dialog box functions:

DialogSetInfo

Changes display elements in the dialog boxes presented by some built-in dialog box functions.

SdAskDestPath

Presents a dialog box that allows the end user to specify a destination location for the setup.

SdAskOptions

Creates a dialog box that has greater flexibility than the standard [AskOptions](#) function.

SdAskOptionsList

Presents a dialog box that allows the end user to select and deselect items from a list.

SdBitmap

Displays a bitmap in a dialog box.

SdComponentDialog

Displays a dialog box that allows end users to select folders and components to install.

SdComponentDialog2

Displays a dialog box that allows end users to select folders, components, and subcomponents to install.

SdComponentDialogAdv

Displays a dialog box that allows the end user to select the components to install. This function uses check boxes. It also provides additional data to help determine the best location for the installation.

SdComponentMult

Displays a dialog box that allows the end user to select the components and subcomponents to install. Additional information about disk space is also provided to help determine the best location for the installation.

SdConfirmNewDir

Prompts the user to confirm the folder selection.

SdConfirmRegistration

Prompts the end user to confirm the information entered in dialog boxes presented by SdRegisterUser or SdRegisterUserEx.

SdDisplayTopics

Displays a list of topics.

SdFinish

Displays a dialog box that informs the end user that the setup is complete and offers a choice of options, such as whether to view an information file or launch an application.

SdFinishReboot

Displays a dialog box that informs the user that the setup is complete and offers a choice of options for restarting Windows and the computer.

SdInit

Prepares a setup for calls to Sd dialog box functions.

SdLicense

Displays a license agreement and gives the end user the option of accepting or rejecting the license terms.

SdMakeName

Creates a section name for a custom dialog. This section name is used in writing to and reading from an .iss file, which is used by InstallShield Silent.

SdOptionsButtons

Displays a dialog box with user-defined buttons that provide an end user with various options.

SdProductName

Inserts your product name in certain static fields of the script dialog boxes.

SdRegisterUser

Displays a dialog box for entering a user name and company name.

SdRegisterUserEx

Displays a dialog box in which the end user can enter a user name, company name, and application serial number.

SdSelectFolder

Presents a dialog box that allows the end user to select a folder from a list of program folders.

SdSetupType

Displays a dialog box that enables the end user to select one of the three standard setup types: Typical, Compact, or Custom.

SdSetupTypeEx

Displays a dialog box that allows the end user to select standard or custom setup types.

SdShowAnyDialog

Displays a general-purpose dialog box from a resource DLL. You cannot receive any input from the end user when showing a dialog box with SdShowAnyDialog function.

SdShowDlgEdit1

Displays a dialog box that has one single-line edit field and other static controls.

SdShowDlgEdit2

Displays a dialog box that has two single-line edit fields and other static controls.

SdShowDlgEdit3

Displays a dialog box that has three single-line edit fields and other static controls.

SdShowFileMods

Presents a dialog box that previews the changes that may be made to a file and allows the end user to approve the changes, reject the changes, or request that the changes be written to a file.

SdShowInfoList

Displays a scrollable list of messages in a dialog box.

SdShowMsg

Displays a message in a small window.

SdStartCopy

Presents a dialog box that displays the options and settings that have been specified by the end user.

SdWelcome

Displays a general-purpose greeting.

{button ,AL(`Tips for using Sd dialog boxes;Exit conditions;Caching dialogs;Displaying special characters and TM in dialogs and titles;Which fonts InstallShield dialog boxes use;Modifying Sd dialog box source files',0,','')} [See also](#)

{button ,PI('"'',`comment')} [Feedback](#)



SdAskDestPath

Syntax

SdAskDestPath (szTitle, szMsg, svDir, nReserved);

Descriptions

The SdAskDestPath function creates a dialog box that allows the end user to select an alternate destination path. When you click the Browse button in that dialog box, the [SelectDir](#) function is called to open a second dialog box that enables the end user either to select an existing folder or to enter a new folder name.



[View sample dialog](#)

If the end user enters the name of a folder that does not exist, a message box is displayed asking whether to create a folder with that name. If you select yes, the specified directory is created. The fully qualified name of the alternate directory you select is assigned to svDir.



If the default folder specified by svDir does not already exist on the end user's system, it will not be created unless the end user presses the Browse button and follows the steps to create it from the Choose Folder dialog box. Therefore, whenever you specify a default folder that you intend to use before calling [ComponentMoveData](#) (which will create the folder if necessary), you must call [ExistsDir](#) when SdAskDestPath returns in order to determine whether that folder exists. If it does not exist, call [CreateDir](#) to create it on the end user's system.



When using SdAskDestPath function, remember to add `#include "Sddialog.h"` before the program block and `#include "Sddialog.rul"` after the endprogram statement. If you are using the Setup.rul generated by the Project Wizard, these lines are already included for you.

Parameters

szTitle

Specifies the title of the dialog box. To display the default title ("Choose Folder"), pass a null string ("") in this parameter.

szMsg

Specifies the text to display in the dialog box. The text is considered a static control. Use the %P place holder in your message string to insert the product name (if any) that has been specified by a previous call to [SdProductName](#). To display the default instructions for this dialog box, pass a null string ("").

svDir

Specifies the name of the directory to be selected by default. Returns the name of the directory selected by the end user.

nReserved

Pass zero in this parameter. No other value is allowed.

Return values

NEXT

Indicates that the Next button was clicked.

BACK

Indicates that the Back button was clicked.

Comments

- Ⓝ If the default folder specified by `svDir` does not already exist on the end user's system, it will not be created unless the end user presses the Browse button and follows the steps to create it from the Choose Folder dialog box. Therefore, whenever you specify a default folder that you intend to use before calling [ComponentMoveData](#) (which will create the folder if necessary), you must call [ExistsDir](#) when `SdAskDestPath` returns in order to determine whether that folder exists. If it does not exist, call [CreateDir](#) to create it on the end user's system.
- Ⓝ Setups that will run in silent mode should create the new folder if it does not exist before calling `SdAskDestPath`. Doing so will ensure that the confirmation dialog is not displayed. Without this step, two response files would be required to handle the two possible conditions.

{button ,JI('LANGREF.HLP>Examples','SdAskDestPath_example')} [Example](#)

{button ,AL('SdConfirmNewDir;Tips for using Sd dialog boxes;Exit conditions;Caching dialogs;Displaying special characters and TM in dialogs and titles;Which fonts InstallShield dialog boxes use;Modifying Sd dialog box source files',0,','')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



SdAskDestPath example

```
/*-----*\
 *
 * This example illustrates the use of the SdAskDestPath dialog box.
 *
 * SdProductName is called to set the %P parameters in SdAskDestPath. The
 * user is then prompted for a path when SdAskDestPath is called. The return
 * value of svDir is set to the TARGETDIR constant. This constant is
 * displayed in the final message box.
 *
 \*-----*/

#include "Sddialog.h";

STRING szTitle, szMsg, svDir;

program

    // Set product name.
    SdProductName("Example Product 3.1");

    // Back button not required for initial dialog box.
    Disable(BACKBUTTON);

    szTitle = "SdAskDestPath Example";
    svDir   = "C:\\EXAMPLE\\TARGET";

    szMsg   = ""; // Display default message.

/*-----*\
 *
 * The following displays the SdAskDestPath dialog box.
 *
 \*-----*/
    if (SdAskDestPath(szTitle, szMsg, svDir, 0) = NEXT) then

        TARGETDIR = svDir;
    endif;

    Enable(BACKBUTTON);

    // Display the new target directory.
    SprintfBox(INFORMATION, "SdAskDestPath", "Successful.\n\nThe Target " +
        "directory is: " + TARGETDIR);

endprogram

#include "Sddialog.rul"

// Source file: Is5fn131.rul
```

{button ,PI("",`comment')} [Feedback](#)



SdBitmap

Syntax

SdBitmap (szTitle, szMsg, szBitmap);

Description

The SdBitmap function displays a bitmap in a dialog box. The maximum allowable size of the bitmap is 440 pixels wide by 275 pixels high. You can also display a message in the SdBitmap dialog box, but only if you use a resource editor to modify the SdBitmap dialog box resource so that the control that displays the message is made visible. See the Comments section, below.



[View sample dialog](#)



When using SdBitmap or any Sd dialog box, remember to add `#include "Sddialog.h"` before the program block and `#include "Sddialog.rul"` after the endprogram statement. If you are using the Setup.rul generated by the Project Wizard, these lines are already included for you.

Parameters

szTitle

Specifies the title of the dialog box. To display the default title ("Welcome"), pass a null string ("") in this parameter.

szMsg

Pass a null string ("") in this parameter unless you use a resource editor to modify the SdBitmap dialog box to display a message. See the Comments section, below.

szBitmap

Specifies the filename of the bitmap to display. If the filename is unqualified (that is, if it does not include a drive designation and path), InstallShield searches for the bitmap in SUPPORTDIR.

Return values

NEXT

Indicates that the Next button was clicked.

BACK

Indicates that the Back button was clicked.

Comments

- n You can use a resource editor to modify the SdBitmap dialog box resource so that a message string passed as the parameter szMsg is displayed in the SdBitmap dialog box.
- n The SdBitmap dialog box resource is contained in _isres.dll. The resource contains a static text control that receives the string passed as the parameter szMsg. However, by default this static text control is out of view in the SdBitmap dialog box (below the dialog box). SdBitmap also uses a static text control to display the bitmap image. You can resize the bitmap image static text control and move the message static text control into view in the dialog box. The message in szMsg will then be visible when SdBitmap is called.
- n Be aware that changing the size of the bitmap image static text control may affect the display of your bitmap image. The bitmap image must be small enough to avoid being clipped when SdBitmap centers it in the bitmap image static text control.
- n This function does not support transparent bitmaps. If you use a transparent bitmap with this function, the transparent portions will be displayed normally.
- n When a metafile is displayed with SdBitmap, it is resized automatically to fit the dialog box; calling [SizeWindow](#) with the METAFILE parameter has no effect on metafiles that are displayed with this function.

{button ,JI('LANGREF.HLP>Examples','SdBitmap_example')} Example

{button ,AL('Tips for using Sd dialog boxes;Which graphic formats InstallShield supports;Using metafiles as images;Preventing color distortion;Using transparent bitmaps;Exit conditions;Caching dialogs;Displaying special characters and TM in dialogs and titles;Which fonts InstallShield dialog boxes use;Modifying Sd dialog box source files',0,'')} See also

{button ,PI('','comment')} Feedback



SdBitmap example

```
/*-----*\
 *
 * This example illustrates the usage of the SdBitmap dialog box function.
 *
 * NOTE: In order for this script to run properly, it is advised to set the
 *       BITMAP_FILE constant to a valid bitmap file on the target system.
 *
 *-----*/

#define BITMAP_FILE "EXAMPLE\\SOURCE\\PROGRAM\\DEMO.BMP"

#include "Sddialog.h"

    STRING    szMsg, szBitmap, szTitle;

program

    szTitle    = "SdBitmap Example";
    szMsg      = "";
    szBitmap   = BITMAP_FILE;

/*-----*\
 *
 * The following displays BITMAP_FILE from within a dialog box.
 *
 *-----*/
    SdBitmap (szTitle, szMsg, szBitmap);

endprogram

#define SD_SINGLE_DIALOGS 1
#define SD_BITMAP        1
#include "Sddialog.rul"

// Source file: Is5fn133.rul
```

{button ,PI("",`comment')} [Feedback](#)



SdComponentDialog2

Syntax

SdComponentDialog2 (szTitle, szMsg, szDir, szComponents);

Description

The SdComponentDialog2 function creates a dialog box that displays a list of components in the current media that the user can install. The component displayed in the Components window can have subcomponents. If a component has subcomponents, the Change button will become active. Clicking on the Change button will bring up the Select Subcomponents dialog box, where further selections can be made. For each component or subcomponent, the description can also be provided. This description will be displayed under the Description field of the dialog box when the user selects or highlights that component.



View sample dialog



If your setup does not use a setup type dialog, you *must* call ComponentSetupTypeSet to specify a setup type that has been defined in the IDE Setup Types pane before calling SdComponentDialog2.

The name of the current media is stored in the system variable MEDIA. During setup initialization, InstallShield assigns to MEDIA the default media name ("DATA"), which is associated with your [file media library](#) (Data1.cab). To display script-created components, follow these steps:

1. Save the current value of MEDIA.
2. Assign to MEDIA the name of the script-created media.
3. Call SdComponentDialog2 to get end-user selections.
4. Assign to MEDIA the value saved in step 1.



The value of the system variable MEDIA is set to 'DATA' when during setup initialization. If you change the value of this variable to refer to a script-created component set, you must change the value back to 'DATA' before calling ComponentMoveData, CreateShellObjects, or CreateRegistrySet.



When using SdComponentDialog2 function, remember to add `#include "Sddialog.h"` before the program block and `#include "Sddialog.rul"` after the endprogram statement. If you are using the Setup.rul generated by the Project Wizard, these lines are already included for you.

Parameters

szTitle

Specifies the title of the dialog box. To display the default title ("Select Components"), pass a null string ("") in this parameter.

szMsg

Specifies the message to display in the dialog box. To display the default instructions for this dialog box, pass a null string ("") in this parameter.

szDir

Specifies the name of the target directory (destination location). Note that the destination folder specified by svDir is not assigned automatically to TARGETDIR or any other system variable. To apply the value of svDir to the setup, you must assign it to TARGETDIR or to a script-defined variable, if one is in use.

szComponents

Specifies the name of the component whose subcomponents are to be displayed. To display all main components, pass a null string ("") in this parameter. For more information about specifying components and

subcomponents in function calls, click [here](#).

Return values

NEXT

Indicates that the Next button was clicked.

BACK

Indicates that the Back button was clicked.

Comments

- n The Change button will be active only if the selected component has any subcomponents. Otherwise, it will turn gray.
- n If a component is deselected by default, then its subcomponents also should be deselected by default. Likewise, if all subcomponents of a component are deselected by default, the parent component also should be deselected by default. Refer to the [ComponentAddItem](#) function for information on default component and subcomponent selection settings.



Default selection settings are cleared when the user selects a component or subcomponent displayed in a dialog box. If the user deselects a component, all of its subcomponents will be deselected. If the user deselects all of a component's subcomponents, the component will be deselected.

{button ,JI('LANGREF.HLP>Examples','SdComponentDialog2_example')} [Example](#)

{button ,AL('ComponentAddItem;ComponentDialog;ComponentGetItemSize;ComponentIsItemSelected;ComponentListItems;ComponentSelectItem;SdComponentDialog;SdComponentMult;SdComponentDialogAdv;Tips for using Sd dialog boxes;Displaying icons in component dialogs;Exit conditions;Caching dialogs;Displaying special characters and TM in dialogs and titles;Which fonts InstallShield dialog boxes use;Modifying Sd dialog box source files',0,';')}
[See also](#)

{button ,PI('','comment')} [Feedback](#)



SdComponentDialog2 example

```
/*-----*\
 *
 * This example demonstrates ComponentSetupTypeGetData, ComponentGetData,
 * ComponentSetData, SdComponentDialog2, and ComponentSelectedItem.
 *
 * Comments: To run this example script, create a project with the
 *           following components (c), subcomponents (sc), and file
 *           groups (fg):
 *
 *           (c) Program Files
 *               (fg) Program DLLS
 *               (fg) Program EXEs
 *           (c) Example Files
 *               (sc) Small Documents
 *                   (fg) Small Document Examples
 *               (sc) Books
 *                   (fg) Book Examples
 *               (sc) Graphics
 *                   (fg) Graphic Examples
 *           (c) Help Files
 *               (fg) Help Files
 *           (c) Utilities
 *               (sc) Grammar Checker
 *                   (fg) Grammar Checker
 *               (sc) Art Studio
 *                   (fg) Art Studio
 *
 * Be sure to enter descriptions into the Description fields of the component
 * properties sheets for the Program Files and Example Files components and
 * their subcomponents.
 *
 *-----*/

#include "sddialog.h"

// Define strings. In a real setup you would define these in your string tables
// and precede each constant with @ to use them in your script.
#define COMP_SELECT_TITLE           "Select Components"
#define COMP_SELECT_MSG1           "IMPORTANT! Note the various component "
#define COMP_SELECT_MSG2           "and subcomponent names, descriptions, "
#define COMP_SELECT_MSG3           "and selection settings."
#define COMP_SELECT_MSG4           "IMPORTANT! Note the CHANGED component "
#define COMP_SELECT_MSG5           "and subcomponent name, description, "
#define COMP_SELECT_MSG6           "and selection settings."
#define COMP_PROGRAMFILES_DISPLAYNAME "Program Files"
#define COMP_EXAMPLEFILES_DISPLAYNAME "Example Files"
#define COMP_SMALLDOCUMENTS_DISPLAYNAME "Small Documents"
#define COMP_BOOKS_DISPLAYNAME      "Books"
#define COMP_GRAPHICS_DISPLAYNAME   "Graphics"
#define SETUP_TYPE                  "Typical"

// Global variable declarations.
STRING  svInfo, szInfo, szComponent;
NUMBER  nvInfo, nInfo, nResult;

program

    // Get the description field data for the SETUP_TYPE setup type.
```

```

ComponentSetupTypeGetData( MEDIA, SETUP_TYPE, SETUPTYPE_INFO_DESCRIPTION,
                           nvInfo, svInfo );
sprintfBox( INFORMATION, "ComponentSetupTypeGetData demo",
           "ComponentSetupTypeGetData got the following " +
           "value from the " + SETUP_TYPE + " description field:\n\n%s",
           svInfo );

// Get the description field data for the COMP_PROGRAMFILES_DISPLAYNAME
// component using ComponentGetData.
szComponent = COMP_PROGRAMFILES_DISPLAYNAME;
nResult = ComponentGetData( MEDIA, szComponent, COMPONENT_FIELD_DESCRIPTION,
                           nvInfo, svInfo );
sprintfBox( INFORMATION, "ComponentGetData demo",
           "ComponentGetData got the following value " +
           "from the " + COMP_PROGRAMFILES_DISPLAYNAME +
           " description field:\n\n%s", svInfo );

// Show the original description field values in the component selection dialog.
Disable( BACKBUTTON); // Back button not needed or handled here.
SdComponentDialog2( COMP_SELECT_TITLE, COMP_SELECT_MSG1 + COMP_SELECT_MSG2 +
                   COMP_SELECT_MSG3, TARGETDIR, "" );

// Change the displayed names for the Program Files component and the
// Example Files component and subcomponents.
szInfo = "CHANGED Component Name!";
nResult = ComponentSetData( MEDIA, szComponent, COMPONENT_FIELD_DISPLAYNAME,
                           nInfo, szInfo );
szComponent = COMP_EXAMPLEFILES_DISPLAYNAME;
nResult = ComponentSetData( MEDIA, szComponent, COMPONENT_FIELD_DISPLAYNAME,
                           nInfo, szInfo );
szComponent = COMP_EXAMPLEFILES_DISPLAYNAME + "\\\" +
              COMP_SMALLDOCUMENTS_DISPLAYNAME;
nResult = ComponentSetData( MEDIA, szComponent, COMPONENT_FIELD_DISPLAYNAME,
                           nInfo, szInfo );
szComponent = COMP_EXAMPLEFILES_DISPLAYNAME + "\\\" +
              COMP_BOOKS_DISPLAYNAME;
nResult = ComponentSetData( MEDIA, szComponent, COMPONENT_FIELD_DISPLAYNAME,
                           nInfo, szInfo );
szComponent = COMP_EXAMPLEFILES_DISPLAYNAME + "\\\" +
              COMP_GRAPHICS_DISPLAYNAME;
nResult = ComponentSetData( MEDIA, szComponent, COMPONENT_FIELD_DISPLAYNAME,
                           nInfo, szInfo );

// Change the descriptions displayed for the Program Files component
// and the Example Files component and subcomponents.
szComponent = COMP_PROGRAMFILES_DISPLAYNAME;
szInfo = "CHANGED description field value!";
nResult = ComponentSetData( MEDIA, szComponent, COMPONENT_FIELD_DESCRIPTION,
                           nInfo, szInfo );
szComponent = COMP_EXAMPLEFILES_DISPLAYNAME;
nResult = ComponentSetData( MEDIA, szComponent, COMPONENT_FIELD_DESCRIPTION,
                           nInfo, szInfo );
szComponent = COMP_EXAMPLEFILES_DISPLAYNAME + "\\\" +
              COMP_SMALLDOCUMENTS_DISPLAYNAME;
nResult = ComponentSetData( MEDIA, szComponent, COMPONENT_FIELD_DESCRIPTION,
                           nInfo, szInfo );
szComponent = COMP_EXAMPLEFILES_DISPLAYNAME + "\\\" +
              COMP_BOOKS_DISPLAYNAME;
nResult = ComponentSetData( MEDIA, szComponent, COMPONENT_FIELD_DESCRIPTION,
                           nInfo, szInfo );
szComponent = COMP_EXAMPLEFILES_DISPLAYNAME + "\\\" +
              COMP_GRAPHICS_DISPLAYNAME;
nResult = ComponentSetData( MEDIA, szComponent, COMPONENT_FIELD_DESCRIPTION,

```

```
        nInfo, szInfo );

// Deselect the Program Files and Example Files components (and all their
// subcomponents, by extension).
ComponentSelectItem( MEDIA, COMP_PROGRAMFILES_DISPLAYNAME, FALSE );
ComponentSelectItem( MEDIA, COMP_EXAMPLEFILES_DISPLAYNAME, FALSE );

// Display the components again, noting the changed names, descriptions,
// and selection settings.
SdComponentDialog2( COMP_SELECT_TITLE, COMP_SELECT_MSG4 + COMP_SELECT_MSG5 +
                    COMP_SELECT_MSG6, TARGETDIR, "" );
endprogram

#include "sddialog.rul"

// Source file: IS5FN623.rul
```

{button ,PI("",`comment')} [Feedback](#)



SdConfirmNewDir

Syntax

SdConfirmNewDir (szTitle, szDir, nReserved);

Description

The SdConfirmNewDir function creates a dialog box that displays a folder name and a prompt for confirmation. If the end user clicks on the Yes button, the new folder is created automatically by this function.



[View sample dialog](#)



When using SdConfirmNewDir, remember to add `#include "Sddialog.h"` before the program block and `#include "Sddialog.rul"` after the endprogram statement. If you are using the Setup.rul generated by the Project Wizard, these lines are already included for you.

Parameters

szTitle

Specifies the title of the dialog box. To display the default title ("Confirm New Folder"), pass a null string ("") in this parameter.

szDir

Specifies the name of the directory to confirm. (Obtain this information by calling [SdAskDestPath](#).)

nReserved

Pass zero in this parameter. No other value is allowed.

Return values

YES

Indicates that the Yes button was clicked—the directory has been confirmed and will be created.

NO

Indicates that the No button was clicked—the specified directory will not be created.

<0

Indicates Yes was selected but the function was unable to create the new directory.

{button ,JI('LANGREF.HLP>Examples','SdConfirmNewDir_example')} [Example](#)

{button ,AL('SdAskDestPath;Tips for using Sd dialog boxes;Exit conditions;Caching dialogs;Displaying special characters and TM in dialogs and titles;Which fonts InstallShield dialog boxes use;Modifying Sd dialog box source files',0,','')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



SdConfirmNewDir example

```
/*-----*\
 *
 * This example illustrates the usage of the SdConfirmNewDir dialog box.
 *
 * First, we get the destination directory from the user using the
 * SdAskDestPath function. If the directory does not exist, then it is
 * created and confirmed using SdConfirmNewDir.
 *
 * NOTE: This script creates directories on the local hard disk.
 *
\*-----*/

#include "Sddialog.h"

STRING  szTitle, szDir, szMsg;
NUMBER  nReturn;

program

    Disable(BACKBUTTON);

    szTitle = "SdConfirmNewDir Example";
    szDir    = "C:\\NONEXIST\\DIR";
    szMsg    = "Select destination directory:";

    start:

    // Retrieve destination directory from user.
    nReturn = SdAskDestPath(szTitle, szMsg, szDir, 0);

    // Check if the selected directory exists.
    if (ExistsDir(szDir) = EXISTS) then

        szMsg = "Directory '%s' exists!\n\nIn order for this example to " +
                "run properly, please select a nonexistent directory.";
        sprintfBox (INFORMATION, szTitle, szMsg, szDir);
        goto start;
    else

/*-----*\
 *
 * Since the directory does not exist, confirm if the user would like it
 * created.
 *
\*-----*/

        nReturn = SdConfirmNewDir(szTitle, szDir, 0);
        endif;

        if (nReturn = NO) then

            // The user did not want it created, select another directory.
            MessageBox("Selected directory was not created.", INFORMATION);
            goto start;
        elseif (nReturn = YES) then

            // The user did want the directory created. SdConfirmNewDir creates the
            // selected directory.
            sprintfBox (INFORMATION, szTitle, "%s directory successfully created.",
                szDir);
```

```
elseif (nReturn < 0) then
    MessageBox("SdConfirmNewDir failed.", SEVERE);
    abort;
endif;

// After finishing the above steps, the directory for installation is
// available. Carry out the installation. . .

endprogram

#define SD_SINGLE_DIALOGS 1
#define SD_CONFIRMNEWDIR 1
#define SD_ASKDESTPATH 1
#include "Sddialog.rul"

// Source file: Is5fn138.rul
```

{button ,PI("",`comment')} Feedback



SdFinish

Syntax

SdFinish (szTitle, szMsg1, szMsg2, szOpt1, szOpt2, bvOpt1, bvOpt2);

Description

The SdFinish function displays a dialog box to inform the end user that the installation is complete and to give the user information or options. The SdFinish dialog box displays up to two messages and two check box selection options. For example, you may want to offer the user the choice of either viewing a README file or launching the application.



[View sample dialog](#)

You can use the place holder %P in strings passed as parameters szMsg1, szMsg2, szOpt1, and szOpt2. When these strings are displayed, %P is replaced by the product name made available with the SdProductName function.



When using SdFinish , remember to add `#include "Sddialog.h"` before the program block and `#include "Sddialog.rul"` after the endprogram statement. If you are using the Setup.rul generated by the Project Wizard, these lines are already included for you.

Parameters

szTitle

Specifies the title of the dialog box. To display the default title ("Setup Complete"), pass a null string ("") in this parameter.

szMsg1

Specifies the message to display at the top of the dialog box. To display the default instructions, which inform the user that the installation is complete, pass a null string ("") in this parameter.

szMsg2

Specifies the message to display at the bottom of the dialog box. To display the default instructions ("Click Finish to complete Setup"), pass a null string ("") in this parameter.

szOpt1

Specifies the text to display beside the first check box. Pass a null string ("") in this parameter to hide the check box.

szOpt2

Specifies the text to display beside the second check box. Pass a null string ("") in this parameter to hide the check box.

bvOpt1

Returns the selection state (TRUE or FALSE) of the first check box.

bvOpt2

Returns the selection state (TRUE or FALSE) of the second check box.

Return values

NEXT

Indicates that the Finish button was clicked.



Because SdFinish is designed to announce the end of the installation, the Back button is disabled.

Comments

- n SdFinish does not end the installation. Your script must end the installation with an exit statement or by calling a function that ends the installation, such as System.
- n Since InstallShield will make every attempt **not** to restart Windows or the system when other instances of InstallShield are running, you must make sure all other instances of InstallShield are shut down before restarting Windows or the system. In addition, your message to the user should request that they ensure all other applications are shut down before restarting Windows or the system.
- n To handle files that were in use during file transfer and recorded for update, you must test the value of the system variable BATCH_INSTALL before calling System to restart the system. If BATCH_INSTALL is TRUE, then locked .dll and .exe files have been marked and [CommitSharedFiles](#) must be called before calling System in order to commit the files for updating the when the system is rebooted.
- n SdFinishReboot, which automatically calls CommitSharedFiles, is preferable to SdFinish.

{button ,JI('LANGREF.HLP>Examples','SdFinish_example')} [Example](#)

{button ,AL('CommitSharedFiles;SdFinishReboot;SdProductName;System;VerUpdateFile;Tips for using Sd dialog boxes;Exit conditions;Caching dialogs;Displaying special characters and TM in dialogs and titles;Which fonts InstallShield dialog boxes use;Modifying Sd dialog box source files',0,'')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



SdFinish example

```
/*-----*\
 *
 * This example illustrates the use of the functions SdProductName,
 * SdWelcome, SdSetupType, and SdFinish.
 *
 * First, SdProductName is called to establish the product name to
 * substitute for the %P place holder, which is embedded in the
 * message strings passed to SdWelcome, SdSetupType, and SdFinish.
 *
 * Next, SdWelcome is called to display a welcome message and then
 * SdSetupType is called. Finally, a call to SdFinish completes the
 * setup process by giving the user final options.
 *
 *-----*/

#include "Sddialog.h"

STRING  szProductName, szTitle, szMsg, svDir, szComponents;
STRING  szMsg1, szMsg2, szOpt1, szOpt2;
BOOL    bvOpt1, bvOpt2;
NUMBER  nReturn;

program

//Set the product name to substitute for the %P place holder.
szProductName = "My Application";
SdProductName(szProductName);

SdWelcomeLabel:
szTitle = "SdWelcome Example";
Disable(BACKBUTTON);

//Display the SdWelcome dialog box. The null string in parameter
//two specifies the default message, which uses the %P place holder.
SdWelcome(szTitle, "");

Enable(BACKBUTTON);

SdSetupTypeLabel:
szTitle = "SdProductName Example";

//The %P place holder is embedded in a custom message that will be
//passed to SdSetupType.
szMsg   = "Choose the type of installation for %P.";
svDir   = "C:\\EXAMPLE";

// Display the SdSetupType dialog box.
if (SdSetupType(szTitle, szMsg, svDir, 0) = BACK) then

    goto SdWelcomeLabel;
endif;

// The %P place holder is embedded in several of the string
// parameters that will be passed to SdFinish.
szTitle = "SdFinish Example";
szMsg1  = "%P Setup is almost complete.\nChoose the options you want below.";
szMsg2  = "Click Finish to complete %P Setup.";
szOpt1  = "I would like to view the README file.";
szOpt2  = "I would like to launch %P.";
```

```
Enable(BACKBUTTON);

// Display the SdFinish dialog box.
SdFinish(szTitle, szMsg1, szMsg2, szOpt1, szOpt2, bvOpt1, bvOpt2);

if(bvOpt1) then
    MessageBox("Display readme file here.", INFORMATION);
endif;

if(bvOpt2) then
    SprintfBox(INFORMATION, szTitle, "Launch %s here.", szProductName);
endif;

endprogram

#include "Sddialog.rul"

// Source file: Is5fn142.rul
```

{button ,PI("",`comment')} [Feedback](#)



SdFinishReboot

Syntax

SdFinishReboot (szTitle, szMsg1, nDefOption, szMsg2, nReserved);

Description

The SdFinishReboot function at the end of your installation announces that the installation is complete and gives the user the option to restart the system. Restarting the system allows changes to Autoexec.bat, Config.sys, and some .ini files to take effect.

The option to restart just Windows is displayed only when the target system is running Windows 3.1. InstallShield automatically determines the target operating system and displays the appropriate options. You can set a default option selection.

The SdFinishReboot dialog box displays up to two messages in static text fields. You can use the %P place holder in the strings passed as the parameters szMsg1, szMsg2, szOpt1, and szOpt2. When these strings are displayed, %P is replaced by the product name made available with the SdProductName function.



View sample dialog



When using SdFinishReboot , remember to add `#include "Sddialog.h"` before the program block and `#include "Sddialog.rul"` after the endprogram statement. If you are using the Setup.rul generated by the Project Wizard, these lines are already included for you.

Parameters

szTitle

Specifies the title of the dialog box. To display the default title ("Setup Complete"), pass a null string ("") in this parameter.

szMsg1

Specifies the text to display at the top of the dialog box, informing the user about the end of the installation. To display the default instructions for this dialog box, pass a null string ("") in this parameter.

nDefOption

Specifies a default radio button option selection. Pass one of the following predefined constants in this parameter:

SYS_BOOTWIN

Restart Windows (Windows 3.1 only).

SYS_BOOTMACHINE

Reboot the machine.

0

Do not reboot or restart Windows.

szMsg2

Specifies the text to display at the bottom of the dialog box, giving the user information about what to do. To display the default instructions, pass a null string ("").

nReserved

Pass zero in this parameter. No other value is allowed.

Return values

NEXT

Indicates that the user chose not to reboot the system or restart Windows.

<0

Indicates that the user chose to either reboot the system or restart Windows, but the reboot or restart failed.



Because SdFinishReboot is designed to announce the end of the installation, the Back button is disabled.

Comments

- n If the default option selection in nDefOption is set to SYS_BOOTWIN (to restart Windows 3.1) and InstallShield determines that the target system is using Windows NT 4.0 or Windows 95, InstallShield changes the default to SYS_BOOTMACHINE (to reboot the machine).
- n Since InstallShield will make every attempt *not* to restart Windows or the system when other instances of InstallShield are running, you must make sure all other instances of InstallShield are shut down before allowing SdFinishReboot to restart Windows or the system. Additionally, your message to the user should request that if they will restart Windows or the system later, they should ensure all other applications are shut down first.
- n InstallShield automatically calls the [CommitSharedFiles](#) function when SdFinishReboot executes, regardless of which restart option was selected. This ensures that locked .dll and .exe files will be updated the next time the system starts.

{button ,JI('LANGREF.HLP>Examples','SdFinishReboot_example')} [Example](#)

{button ,AL('CommitSharedFiles;RebootDialog;SdFinish;SdProductName;Tips for using Sd dialog boxes;Exit conditions;Caching dialogs;Displaying special characters and TM in dialogs and titles;Which fonts InstallShield dialog boxes use;Modifying Sd dialog box source files',0,','')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



SdFinishReboot example

```
/*-----*\
*
* This example illustrates the usage of the SdFinishReboot dialog box.
*
* SdProductName is called to enable the %P parameter in strings used for
* SdFinishReboot. SdFinishReboot is then called, prompting the user for
* input.
*
* NOTE: In this script, if the reboot selection is made, the system will
*       reboot.
*
\*-----*/

#include "Sddialog.h";

STRING szProduct, szTitle, szMsg1, szMsg2;
NUMBER nDefOption, nReserved;

program

    szProduct = "Your Product";
    SdProductName(szProduct);

    szTitle    = "SdFinishReboot Example";
    szMsg1     = "Setup has completed installing %P.";
    nDefOption = 0;
    szMsg2     = "Click Finish to exit %P setup.";
    nReserved  = 0;

/*-----*\
*
* SdFinishReboot is called; if the reboot option is selected, the system will
* reboot.
*
\*-----*/
    if (SdFinishReboot(szTitle, szMsg1, nDefOption, szMsg2,
                       nReserved) < 0) then

        MessageBox("SdFinishReboot failed..", SEVERE);
    endif;

endprogram

#define SD_SINGLE_DIALOGS 1
#define SD_PRODUCTNAME   1
#define SD_FINISHREBOOT  1

#include "Sddialog.rul"

//Source File : Is5fn143.rul
```

{button ,PI("",`comment')} [Feedback](#)



Sdlnit

Syntax

Sdlnit ();

Description

The Sdlnit function prepares a setup for Sd dialog box function calls by loading required resource strings, restoring the InstallShield window if it is minimized, and specifying Windows 95-style check boxes in Sd dialog boxes.

Parameters

This function takes no parameters.

Return values

0

Indicates that the setup was initialized for Sd dialog box function calls.

1

Indicates that the setup has already been initialized for Sd dialog box function calls.

Comment

- n This function is called automatically by each Sd function. It is unnecessary to call Sdlnit explicitly unless your script calls DialogSetInfo before calling any of the Sd dialog functions. In that case, your script must call Sdlnit before it calls DialogSetInfo; otherwise the call to DialogSetInfo will have no effect.
- n A call to Sdlnit is inserted automatically into all project wizard-generated scripts.

{button ,JI('LANGREF.HLP>Examples','Sdlnit_example')} [Example](#)

{button ,AL('DialogSetInfo;Tips for using Sd dialog boxes;Exit conditions;Caching dialogs;Displaying special characters and TM in dialogs and titles;Which fonts InstallShield dialog boxes use;Modifying Sd dialog box source files',0,'')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



SdInit example

```
/*-----*\
 *
 * This example illustrates the use of the SdInit function.
 *
 * Note that SdInit must be called explicitly in this script because
 * DialogSetInfo is called before any Sd dialog box function have
 * been called.
 *
 \*-----*/

#include "sddialog.h"

STRING szInfoString;

program

    //Initialize setup for calls to Sd dialog box functions
    SdInit();

    //Set check box style for component selection
    DialogSetInfo ( DLG_INFO_CHECKSELECTION , szInfoString , CHECKBOX );

    //Get component selections
    SdComponentDialog2 ( "" , "" , TARGETDIR , "" );

endprogram

#include "sddialog.rul"

// Source file: is5fn209.rul
```

{button ,PI("",`comment')} [Feedback](#)



SdLicense

Syntax

SdLicense (szTitle, szMsg, szQuestion, szLicenseFile);

Description

The SdLicense function displays a dialog box containing a license agreement in a multi-line edit field. The license agreement is stored in a text file identified in the parameter szLicenseFile.

The user can scroll up and down to read the agreement, then must choose either the Yes, No, or, if enabled, Back button. Since this is likely the first dialog you would display, you may want to disable the Back button. If the user selects Yes, Setup will continue. If the user selects No, InstallShield will display the Exit Setup dialog.



[View sample dialog](#)



When using SdLicense , remember to add `#include "Sddialog.h"` before the program block and `#include "Sddialog.rul"` after the endprogram statement. If you are using the Setup.rul generated by the Project Wizard, these lines are already included for you.

Parameters

szTitle

Specifies the title of the dialog box. To display the default title ("Software License Agreement"), pass a null string ("") in this parameter.

szMsg

Specifies the message to display in the static text field above the multi-line edit field. To display the default instructions, pass a null string ("") in this parameter.

szQuestion

Specifies the text to display in the static text field below the multi-line edit field. You would likely place a question here, to which the user should respond by selecting either Yes and No. To display the default instructions, pass a null string ("") in this parameter.

szLicenseFile

Specifies the name of the text file that contains the license agreement. This file must be added to the appropriate language folder in the Setup Files pane.

Return values

YES

Indicates that the user selected the Yes button.

BACK

Indicates that the user selected the Back button.

Comments

- n This function cannot return NO, since, if the user selects the No button, the Exit Setup dialog is displayed.
 - n You can also specify szLicenseFile by entering the fully qualified name, in quotation marks, or a Universal Naming Convention (UNC) path.
 - n The text in szLicenseFile should contain hard returns after lines with more than 512 characters (16-bit InstallShield) or 1,024 characters (32-bit InstallShield). The text from this file is read into string lists at 512- or 1,024-byte intervals. If the text from szLicenseFile does not contain hard returns, the words may wrap inside the SdLicense dialog box with unexpected results.
-

{button ,JI(`LANGREF.HLP>Examples','SdLicense_example')} Example

{button ,AL(`Tips for using Sd dialog boxes;Exit conditions;Caching dialogs;Displaying special characters and TM in dialogs and titles;Which fonts InstallShield dialog boxes use;Modifying Sd dialog box source files',0,','')} See also

{button ,PI("",`comment')} Feedback



SdLicense example

```
/*-----*\
 *
 * This example illustrates the usage of the SdLicense dialog box.
 *
 * SdLicense is called to display a license agreement stored in LICENSE.TXT.
 * It asks if the user accepts the agreement and wants to proceed.
 *
 * NOTE: In order for this script to run properly, it is advised to set the
 *       LICENSE_PATH constant to a valid text file on the target system.
 *
 \*-----*/

#define LICENSE_PATH "EXAMPLE\\LICENSE.TXT"

#include "Sddialog.h"

STRING szTitle, szMsg, szQuestion, szLicenseFile;

program

// Back button not required for initial dialog box.
Disable(BACKBUTTON);

// Set values for the variables passed to SdLicense.
szTitle      = "SdLicense Example";
szMsg        = "Please read the following license agreement. Use " +
               "the scroll bar to view\nthe rest of this agreement";
szQuestion   = "Do you accept all the terms of the preceding license " +
               "agreement? If so,\nclick on the Yes push button. " +
               "If you select No, Setup will close.";
szLicenseFile = LICENSE_PATH;

/*-----*\
 *
 * The following displays the SdLicense dialog box.
 *
 \*-----*/
if (SdLicense(szTitle, szMsg, szQuestion, szLicenseFile) = YES) then

    MessageBox("Continue with the installation.", INFORMATION);
endif;

endprogram

#include "Sddialog.rul"

// Source file: Is5fn144.rul
```

{button ,PI("",`comment')} [Feedback](#)



SdProductName

Syntax

```
SdProductName (szProductName);
```

Description

The SdProductName function makes your product name available to all instances of the %P place holder. The %P place holder is found in static text fields in some Sd dialog boxes. In addition, some Sd dialog box functions, such as [SdFinish](#), allow you to include %P in strings passed as parameters to functions.



When using SdProductName or any other Sd dialog box function, remember to add `#include "Sddialog.h"` before the program block and `#include "Sddialog.rul"` after the endprogram statement. If you are using the Setup.rul generated by the Project Wizard, these lines are already included for you.

Parameters

szProductName

Specifies the name of product that is being installed. This name will appear by default wherever %P appears in the appropriate static fields in Sd dialogs.

Return values

This function does not return a value.

```
{button ,JI('LANGREF.HLP>Examples',`SdProductName_example')}} Example
```

```
{button ,AL(`SdFinish;SdFinishReboot;SdLicense;SdWelcome;Tips for using Sd dialog boxes;Exit conditions;Caching dialogs;Displaying special characters and TM in dialogs and titles;Which fonts InstallShield dialog boxes use;Modifying Sd dialog box source files',0,','')}} See also
```

```
{button ,PI('','comment')}} Feedback
```



SdProductName example

```
/*-----*\
 *
 * This example illustrates the use of the functions SdProductName,
 * SdWelcome, SdSetupType, and SdFinish.
 *
 * First, SdProductName is called to establish the product name to
 * substitute for the %P place holder, which is embedded in the
 * message strings passed to SdWelcome, SdSetupType, and SdFinish.
 *
 * Next, SdWelcome is called to display a welcome message and then
 * SdSetupType is called. Finally, a call to SdFinish completes the
 * setup process by giving the user final options.
 *
 *-----*/

#include "Sddialog.h"

STRING  szProductName, szTitle, szMsg, svDir, szComponents;
STRING  szMsg1, szMsg2, szOpt1, szOpt2;
BOOL    bvOpt1, bvOpt2;
NUMBER  nReturn;

program

//Set the product name to substitute for the %P place holder.
szProductName = "My Application";
SdProductName(szProductName);

SdWelcomeLabel:
szTitle = "SdWelcome Example";
Disable(BACKBUTTON);

//Display the SdWelcome dialog box. The null string in parameter
//two specifies the default message, which uses the %P place holder.
SdWelcome(szTitle, "");

Enable(BACKBUTTON);

SdSetupTypeLabel:
szTitle = "SdProductName Example";

//The %P place holder is embedded in a custom message that will be
//passed to SdSetupType.
szMsg    = "Choose the type of installation for %P.";
svDir    = "C:\\EXAMPLE";

// Display the SdSetupType dialog box.
if (SdSetupType(szTitle, szMsg, svDir, 0) = BACK) then

    goto SdWelcomeLabel;
endif;

// The %P place holder is embedded in several of the string
// parameters that will be passed to SdFinish.
szTitle = "SdFinish Example";
szMsg1  = "%P Setup is almost complete.\nChoose the options you want below.";
szMsg2  = "Click Finish to complete %P Setup.";
szOpt1  = "I would like to view the README file.";
szOpt2  = "I would like to launch %P.";
```

```
Enable(BACKBUTTON);

// Display the SdFinish dialog box.
SdFinish(szTitle, szMsg1, szMsg2, szOpt1, szOpt2, bvOpt1, bvOpt2);

if(bvOpt1) then
    MessageBox("Display readme file here.", INFORMATION);
endif;

if(bvOpt2) then
    SprintfBox(INFORMATION, szTitle, "Launch %s here.", szProductName);
endif;

endprogram

#include "Sddialog.rul"

// Source file: Is5fn142.rul
```

{button ,PI("",`comment')} [Feedback](#)



SdRegisterUserEx

Syntax

SdRegisterUserEx (szTitle, szMsg, svName, svCompany, svSerial);

Description

The SdRegisterUserEx function creates a dialog box that retrieves the user name, company name, and serial number. If both svName and svCompany contain null strings, InstallShield will obtain the user name and company name from the registry in 32-bit setups or the User.exe file in 16-bit setups.

The Next button becomes enabled only when data exists in all three edit fields. You cannot leave any field blank.



[View sample dialog](#)



When using SdRegisterUserEx or any other Sd dialog box function, remember to add `#include "Sddialog.h"` before the program block and `#include "Sddialog.rul"` after the endprogram statement. If you are using the Setup.rul generated by the Project Wizard, these lines are already included for you.

Parameters

szTitle

Specifies the title of the dialog box. To display the default title ("User Information"), pass a null string ("") in this parameter.

szMsg

Specifies the message to display in the dialog box. This text is considered a static control. To display the default instructions for this dialog box, pass a null string ("") in this parameter.

svName

Returns the name entered by the user. Note that SdRegisterUserEx displays the initial value of this parameter for editing by the user. If both svName and svCompany are null strings, the user name found in the target system's registry in 32-bit setups or the User.exe file in 16-bit setups is displayed for editing.

svCompany

Returns the company name entered by the user. Note that SdRegisterUserEx displays the initial value of this parameter for editing by the user. If both svName and svCompany are null strings, the company name found in the target system's registry in 32-bit setups or the User.exe file in 16-bit setups is displayed for editing.

svSerial

Returns the serial number entered by the user. You can use this information and write it to a file or display it in a confirmation dialog.

Return values

NEXT

Indicates that the Next button was clicked.

BACK

Indicates that the Back button was clicked.

{button ,JI('LANGREF.HLP>Examples','SdRegisterUserEx_example')} [Example](#)

{button ,AL('SdConfirmRegistration;SdRegisterUser;Tips for using Sd dialog boxes;Exit conditions;Caching dialogs;Displaying special characters and TM in dialogs and titles;Which fonts InstallShield dialog boxes use;Modifying Sd dialog box source files',0,',')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



SdRegisterUserEx example

```
/*-----*\
 *
 * This example illustrates the use of the SdRegisterUserEx, SdSetupType, and
 * the SdStartCopy functions.
 *
 * SdRegisterUserEx and SdSetupType are called to collect installation
 * information from the user. It stores the information in a list and then
 * displays it using SdStartCopy.
 *
 *-----*/

#include "Sddialog.h"

STRING  szTitle, szMsg, svName, svCompany, svSerial, svDir, svSetupType;
LIST    listData;
NUMBER  nResult;

program

start:

// Create the list.
listData = ListCreate (STRINGLIST);
szMsg = "Please enter your name, company, and serial no.";

// Retrieve registration information and store in a list.
SdRegisterUserEx("Registration", szMsg, svName, svCompany, svSerial);

ListAddString(listData, "User Information: ", AFTER);
ListAddString(listData, "          " + svName, AFTER);
ListAddString(listData, "          " + svCompany, AFTER);
ListAddString(listData, "          " + svSerial, AFTER);
ListAddString(listData, "", AFTER);

SetupTypeLabel:

// Retrieve setup type and directory information.
szMsg = "Choose the type of installation by clicking one of the buttons.";
svDir = TARGETDIR;
nResult = SdSetupType("Select Setup Type", szMsg, svDir, 0);

switch (nResult)
case TYPICAL:
    svSetupType = "TYPICAL: Application will be installed " +
                  "with the most common options.";
case COMPACT:
    svSetupType = "COMPACT: Application will be installed " +
                  "with the minimum required options.";
case CUSTOM:
    svSetupType = "CUSTOM: You select the options that you " +
                  "want installed.";
case BACK:
    goto start;
default:
    MessageBox ("Invalid setup type selection!", SEVERE);
    abort;
endswitch;

// Store retrieved information in list.
```

```

ListAddString(listData, "Setup Type:", AFTER);
ListAddString(listData, "          " + svSetupType, AFTER);
ListAddString(listData, "", AFTER);
ListAddString(listData, "Destination Directory:", AFTER);
ListAddString(listData, "          " + svDir, AFTER);

// Set title and static text for SdStartCopy.
szTitle = "Check Setup Information";
szMsg    = "Setup has enough information to begin the file-transfer\n" +
           "operation.  If you want to review or change any of the\n" +
           "settings, click Back.  If you are satisfied with the\n" +
           "settings, click Next to begin copying files.";

// Call SdStartCopy to display user selections.
nResult = SdStartCopy (szTitle, szMsg, listData);

switch(nResult)

    case NEXT:

        MessageBox("SdStartCopy successful.", INFORMATION);
    case BACK:

        goto SetupTypeLabel;
    default:

        MessageBox("SdStartCopy failed.", SEVERE);
endswitch;

endprogram

#define SD_SINGLE_DIALOGS 1
#define SD_REGISTERUSEREX 1
#define SD_SETUPTYPE     1
#define SD_STARTCOPY     1

#include "Sddialog.rul"

// Source file: Is5fn148.rul

```

{button ,PI("",`comment')} [Feedback](#)



SdSelectFolder

Syntax

SdSelectFolder (szTitle, szMsg, svDefGroup);

Description

The SdSelectFolder function displays program folders for selection. SdSelectFolder allows you to offer a default selection. The user can also enter a new folder name. SdSelectFolder will return only the selected or entered folder name. It cannot create the folder.



[View sample dialog](#)



When using SdSelectFolder or any other Sd dialog box function, remember to add `#include "Sddialog.h"` before the program block and `#include "Sddialog.rul"` after the endprogram statement. If you are using the Setup.rul generated by the Project Wizard, these lines are already included for you.

Parameters

szTitle

Specifies the title of the dialog box. To display the default title ("Select Program Folder"), pass a null string ("") in this parameter.

szMsg

Specifies the message to display in the dialog box. This text is considered a static control. To display the default instructions for this dialog box, pass a null string ("") in this parameter.

svDefGroup

Returns the name of the selected folder.

Return values

NEXT

Indicates that the Next button was clicked.

BACK

Indicates that the Back button was clicked.

{button ,JI('LANGREF.HLP>Examples',`SdSelectFolder_example`)} [Example](#)

{button ,AL(`ComponentDialog;SelectDir;Tips for using Sd dialog boxes;Exit conditions;Caching dialogs;Displaying special characters and TM in dialogs and titles;Which fonts InstallShield dialog boxes use;Modifying Sd dialog box source files',0,',')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



SdSelectFolder example

```
/*-----*\
*
* This example illustrates the use of the SdSelectFolder function.
*
* SdSelectFolder is called, prompting the user to select a folder.
*
* NOTE: The DEF_FOLDER constant must be set to an existing folder on the
*       local system in order for this script to work properly.
*
*\-----*/

#define DEF_FOLDER "Startup"

#include "Sddialog.h"

STRING  szTitle, szMsg, svDefGroup;

program

    szTitle  = "SdSelectFolder Example";

    // Use default message.
    szMsg    = "";
    svDefGroup = DEF_FOLDER;
    SdSelectFolder (szTitle, szMsg, svDefGroup);

endprogram

#define SD_SINGLE_DIALOGS 1
#define SD_SELECTFOLDER  1

#include "Sddialog.rul"

// Source file: Is5fn149.rul
```

{button ,PI("",`comment')} [Feedback](#)



SdShowInfoList

Syntax

```
SdShowInfoList (szTitle, szMsg, listID);
```

Description

The SdShowInfoList function creates a dialog box that displays a list of scrollable messages.



[View sample dialog](#)



When using SdShowInfoList or any other Sd dialog box function, remember to add `#include "Sddialog.h"` before the program block and `#include "Sddialog.rul"` after the endprogram statement. If you are using the Setup.rul generated by the Project Wizard, these lines are already included for you.

Parameters

szTitle

Specifies the title of the dialog box. To display the default title ("Information"), pass a null string ("") in this parameter.

szMsg

Specifies the message to display above the information box. To display the word "Text" above the list, pass a null string ("") in this parameter.

listID

Specifies the list of messages to display in the dialog box. All messages that appear in the dialog box are read-only.

Return values

NEXT

Indicates that the Next button was clicked.

BACK

Indicates that the Back button was clicked.

Comments

The multi-line edit field should be read-only.

```
{button ,JI('LANGREF.HLP>Examples',`SdShowInfoList_example`)} Example
```

```
{button ,AL(`SdShowAnyDialog;SdShowDlgEdit1;SdShowDlgEdit2;SdShowDlgEdit3;SdShowMsg;Tips for using Sd dialog boxes;Exit conditions;Caching dialogs;Displaying special characters and TM in dialogs and titles;Which fonts InstallShield dialog boxes use;Modifying Sd dialog box source files',0,`,`)} See also
```

```
{button ,PI("",`comment`)} Feedback
```



SdShowInfoList example

```
/*-----*\
 *
 * The following example illustrates the usage of the SdShowInfoList function.
 *
 * The GetSystemInfo function is called to retrieve information about the
 * user's system. The ListAdd function is used to add the resultant strings
 * to the list. Finally, the SdShowInfoList function is called to display the
 * various results.
 *
 *-----*/

#include "Sddialog.h"

STRING  szTitle, szMsg, svReturn, szInfo;
NUMBER  nvReturn;
LIST    listInfo;

program

    listInfo = ListCreate(STRINGLIST);

    // Check if the system has a CD-ROM drive.
    GetSystemInfo (CDROM, nvReturn, svReturn);

    if (nvReturn = TRUE) then

        szInfo = "Your machine has a CD-ROM Drive.";
    else

        szInfo = "Your machine does not have a CD-ROM drive.";
    endif;

    // Add the proper string to the list.
    ListAddString(listInfo, szInfo, AFTER);

    // Check the time on the system.
    GetSystemInfo (TIME, nvReturn, svReturn);
    Sprintf(szInfo, "The time now is %s.", svReturn);

    // Add the result to the list
    ListAddString(listInfo, szInfo, AFTER);

    // Check how much base memory the system has.
    GetSystemInfo (BASEMEMORY, nvReturn, svReturn);
    Sprintf(szInfo, "Your machine has %ld k base memory", nvReturn);

    ListAddString(listInfo, szInfo, AFTER);

    szTitle  = "SdShowInfoList Example";
    szMsg     = "Following is some information related to your system:";

/*-----*\
 *
 * Call SdShowInfoList to display the information on the string list.
 *
 *-----*/

    SdShowInfoList (szTitle, szMsg, listInfo);

endprogram
```

```
#define SD_SINGLE_DIALOGS 1
#define SD_SHOWINFOLIST 1

#include "Sddialog.rul"

// Source file: Is5fn156.rul
```

```
{button ,PI("",`comment')} Feedback
```



SdStartCopy

Syntax

```
SdStartCopy (szTitle, szMsg, listData);
```

Description

The SdStartCopy function creates a multi-line edit field displaying the settings and selections made during the installation. The user can click the Back button on the dialog to return to previous dialogs in order to change settings as required. Call SdStartCopy after retrieving the selections from the user, but before beginning the file-transfer process.

Use a string list to collect the information obtained during the installation. You then pass the string list to SdStartCopy in the parameter listData. SdStartCopy will display the list and allow the user to verify that the information is correct before continuing with the file transfer process.



[View sample dialog](#)



When using SdStartCopy or any other Sd dialog box function, remember to add `#include "Sddialog.h"` before the program block and `#include "Sddialog.rul"` after the endprogram statement. If you are using the Setup.rul generated by the Project Wizard, these lines are already included for you.

Parameters

szTitle

Specifies the title of the dialog box. To display the default title ("Start Copying Files"), pass a null string ("") in this parameter.

szMsg

Specifies the message to display in the static text field above the multi-line edit field. To display the default instructions for this dialog box, pass a null string ("") in this parameter.

listData

Specifies a string list of information retrieved from the end user. SdStartCopy will automatically place each element into the multi-line edit field. If the list is empty, the multi-line edit field will be hidden and only the static text field will be visible.

Return values

BACK

Indicates that the user selected the Back button.

NEXT

Indicates that the user selected the Next button.

```
{button ,JI(' LANGREF.HLP>Examples',`SdStartCopy_example`)} Example
```

```
{button ,AL(` SelectDir;SelectFolder;SetupType;SdSetupType;SdSelectFolder;SdRegisterUser;SdRegisterUserEx;Sd AskDestPath;Tips for using Sd dialog boxes;Exit conditions;Caching dialogs;Displaying special characters and TM in dialogs and titles;Which fonts InstallShield dialog boxes use;Modifying Sd dialog box source files',0,','')} See also
```

```
{button ,PI('','',`comment`)} Feedback
```



SdStartCopy example

```
/*-----*\
 *
 * This example illustrates the use of the SdRegisterUserEx, SdSetupType, and
 * the SdStartCopy functions.
 *
 * SdRegisterUserEx and SdSetupType are called to collect installation
 * information from the user. It stores the information in a list and then
 * displays it using SdStartCopy.
 *
 *-----*/

#include "Sddialog.h"

STRING  szTitle, szMsg, svName, svCompany, svSerial, svDir, svSetupType;
LIST    listData;
NUMBER  nResult;

program

start:

// Create the list.
listData = ListCreate (STRINGLIST);
szMsg = "Please enter your name, company, and serial no.";

// Retrieve registration information and store in a list.
SdRegisterUserEx("Registration", szMsg, svName, svCompany, svSerial);

ListAddString(listData, "User Information: ", AFTER);
ListAddString(listData, "          " + svName, AFTER);
ListAddString(listData, "          " + svCompany, AFTER);
ListAddString(listData, "          " + svSerial, AFTER);
ListAddString(listData, "", AFTER);

SetupTypeLabel:

// Retrieve setup type and directory information.
szMsg = "Choose the type of installation by clicking one of the buttons.";
svDir = TARGETDIR;
nResult = SdSetupType("Select Setup Type", szMsg, svDir, 0);

switch (nResult)
case TYPICAL:
    svSetupType = "TYPICAL: Application will be installed " +
                  "with the most common options.";
case COMPACT:
    svSetupType = "COMPACT: Application will be installed " +
                  "with the minimum required options.";
case CUSTOM:
    svSetupType = "CUSTOM: You select the options that you " +
                  "want installed.";
case BACK:
    goto start;
default:
    MessageBox ("Invalid setup type selection!", SEVERE);
    abort;
endswitch;

// Store retrieved information in list.
```

```

ListAddString(listData, "Setup Type:", AFTER);
ListAddString(listData, "          " + svSetupType, AFTER);
ListAddString(listData, "", AFTER);
ListAddString(listData, "Destination Directory:", AFTER);
ListAddString(listData, "          " + svDir, AFTER);

// Set title and static text for SdStartCopy.
szTitle = "Check Setup Information";
szMsg   = "Setup has enough information to begin the file-transfer\n" +
          "operation.  If you want to review or change any of the\n" +
          "settings, click Back.  If you are satisfied with the\n" +
          "settings, click Next to begin copying files.";

// Call SdStartCopy to display user selections.
nResult = SdStartCopy (szTitle, szMsg, listData);

switch(nResult)

    case NEXT:

        MessageBox("SdStartCopy successful.", INFORMATION);
    case BACK:

        goto SetupTypeLabel;
    default:

        MessageBox("SdStartCopy failed.", SEVERE);
endswitch;

endprogram

#define SD_SINGLE_DIALOGS 1
#define SD_REGISTERUSEREX 1
#define SD_SETUPTYPE     1
#define SD_STARTCOPY     1

#include "Sddialog.rul"

// Source file: Is5fn148.rul

```

{button ,PI("",`comment')} [Feedback](#)



SdWelcome

Syntax

SdWelcome (szTitle, szMsg);

Description

The SdWelcome function displays a dialog box that welcomes the end user.



[View sample dialog](#)



When using SdWelcome or any other Sd dialog box function, remember to add `#include "Sddialog.h"` before the program block and `#include "Sddialog.rul"` after the endprogram statement. If you are using the Setup.rul generated by the Project Wizard, these lines are already included for you.

Parameters

szTitle

Specifies the text to display in the title of the dialog box. To display the default title ("Welcome"), pass a null string ("") in this parameter.

szMsg

Specifies the message to display in the Welcome dialog box. To include in this message the product name set by a previous call to [SdProductName](#), insert the place holder %P anywhere in the message string. When the message is displayed, %P is replaced by the product name. To display the default welcome message ("Welcome to the %P Setup program. This program will install %P on your computer"), pass a null string ("") in this parameter.

Return values

NEXT

Indicates that the Next button was clicked.

BACK

Indicates that the Back button was clicked.

{button ,JI('LANGREF.HLP>Examples','SdWelcome_example')} [Example](#)

{button ,AL('SdBitmap;Welcome;Tips for using Sd dialog boxes;Exit conditions;Caching dialogs;Displaying special characters and TM in dialogs and titles;Which fonts InstallShield dialog boxes use;Modifying Sd dialog box source files',0,','')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



SdWelcome example

```
/*-----*\
 *
 * This example illustrates the use of the functions SdProductName,
 * SdWelcome, SdSetupType, and SdFinish.
 *
 * First, SdProductName is called to establish the product name to
 * substitute for the %P place holder, which is embedded in the
 * message strings passed to SdWelcome, SdSetupType, and SdFinish.
 *
 * Next, SdWelcome is called to display a welcome message and then
 * SdSetupType is called. Finally, a call to SdFinish completes the
 * setup process by giving the user final options.
 *
 *-----*/

#include "Sddialog.h"

STRING  szProductName, szTitle, szMsg, svDir, szComponents;
STRING  szMsg1, szMsg2, szOpt1, szOpt2;
BOOL    bvOpt1, bvOpt2;
NUMBER  nReturn;

program

//Set the product name to substitute for the %P place holder.
szProductName = "My Application";
SdProductName(szProductName);

SdWelcomeLabel:
szTitle = "SdWelcome Example";
Disable(BACKBUTTON);

//Display the SdWelcome dialog box. The null string in parameter
//two specifies the default message, which uses the %P place holder.
SdWelcome(szTitle, "");

Enable(BACKBUTTON);

SdSetupTypeLabel:
szTitle = "SdProductName Example";

//The %P place holder is embedded in a custom message that will be
//passed to SdSetupType.
szMsg    = "Choose the type of installation for %P.";
svDir    = "C:\\EXAMPLE";

// Display the SdSetupType dialog box.
if (SdSetupType(szTitle, szMsg, svDir, 0) = BACK) then

    goto SdWelcomeLabel;
endif;

// The %P place holder is embedded in several of the string
// parameters that will be passed to SdFinish.
szTitle = "SdFinish Example";
szMsg1  = "%P Setup is almost complete.\nChoose the options you want below.";
szMsg2  = "Click Finish to complete %P Setup.";
szOpt1  = "I would like to view the README file.";
szOpt2  = "I would like to launch %P.";
```

```
Enable(BACKBUTTON);

// Display the SdFinish dialog box.
SdFinish(szTitle, szMsg1, szMsg2, szOpt1, szOpt2, bvOpt1, bvOpt2);

if(bvOpt1) then
    MessageBox("Display readme file here.", INFORMATION);
endif;

if(bvOpt2) then
    SprintfBox(INFORMATION, szTitle, "Launch %s here.", szProductName);
endif;

endprogram

#include "Sddialog.rul"

// Source file: Is5fn142.rul
```

{button ,PI("",`comment')} [Feedback](#)



Shared and locked file functions

A shared file is a file, such as a .dll, .vbx, or driver that can be used by more than one application. InstallShield protects shared files from being removed during uninstallation.

Functions using the SHAREDFILE option under Windows 95 and Windows NT consider all files to be shared files, and therefore increment registry reference counters for all files involved. InstallShield increments the registry reference counter by one if the file exists in the target directory and it has a reference count greater than 0. If the shared file does not exist in the target directory and it has no reference counter, InstallShield creates the counter and sets it to 1. If the shared file already exists in the target directory but has no reference counter, InstallShield creates the counter and initializes it to 2 as a precaution against accidental removal during uninstallation.

Under 16-bit Windows platforms, there is no registry reference counter, so no files updated or copied using the SHAREDFILE option are logged for uninstallation. You do not need to disable logging, which would interfere with performance.

Shared files should not be updated when they are locked. Some InstallShield file transfer functions use the SHAREDFILE option so that .dll and .exe files that are locked during file transfer can be recorded and updated when Windows or the system restarts.

InstallShield considers a file locked when it is in use by an application or the system. Locked files are not necessarily shared files.

The following functions handle shared or locked files:

CommitSharedFiles

Records locked .dll and .exe files, ensuring that the files are updated when Windows is restarted.

Is

Provides file and path checking services, searches for a math coprocessor, tests for administrator status under Windows NT, and determines whether Microsoft Windows is running from a shared copy on a network.

RebootDialog

Presents a dialog box with which the end user can choose to restart Windows or reboot the computer.

SdFinishReboot

Presents a dialog box stating that the installation is complete and allowing the end user to choose to restart Windows or reboot the computer.

VerUpdateFile

Updates files using version resource information. Allows updating of locked .dll and .exe files and incrementing of registry reference counters for all files involved.

XCopyFile

Copies files and subdirectories from the source directory to the target directory. Combines shared file and locked file handling by causing XCopyFile to treat all files as shared, and to record locked .dll and .exe files for update when Windows or the system restarts.

{button ,PI("",`comment')} [Feedback](#)



CommitSharedFiles

Syntax

CommitSharedFiles (IReserved);

Description

The CommitSharedFiles function ensures that when Windows is restarted, .dll and .exe files that were locked during file transfer will be updated based on date.

[VerUpdateFile](#) and [XCopyFile](#) have the SHAREDFILE and LOCKEDFILE options. (The SHAREDFILE option differs from the LOCKEDFILE option in that it increases the registry reference count for the transferred file(s).) When InstallShield encounters a locked file while transferring files with these options, it sets the system variable BATCH_INSTALL to TRUE. If at the end of a setup BATCH_INSTALL is found to be TRUE, then calling CommitSharedFiles ensures that properly recorded locked .dll and .exe files are updated (newer files replace older files) when Windows restarts.

If the [SdFinishReboot](#) function is used to restart Windows or the system, locked .dll and .exe files are committed automatically and there is no need to call CommitSharedFiles.

If the [RebootDialog](#) function is used to restart Windows or the system, locked .dll and .exe files are committed automatically and there is no need to call CommitSharedFiles, unless the user chooses to start Windows later. In that case, you must call CommitSharedFiles before restarting Windows or the system. Refer to the RebootDialog function for details.

CommitSharedFiles should be executed only once in a setup script. However, if you are launching multiple scripts using [DoInstall](#), then you can, in effect, call CommitSharedFiles more than once in a setup because each individual script can call CommitSharedFiles once.

Parameters

IReserved

Pass zero in this parameter. No other value is allowed.

Return values

0

Indicates that the function successfully wrote the locked file data.

< 0

Indicates that the function was unable to write out the locked file data.

Comments

Before calling any of the functions that use the SHAREDFILE or LOCKEDFILE option, and before calling CommitSharedFiles or SdFinishReboot, the application information must be set using InstallationInfo, and uninstallation information must be set using the [DeinstallStart](#) function.

{button ,JI('LANGREF.HLP>Examples','CommitSharedFiles_example')} [Example](#)

{button ,AL('DoInstall;InstallationInfo;RebootDialog;SdFinishReboot;VerSearchAndUpdateFile;VerUpdateFile',0,'')}
} [See also](#)

{button ,PI('','comment')} [Feedback](#)



CommitSharedFiles example

```
/*-----*\
*
* This example illustrates the usage of the CommitSharedFiles function.
*
* The BATCH_INSTALL variable is set to TRUE when InstallShield encounters a
* locked or shared file during a transfer. If the BATCH_INSTALL system
* variable is set to TRUE in this script, the RebootDialog dialog box is
* displayed. The user is provided with an option to restart the system. If
* the user chooses to restart the system immediately, the locked file will be
* updated automatically. Otherwise, CommitSharedFiles is called so that
* the files will be updated the next time the system is restarted.
*
* NOTE: In order to run CommitSharedFiles in this script, it will be
*       necessary to write in a shared file or locked file transfer.
*
*       Also note the application and uninstallation information must be set
*       into the registry by calling the InstallationInfo and DeinstallStart
*       functions. To remove these changes in the registry after this script
*       has run, either delete the alterations using the REGEDIT executable,
*       or simply run the uninstallation executable with the log file as a
*       command-line parameter.
*
*-----*\

#define COMPANY_NAME      "Example_Company"
#define PRODUCT_NAME     "Example_App"
#define PRODUCT_VERSION  "5.0"
#define PRODUCT_KEY      "EXAMPLE.EXE"
#define DEINSTALL_KEY    "Example_DeinstKey"
#define DEFAULT_LOG_PATH "EXAMPLE"

    STRING szMsg, svLogFile, szTitle;
    NUMBER nReturn;

program

    // Set up the application information in the registry.
    InstallationInfo(COMPANY_NAME, PRODUCT_NAME, PRODUCT_VERSION, PRODUCT_KEY);
    DeinstallStart(DEFAULT_LOG_PATH, svLogFile, DEINSTALL_KEY, 0);

/*-----*\
*
* Display and execute a file transfer here. The file transfer function,
* (XCopyFile, etc.) must contain the SHAREDFILE parameter.
* For example:
*
* XCopyFile(szSrcFile, szTargetFile, SHAREDFILE);
*
* For the CommitSharedFiles function to work, a certain file within the transfer
* must be locked or shared, and the user must select to restart the system at
* a later time.
*
*-----*\

// Display the value of BATCH_INSTALL (TRUE = 1, FALSE = 0).
SprintfBox(INFORMATION, "", "BATCH_INSTALL = %ld", BATCH_INSTALL);

// If BATCH_INSTALL is TRUE, display the RebootDialog dialog box.
if (BATCH_INSTALL = TRUE) then
```

```

    szMsg = "Some files could not be updated because they are currently " +
        "in use by other programs on the system. Files in use will " +
        "be updated the next time you restart your system.";

    // If RebootDialog reboots the system, the shared file data is
    // automatically recorded.
    if (RebootDialog("CommitSharedFiles Example", szMsg, 0) = 0) then

/*-----*\
*
* If the user wishes to reboot later, CommitSharedFiles must be called to
* record the shared file data.
*
*\-----*/

        if (CommitSharedFiles(0) < 0) then
            MessageBox("CommitSharedFiles failed.", SEVERE);
        else
            MessageBox("CommitSharedFiles successful.", INFORMATION);
        endif;
    endif;

    // If BATCH_INSTALL is FALSE, Setup is complete.
    else

        szMsg = "Setup is complete. You may run the installed program.";
        MessageBox(szMsg, INFORMATION);
    endif;

    szMsg = "The uninstallation log file is located at: %s";
    sprintfBox(INFORMATION, szTitle, szMsg, svLogFile);

endprogram

// Source file: Is5fn195.rul

```

{button ,PI("",`comment')} [Feedback](#)



Shell functions

Shell functions create new program folders, delete existing program folders, and add items to existing program folders. At the end of the setup, add the application to the appropriate program folder to allow the user to access your software immediately. The following functions also support various icon options.

AddFolderIcon

Adds an icon to a folder.

CreateProgramFolder

Creates a program folder.

CreateShellObjects

Creates the folders and shortcuts (or groups and icons) specified in the Resources pane's Shell Objects folder.

DeleteFolderIcon

Removes an icon or item from a program folder.

DeleteProgramFolder

Removes a program folder from the target system.

ExitProgMan

Closes a Program Manager shell that was launched during the setup.

GetFolderNameList

Retrieves all subfolder names and shortcuts in the specified folder.

GetGroupNameList

Retrieves a list of all program folder names.

GetItemNameList

Retrieves all program items names from a specified program group under the Program Manager shell.

ProgDefGroupType

Designates groups as either Personal or Common in a Windows NT environment.

QueryProgGroup

Returns information about a specified group.

QueryProgItem

Returns information about a specified program item or subfolder.

QueryShellMgr

Returns the name of the current shell manager.

ReloadProgGroup

Reloads (updates) a program group.

ReplaceFolderIcon

Replaces an icon in a specified folder.

SelectFolder

Presents a dialog box that allows the end user to select a folder from a list of program folders.

ShowGroup

Displays the specified program group.

ShowProgramFolder

Displays the specified program folder.

{button ,PI("",`comment')} [Feedback](#)



AddFolderIcon

Syntax

AddFolderIcon (szProgramFolder, szItemName, szCommandLine, szWorkingDir, szIconPath, nIcon, szShortCutKey, nFlag);

Description

The AddFolderIcon function inserts or replaces an icon in the program folder specified by szProgramFolder. If that program folder does not exist, AddFolderIcon creates it. szProgramFolder may specify a subfolder in a multi-level cascading menu. If the subfolder does not exist, AddFolderIcon will create the subfolder and, if necessary, its parent folders.

When adding icons to groups under Windows NT, call [ProgDefGroupType](#) first to establish the group as either COMMON or PERSONAL. By default, the folder icon is added under COMMON.

Parameters

szProgramFolder

Specifies the name of the folder to which to add the icon. To add a shortcut icon to the Windows 95 Start Programs menu, pass a null string ("") in this parameter.

You can enter a fully qualified path, such as:

```
"C:\\WINDOWS\\STARTMENU\\PROGRAMS\\ACCESSORIES\\GAMES"
```

Under Windows 95 and Windows NT 4.0 you can also use an InstallShield system variable:

FOLDER_DESKTOP

Adds the icon to the Desktop folder.

FOLDER_STARTMENU

Adds the icon to the Start Menu folder.

FOLDER_STARTUP

Adds the icon to the Startup menu folder.

FOLDER_PROGRAMS

Adds the icon to the Start Menu\\Programs folder.

Or you could use a relative path, such as:

```
FOLDER_STARTMENU ^ "ACCESSORIES\\GAMES"
```

If the specified folder does not exist, InstallShield creates it.



Folders created with AddFolderIcon in 16-bit setups are not logged for uninstallation. In 16-bit setups, call [CreateProgramFolder](#) to create the folder before calling AddFolderIcon. This method ensures that a newly-created folder is logged for uninstallation.

szItemName

Specifies the name of the icon to add to the folder. The name will appear beneath the icon. Under Windows 95, calling AddFolderIcon to add an icon to a program folder also creates a link file in the links directory specified by szCommandLine.

To use special characters, such as commas and parentheses, in the names of program item icons under Program Manager, enclose the string within an additional set of quotation marks, as in the following example:

```
"Add, Edit, or Delete (Line)""
```

On Windows 95 and Windows NT 4.0 platforms, no special formatting is required. However, be aware that the Explorer shell does not allow the following characters in item names: /, \, :, ?, <, >, or |.

szCommandLine

Specifies one of the following:

- n The fully qualified name of the executable associated with the icon, including any command line parameters. Refer to the Comments section below for more information. To add a shortcut icon to the Windows 95 Start Programs menu, enter the fully qualified path of the links directory, which is where your application stores its icon link files.
- n The fully qualified path if szItemName is a subfolder (Windows 95 and Windows NT 4.0 only).

szWorkingDir

Specifies the directory where the application's program files are located. (Not applicable if szItemName is a subfolder.) To make the directory that contains the program file the working directory, pass a null string ("") in this parameter. Refer to the Comments section below for more information.

szIconPath

Specifies the filename of the icon to display. (Not applicable if szItemName is a subfolder.) Refer to the Comments section below for more information.

nIcon

Specifies the icon ordinal in the Windows executable specified in szIconPath. (Not applicable if szItemName is a subfolder.) Icon ordinal numbers begin at 0, so to display the first icon in the executable file, enter 0; to display the second, enter 1, and so on. If you are not using a Windows icon, enter 0 in this parameter.

szShortcutKey

Specifies the shortcut key (in the form of a string) that allows the end user to start the application quickly. For example, if you wanted to be able to open the application by depressing the "Ctrl," the "Alt," and then the "1" key, enter "Ctrl + Alt + 1" in this parameter. (Not applicable if szItemName is a subfolder.)

nFlag

Specifies icon appearance. Pass one or more of the following predefined constants in this parameter. To pass two or more predefined constants in this parameter, combine those constants with the bitwise OR operator (|).

REPLACE

Indicates that the current icon or item in the folder is replaced.

RUN_MAXIMIZED

Indicates that the program should be maximized when launched. Note that the icon will not be created if you specify this constant in a setup running under the Program Manager shell. If your setup will call AddFolderIcon with this constant in the parameter nFlag, call [QueryShellMgr](#) first to determine the current shell; if Program Manager is the shell, assign one of the other constants to nFlag.

RUN_MINIMIZED

Indicates that the program should be minimized when launched.

RUN_SEPARATEMEMORY

Applies to 16-bit applications under Windows NT only. Checks the "Run in Separate Memory Space" check box in the icon properties sheet. This tells the operating system that the 16-bit application must run in its own memory space.

NULL

Performs none of the above.

Return values

0

Indicates that the function successfully added or replaced the icon in the specified folder and associated the executable with the icon.

< 0

Indicates that the function was unable to add or replace the icon and associate the executable with it.

Comments

- n If the path to your application executable contains long path names, you must enclose the fully qualified filename within single or double quotation marks. (If the filename has been assigned to a variable, pass that variable to [LongPathToQuote](#) to insert the quotation marks.) Note that command line parameters should *not* be surrounded with quotation marks. For that reason, it is advisable to build the szCommandLine string from

two separate strings.

Also note that when the program specified in `szCommandLine` is a 16-bit application, any path and/or filename that is passed to the program must be in its short form. Call [LongPathToShortPath](#) when necessary to convert long paths to short paths. The example below shows how to construct the parameter `szCommandLine` when specifying a 16-bit application.

```
// Path to application executable
szProgram = szMainDirectory ^ "Support\\Mviewer2.exe";

/ Command line parameters
szParam = szMainDirectory ^ "knowbase.mvb";

// Enclose the path to the application executable
// in quotation marks
LongPathToQuote (szProgram, TRUE);

// Necessary only because Mviewer2.exe is 16-bit
LongPathToShortPath (szParam);

szCommandLine = szProgram + " " + szParam; // Create the final command line.
```

- n Do not call `LongPathToQuote` for the expressions passed as the parameters `szWorkingDir` and `szIconPath`. `InstallShield` automatically encloses these paths in quotation marks.

{button ,JI('LANGREF.HLP>Examples','AddFolderIcon_example')}

[Examples](#)

{button ,AL('CreateProgramFolder;DeleteProgramFolder;DeleteFolderIcon;ProgDefGroupType;QueryShellMgr;SelectFolder;Launching unls under the PM shell;Prepare my setup for uninstallation',0,',';')}

[See also](#)

{button ,PI('','comment')} [Feedback](#)



AddFolderIcon examples

Place a shortcut to an executable file on the Start menu and the Start Programs Menu.

Create a cascading submenu on the StartUp menu and add an icon to the menu.

Place a subfolder on the Desktop and an icon pointing to an executable in the new folder.

{button ,PI("",`comment')} Feedback



AddFolderIcon example

```
/*-----*\
*
* AddFolderIcon Example 1
*
* This example places a shortcut to an executable file on the Start menu and
* the Start Programs Menu.
*
* NOTE: In order for this script to run properly, it is necessary for the
* preprocessor constants to be set to the Notepad executable and a
* valid text file on the target system.
*
* Also note this script will run properly only under Windows 95 or
* Windows NT 4.0. In order to run this example correctly under Program
* Manager, it is necessary to alter the szProgramFolder string variable
* to a valid group.
*-----*/

#define PROGRAM "C:\\WINDOWS\\NOTEPAD.EXE"
#define PARAM   "C:\\WINDOWS\\README.TXT"

    STRING szProgramFolder, szItemName, szCommandLine, szWorkingDir, szIconPath;
    STRING szShortCutKey, szProgram, szParam;
    NUMBER nIcon, nFlag;

program

    szProgramFolder = FOLDER_STARTMENU;
    szItemName      = "Notepad Example1";

    szProgram = PROGRAM;
    szParam   = PARAM;
    LongPathToQuote(szProgram, TRUE);
    LongPathToShortPath(szParam);

    szCommandLine = szProgram + " " + szParam;

    szWorkingDir = "";
    szIconPath   = "";
    nIcon        = 0;
    szShortCutKey = "";
    nFlag        = REPLACE;

/*-----*\
*
* The following adds a shortcut to the Start menu.
*-----*/

    if (AddFolderIcon(szProgramFolder, szItemName, szCommandLine, szWorkingDir,
        szIconPath, nIcon, szShortCutKey, nFlag) < 0) then
        MessageBox("AddFolderIcon failed.", SEVERE);
    else
        sprintfBox(INFORMATION, "AddFolderIcon", "%s created successfully.",
            szItemName);
    endif;

    szProgramFolder = "";
    szItemName      = "Notepad Example2";
```

```
/*-----*\
*
* The following adds a shortcut to the Programs menu.
*
\*-----*/

    if (AddFolderIcon (szProgramFolder, szItemName, szCommandLine, szWorkingDir,
                      szIconPath, nIcon, szShortCutKey, nFlag) < 0) then
        MessageBox("AddFolderIcon failed.", SEVERE);
    else
        sprintfBox(INFORMATION, "AddFolderIcon", "%s created successfully.",
                  szItemName);
    endif;

endprogram

// Source file: Is5fn188.rul
```

{button ,PI("",`comment')} [Feedback](#)


```
        szShortCutKey, nFlag);

if (nResult < 0) then
    MessageBox("AddFolderIcon failed.", SEVERE);
else
    sprintfBox(INFORMATION, "AddFolderIcon", "%s created successfully.",
              szItemName);
endif;

endprogram

// Source file: Is5fn588.rul
```

{button ,PI("",`comment')} Feedback



AddFolderIcon example

```
/*-----*\
*
* AddFolderIcon Example 3
*
* This example places a subfolder on the Desktop and an icon pointing to an
* executable in the new folder. The folder is a shortcut that points to an
* actual directory. From this folder the user can execute a shortcut which
* runs the program.
*
* NOTE: In order for this script to run properly, it is necessary for the
*       preprocessor constants to be set to a valid file name and path on the
*       target system.
*
*       Also note that this script will run properly only under Windows 95 or
*       Windows NT 4.0. In order to run this example correctly under the
*       Program Manager shell, it is necessary to alter the szProgramFolder
*       string variable to a valid group.
*
*-----*\

#define FOLDER      "C:\\WINDOWS\\"
#define PROGRAM     "C:\\WINDOWS\\NOTEPAD.EXE"
#define PARAM       "C:\\WINDOWS\\README.TXT"

    STRING  szProgramFolder, szItemName, szCommandLine, szWorkingDir;
    STRING  szIconPath, szShortCutKey;
    STRING  szProgram, szParam, szFolderDir;
    NUMBER  nIcon, nFlag, nResult;

program

    // szProgramFolder is the Desktop on the local system.
    szProgramFolder = FOLDER_DESKTOP;
    szItemName      = "Example folder";

    // Create the folder which the folder icon will point to.
    szFolderDir = FOLDER ^ szItemName;
    CreateDir(szFolderDir);

    // The command line for the folder icon must be the folder path, and
    // it must be enclosed in quotation marks if the path is longer than
    // eight characters.

    szCommandLine = szFolderDir;
    LongPathToQuote(szCommandLine, TRUE);

    szWorkingDir  = "";
    szIconPath    = "";
    nIcon         = 0;
    szShortCutKey = "";
    nFlag         = REPLACE|RUN_MINIMIZED;

/*-----*\
*
* Create the folder icon, and show the folder it points to.
*
*-----*\

nResult = AddFolderIcon (szProgramFolder, szItemName, szCommandLine,
                        szWorkingDir, szIconPath, nIcon, szShortCutKey,
```

```

        nFlag);

if (nResult < 0) then
    MessageBox("AddFolderIcon failed.", SEVERE);
else
    sprintfBox(INFORMATION, "AddFolderIcon", "%s created successfully.",
              szItemName);
endif;

// Display the folder just created.
ShowProgramFolder(szFolderDir, SW_SHOW);

/*-----*\
*
* AddFolderIcon now adds the Example icon to the newly created folder.  When
* this shortcut is clicked, the executable is minimized when run.
*
*\-----*/
szProgramFolder = szFolderDir;
szItemName      = "Notepad Example";

// Make sure the white space is not seen as a delimiter.  See comments.
szProgram      = PROGRAM;
LongPathToQuote(szProgram, TRUE);

szParam = PARAM;
LongPathToShortPath(szParam);

szCommandLine = szProgram + " " + szParam;
szWorkingDir  = "";
szIconPath    = "";

nResult = AddFolderIcon(szProgramFolder, szItemName, szCommandLine,
                      szWorkingDir, szIconPath, nIcon, szShortCutKey,
                      nFlag);

if (nResult < 0) then
    MessageBox("AddFolderIcon failed.", SEVERE);
else
    sprintfBox(INFORMATION, "AddFolderIcon", "%s created successfully.",
              szItemName);
endif;

endprogram

// Source File: Is5fn589.rul

```

{button ,PI("",`comment')} Feedback



CreateProgramFolder

Syntax

CreateProgramFolder (szFolderName);

Description

The CreateProgramFolder function creates a new folder on the target system. If the folder already exists, it is highlighted. Under Windows 95, the folder is created in the Start Programs menu. When creating program groups under Windows NT, call the [ProgDefGroupType](#) function first to establish the group as either COMMON or PERSONAL. COMMON is the default setting.

To use special characters such as commas and parentheses in the names of program item icons in Windows 3.1, enclose the name string, including its double quotation marks, inside additional single quotation marks, as in the following example:

```
"Add, Edit, or Delete (Line)"
```

Under Windows 95 and Windows NT 4.0 platforms no special formatting is required.

Parameters

szFolderName

Specifies the name of the folder to add to the target system.

Return values

0

Indicates that the function successfully added the folder to the target system.

< 0

Indicates that the function was unable to add the specified program folder.

{button ,JI('LANGREF.HLP>Examples','CreateProgramFolder_example')} [Example](#)

{button ,AL('AddFolderIcon;DeleteProgramFolder;ProgDefGroupType;SelectFolder;Prepare my setup for uninstallation',0,'')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



CreateProgramFolder example

```
/*-----*\
 *
 * This example illustrates the use of the CreateProgramFolder function.
 *
 * CreateProgramFolder creates a program folder named ExampleFolder on the
 * target system.
 *
 \*-----*/

    STRING  szFolderName, szTitle, szMsg;

program

    szFolderName = "ExampleFolder";
    szTitle      = "CreateProgramFolder";
    szMsg        = "%s created successfully.";

    if (CreateProgramFolder(szFolderName) < 0) then
        MessageBox("Failed To Create Program Folder!", SEVERE);
    else
        sprintfBox(INFORMATION, szTitle, szMsg, szFolderName);
    endif;

endprogram

// Source file: Is5fn201.rul
```

{button ,PI("",`comment')} Feedback



CreateShellObjects

Syntax

CreateShellObjects (szReserved);

Description

The CreateShellObjects function creates the shell objects (folders and shortcuts or groups and icons) that have been specified in the current media. Shell objects are defined in the Resources pane's Shell Objects folder. The name of the current media is stored in the system variable MEDIA.

Parameters

szReserved

Pass a null string ("") in this parameter. No other value is allowed.

Return values

0

Indicates that the function was successful.

< 0

Indicates that the function failed.

Comments

- n The value of the system variable MEDIA is set to 'DATA' during setup initialization. If you change the value of this variable to refer to a script-created component set, you must change the value back to 'DATA' before calling CreateShellObjects.
- n This function should be called only after ComponentMoveData has been called.

{button ,PI("", `comment`)} Feedback



DeleteFolderIcon

Syntax

DeleteFolderIcon (szProgramFolder, szItemName);

Description

The DeleteFolderIcon function removes a program icon from a folder.

Parameters

szProgramFolder

Specifies the name of the folder that contains the icon to remove.

szItemName

Specifies the name of the icon to delete.

Return values

0

Indicates that the function successfully deleted the specified icon.

< 0

Indicates that the function was unable to delete the icon.

{button ,JI('LANGREF.HLP>Examples',`DeleteFolderIcon_example`)} [Example](#)

{button ,AL(`AddFolderIcon;DeleteProgramFolder`,0,`,`)} [See also](#)

{button ,PI("",`comment`)} [Feedback](#)



DeleteFolderIcon example

```
/*-----*\
*
* This example illustrates the usage of the DeleteFolderIcon and
* DeleteProgramFolder functions.
*
* The following script deletes the 'Notepad Example' icon from the 'Example
* folder' folder. DeleteProgramFolder is then called again to delete this
* folder.
*
* NOTE: In order for this script to run properly, it is necessary for the
*       preprocessor constants to be set to a valid folder and icon on the
*       target system. To easily create this example folder and icon, run
*       the AddFolderIcon example #3.
*
\*-----*/

#define FOLDER 'C:\\Windows\\"Example folder"'
#define ICON   "Notepad Example"

    STRING szItemName, szProgramFolder;

program

    // Display the folder.
    szProgramFolder = FOLDER;
    ShowProgramFolder(szProgramFolder, SW_SHOW);
    Delay(3);

/*-----*\
*
* DeleteFolderIcon is called to delete the 'Notepad Example' icon.
*
\*-----*/

    szItemName = ICON;
    if (DeleteFolderIcon(szProgramFolder, szItemName) < 0) then

        MessageBox("DeleteFolderIcon failed.", SEVERE);

    endif;

    Delay(3);

/*-----*\
*
* DeleteProgramFolder is called to delete the 'Example folder' icon.
*
\*-----*/

    if (DeleteProgramFolder(szProgramFolder) < 0) then
        MessageBox("DeleteProgramFolder failed.", SEVERE);
    endif;

endprogram

// Source file: Is5fn206.rul
```

{button ,PI("",`comment')} Feedback



DeleteProgramFolder

Syntax

```
DeleteProgramFolder (szFolderName);
```

Description

The DeleteProgramFolder function deletes a program folder and its contents, including all folders that reside under the specified folder. It cannot delete the "Programs" folder.

Parameters

szFolderName

Specifies the name of the folder to remove.

Return values

0

Indicates that the function successfully removed the specified folder.

< 0

Indicates that the function was unable to remove the folder.

```
{button ,JI('LANGREF.HLP>Examples',`DeleteProgramFolder_example')}
```

[Example](#)

```
{button ,AL(`AddFolderIcon;DeleteFolderIcon;CreateProgramFolder',0,'`')}
```

[See also](#)

```
{button ,PI('','comment')} Feedback
```



DeleteProgramFolder example

```
/*-----*\
 *
 * This example illustrates the usage of the DeleteFolderIcon and
 * DeleteProgramFolder functions.
 *
 * The following script deletes the 'Notepad Example' icon from the 'Example
 * folder' folder. DeleteProgramFolder is then called again to delete this
 * folder.
 *
 * NOTE: In order for this script to run properly, it is necessary for the
 *       preprocessor constants to be set to a valid folder and icon on the
 *       target system. To easily create this example folder and icon, run
 *       the AddFolderIcon example #3.
 *
 *-----*/

#define FOLDER 'C:\\Windows\\"Example folder"'
#define ICON   "Notepad Example"

    STRING szItemName, szProgramFolder;

program

    // Display the folder.
    szProgramFolder = FOLDER;
    ShowProgramFolder(szProgramFolder, SW_SHOW);
    Delay(3);

/*-----*\
 *
 * DeleteFolderIcon is called to delete the 'Notepad Example' icon.
 *
 *-----*/

    szItemName = ICON;
    if (DeleteFolderIcon(szProgramFolder, szItemName) < 0) then

        MessageBox("DeleteFolderIcon failed.", SEVERE);

    endif;

    Delay(3);

/*-----*\
 *
 * DeleteProgramFolder is called to delete the 'Example folder' icon.
 *
 *-----*/

    if (DeleteProgramFolder(szProgramFolder) < 0) then
        MessageBox("DeleteProgramFolder failed.", SEVERE);
    endif;

endprogram

// Source file: Is5fn206.rul
```

{button ,PI("",`comment')} Feedback



ExitProgMan

Syntax

ExitProgMan (bSaveGroup);

Description

The ExitProgMan function, which is for use only on Windows 3.1 systems, exits a shell that was launched as a secondary shell during the setup. If the shell running on the target system is not 100% compatible with the Windows Program Manager, you must launch the Program Manager as a secondary shell using the [LaunchApp](#) function before you call any functions that create the program groups and program items. After you create the program groups and program items, call the ExitProgMan function to exit the secondary shell.

For example, if the target system is using Norton Desktop on Windows 3.1, you must launch Program Manager as a secondary shell in order to create program groups and items. The ExitProgMan function instructs the secondary shell to exit and optionally save its group information. The ExitProgMan function works only if the secondary shell was launched as an alternative shell during the setup. Call the [QueryShellMgr](#) function to determine the current shell.

The following restrictions apply to this function:

- n You cannot use this function to exit Windows. To exit Windows you must call the [System](#) function.
- n InstallShield's program group and program item functions work only with the Windows 3.1 Program Manager, or with 100% compatible shells.

Parameters

bSaveGroup

Specifies whether the Program Manager should save its group information when the program exits. Pass one of the following predefined constants in this parameter:

TRUE

The Program Manager saves its group information.

FALSE

The Program Manager will not save its group information.

Return values

0

ExitProgMan successfully closed the secondary Program Manager.

< 0

ExitProgMan was unable to close the Program Manager.

{button ,JI('LANGREF.HLP>Examples',`ExitProgMan_example')}

[Example](#)

{button ,AL(`LaunchApp;QueryShellMgr',0,`,`')}

[See also](#)

{button ,PI('`,`comment')} [Feedback](#)



ExitProgMan example

```
/*-----*\
 *
 * This example illustrates the usage of the ExitProgMan function.
 *
 * QueryShellMgr is called to test if Program Manager is the shell. If it is
 * not, PROGMAN.EXE is launched. ExitProgMan then closes the Program Manager.
 *
 * NOTE: ExitProgMan and therefore this script has no effect under the
 *       Windows95 shell.
 *
\*-----*/

    STRING svShellMgrName;

program

    QueryShellMgr(svShellMgrName);

    if (svShellMgrName != "PROGMAN.EXE") then

        // Launch the program manager.
        LaunchApp(WINDIR^"PROGMAN.EXE", "");
    endif;

    if (AskYesNo("Would you like to save the folder info?", YES) = YES) then

/*-----*\
 *
 * The following exits the Program manager shell, saving the folder info.
 *
\*-----*/

        ExitProgMan(TRUE);
    else

/*-----*\
 *
 * The following exits the Program manager shell without saving the folder
 * info.
 *
\*-----*/

        ExitProgMan(FALSE);
    endif;
endprogram

// Source file: Is5fn061.rul
```

{button ,PI("",`comment')} [Feedback](#)



GetFolderNameList

Syntax

GetFolderNameList (szFolderName, listItemsID, listSubFoldersID);

Description

The GetFolderNameList function is used to enumerate all program item shortcuts and subfolders in a specified folder. This function also can be used to enumerate program item shortcuts and subfolders in the root folder.



GetFolderNameList is supported only in 32-bit setups running under the Explorer Shell. The function will fail when used in a 16-bit setup running under the Program Manager Shell. In a 16-bit setup or in 32-bit setups running under Program Manager, call [GetItemNameList](#) and [GetGroupNameList](#) to retrieve this information.

When this function is used in a setup running under Windows NT 4.0, other restrictions apply. See the Comments section for more information.

Parameters

szFolderName

Specifies the name of the folder to be queried. You can enter a fully qualified path for szFolderName, such as: "C:\\Windows\\Start Menu\\Programs\\Accessories\\Games"

If szFolderName is null, GetFolderNameList searches the default Programs directory (see below). If you do not specify an [absolute path](#) for szFolderName, GetFolderNameList searches for a subfolder under the default Programs directory, according to the operating system:

- Windows 95: The Start Menu\\Programs folder located under the Windows folder. Note that this location will not change even if user profiles have been enabled.
- Windows NT 4.0 (if COMMON is selected with [ProgDefGroupType](#)): The ..\\profiles\\All Users\\Start Menu\\Programs folder located under the Windows folder.
- Windows NT 4.0 (if PERSONAL is selected with ProgDefGroupType): The . .\\profiles\\<user name>\\Start Menu\\Programs located under the Windows folder; <user name> is the user name of the current user.

You can also use an InstallShield system variable:

FOLDER_DESKTOP

Searches the Desktop folder.

FOLDER_STARTMENU

Searches the Start menu folder.

FOLDER_STARTUP

Searches the StartUp menu folder.

Or you could use a relative path, such as:

```
FOLDER_STARTMENU ^ "ACCESSORIES\\GAMES"
```

listItemsID

Returns a list with the names of the program item shortcuts in szFolderName. Note that on Windows NT, if szFolderName includes both personal and common program items, the list returned in this parameter will contain either the common or personal program item shortcuts, but not both.

listSubFoldersID

Returns a list with the names of the subfolders in szFolderName. Note that on Windows NT, if szFolderName includes both personal and common folders, the list returned in this parameter will contain either the common or personal folders, but not both.

Return values

0

GetFolderNameList successfully retrieved all the programs items and subfolder names.

< 0

GetFolderNameList was unable to retrieve the program items and subfolder names.

Comments

When this function is used in a setup running under Windows NT 4.0, the following restrictions apply:

- If you are enumerating a specific program folder (that is, you have specified a path to the folder), that folder must match the current folder type, common or personal. If the folder does not match the current folder type, it will not be located and the function will fail. To change the default folder type, call ProgDefGroupType.
- If you are enumerating a folder that supports personal and common items and folders, such as FOLDER_PROGRAMS, only the program item shortcuts and folders of the appropriate type will be returned by the function. To change the default folder type, call ProgDefGroupType.

{button ,JI('LANGREF.HLP>Examples','GetFolderNameList_example')} [Example](#)

{button ,AL('AddFolderIcon;GetItemNameList;GetGroupNameList;SdShowInfoList',0,'')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



GetFolderNameList example

```
/*-----*\
 *
 * This example illustrates the usage of the GetFolderNameList function.
 *
 * GetFolderNameList is called to list all the subfolders and program items
 * on the desktop.
 *
 * NOTE: This can work properly only under Windows 95 or NT 4.0. In order
 *       to make this script run correctly under Windows 3.1, be sure
 *       to set the szFolderName string to a valid folder path.
 *
 *-----*/

#include "Sddialog.h"

STRING  szFolderName;
NUMBER  nResult;
LIST    listItemsID, listFoldersID;

program

// Create lists for folders and program names.
listItemsID = ListCreate(STRINGLIST);

if (listItemsID = LIST_NULL) then

    MessageBox("Unable to create list.", SEVERE);
endif;

listFoldersID = ListCreate(STRINGLIST);

if (listFoldersID = LIST_NULL) then

    MessageBox("Unable to create subfolders list.", SEVERE);
endif;

// Place the folder and program names into the lists.
szFolderName = FOLDER_DESKTOP;
nResult = GetFolderNameList(szFolderName, listItemsID, listFoldersID);

if (nResult < 0) then

    MessageBox("GetFolderNameList failed.", SEVERE);
else

    // Display the lists.
    SdShowInfoList("", "Items list:", listItemsID);
    SdShowInfoList("", "Subfolders list:", listFoldersID);
endif;

endprogram

// Only include code for SdShowInfoList function
// to reduce compiled file size.
#define SD_SINGLE_DIALOGS 1
#define SD_SHOWINFOLIST 1

#include "Sddialog.rul"
```

// Source file: Is5fn079.rul

{button ,PI("",`comment')} Feedback



GetGroupNameList

Syntax

GetGroupNameList (listID);

Description

Under the Program Manager Shell, the GetGroupNameList function retrieves all program group names and places them in the list specified by listID. Under the Explorer Shell, the root level program folder names are retrieved and placed in the list specified by listID.

Note that on Windows NT 4.0 (32-bit setups only), only the appropriate program folder type will be returned (either personal or common), based on the current folder type, which is set with the function [ProgDefGroupType](#). On Windows NT 3.51 all program folder names will be returned regardless of the current folder type setting.

When this function is run on a system which is using a third party shell, the function may not return correct information. Consult the documentation for the shell in question to determine whether the shell is 100% Program Manager or Explorer shell compatible when the function returns incorrect information.

Parameters

listID

Returns a list of folder or group names; each name is an element in the list.

Return values

0

GetGroupNameList successfully retrieved the list of folder or group names.

< 0

GetGroupNameList was unable to retrieve the list of folder or group names.

{button ,JI('LANGREF.HLP>Examples','GetGroupNameList_example')} [Example](#)

{button ,AL('GetFolderNameList;GetItemNameList;ListCreate;ListDestroy;ListGetFirstString;ListGetNextString',0,'')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



GetGroupNameList example

```
/*-----*\
 *
 * This example illustrates the usage of the GetGroupNameList function.
 *
 * GetGroupNameList creates a list of all the folders in the Start Programs
 * menu of a Windows 95 system. This list is then displayed in a dialog box.
 *
 \*-----*/

#include "Sddialog.h";

STRING  szTitle, szMsg;
LIST    listID;

program

    // Create a string list.
    listID = ListCreate(STRINGLIST);

    if (listID = LIST_NULL) then

        MessageBox("Unable to create list.", SEVERE);
    endif;

/*-----*\
 *
 * Retrieve the folders in the Start menu and assign them to a list.
 *
 \*-----*/

    if (GetGroupNameList(listID) < 0) then

        MessageBox("GetGroupNameList failed.", SEVERE);
    endif;

    szTitle = "GetGroupNameList Example";
    szMsg    = "Contents of listID:";

    // Display the string list.
    SdShowInfoList(szTitle, szMsg, listID);

    // Remove the list from memory.
    ListDestroy(listID);

endprogram

#define SD_SINGLE_DIALOGS 1
#define SD_SHOWINFOLIST 1

#include "Sddialog.rul"

// Source file: Is5fn081.rul
```

{button ,PI("",`comment')} [Feedback](#)



GetItemNameList

Syntax

GetItemNameList (szGroup, listID);

Description

The GetItemNameList function is used to enumerate all program items (shortcuts) in a specified program group (folder). The function is supported in 16-bit setups on all platforms and in 32-bit setup running under the Program Manager shell (Windows NT 3.51). GetItemNameList will fail if used in a 32-bit setup running under the Explorer shell. In a 32-bit setup running under the Explorer shell, call [GetFolderNameList](#) instead to retrieve this information.

When this function is used under Windows NT 3.51, it will enumerate correctly a personal or common folder regardless of the currently defined folder type (common or personal).



This function requires that szGroup be a valid program group (folder); therefore this function cannot be used by a 16-bit setup to enumerate the root level program items that can be created under the Explorer shell by 32-bit setups. (The Program Manager Shell does not allow root level program items.) This is a limitation of a 16-bit setup running on a 32-bit platform.

Parameters

szGroup

Specifies the program group from which to retrieve program items. To use special characters such as commas and parentheses in the names of program item icons under the Program Manager shell, enclose the name string, including its double quotation marks, inside additional single quotation marks, as in the following example:

```
'"Add, Edit, or Delete (Line)'"'
```

Under the Explorer shell, no special formatting is required.

listID

Returns the name of each program item in the folder specified by szGroup.

Return values

0

GetItemNameList successfully retrieved all the program items.

< 0

GetItemNameList was unable to retrieve the program items. In 32-bit setups running under the Explorer shell, call [GetFolderNameList](#) instead to retrieve this information.

{button ,JI('LANGREF.HLP>Examples','GetItemNameList_example')} [Example](#)

{button ,AL('GetFolderNameList;GetGroupNameList;ListCreate;ListDestroy;ListGetFirstString;ListGetNextString',0,'',')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



GetItemNameList example

```
/*-----*\
 *
 * This example illustrates the usage of the GetItemNameList function.
 *
 * The GetItemNameList function retrieves a list of program items in the
 * user-defined program group. All the item names in this group then appear
 * to the user in a dialog box.
 *
 * NOTE: GetItemNameList may return an error if the target system has a shell
 *       other than the Program Manager.
 *
 \*-----*/

#include "Sddialog.h";

STRING szGroup, szTitle, szMsg, svResult;
LIST listID;

program

// Create a string list.
listID = ListCreate(STRINGLIST);

if (listID = LIST_NULL) then

    MessageBox("Unable to create list.", SEVERE);
    abort;
endif;

// Prompt user for a folder.
szTitle = "GetItemNameList Example";
SelectFolder(szTitle, "Startup", svResult);
szGroup = svResult;

/*-----*\
 *
 * Retrieve the program names in szGroup, and assign them to a string list.
 *
 \*-----*/
if (GetItemNameList(szGroup, listID) < 0) then

    MessageBox("GetItemNameList failed.", SEVERE);
else

    szMsg = "Contents of listID:";

    // Display the contents of the list.
    SdShowInfoList(szTitle, szMsg, listID);
endif;

// Remove the list from memory.
ListDestroy(listID);

endprogram

#define SD_SINGLE_DIALOGS 1
#define SD_SHOWINFOLIST 1
```

```
#include "Sddialog.rul"  
// Source file: Is5fn082.rul
```

```
{button ,PI("",`comment')} Feedback
```



ProgDefGroupType

Syntax

ProgDefGroupType (nType);

Description

The ProgDefGroupType function designates a program group as either Personal or Common under Windows NT. Call this function before you call [AddFolderIcon](#) or [CreateProgramFolder](#). The default program group type is Common.



This function is for use only in a Windows NT environment. It is ignored in other environments.

Parameters

nType

Specifies a program group type. Pass one of the following predefined constants in this parameter:

PERSONAL

Specifies a Personal program group.

COMMON

Specifies a Common program group.

Return values

0

This function always returns zero.

{button ,JI('LANGREF.HLP>Examples','ProgDefGroupType_example')} [Example](#)

{button ,AL('AddFolderIcon;CreateProgramFolder;GetFolderNameList',0,',')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



ProgDefGroupType example

```
/*-----*\
*
* This example illustrates the usage of the ProgDefGroupType function.
*
* NOTE: In order for this script to run properly, it is necessary for the
*       preprocessor constants to be set to the Notepad executable and a
*       valid text file on the target system.
*
*       Also note this script will run properly only under Windows 95 or
*       Windows NT 4.0. In order to run this example correctly under Program
*       Manager, it is necessary to alter the szProgramFolder string variable
*       to a valid group.
*
*-----*/

#define FOLDER "C:\\WINDOWS"
#define PROGRAM "C:\\WINDOWS\\NOTEPAD.EXE"
#define PARAM "C:\\WINDOWS\\README.TXT"

STRING szProgramFolder, szItemName, szCommandLine, szWorkingDir, szIconPath;
STRING szShortCutKey, szProgram, szParam, szIsData;
NUMBER nIcon, nFlag, nType;

program

    szProgramFolder = FOLDER;
    szItemName      = "Notepad Example1";

    szProgram = PROGRAM;
    szParam   = PARAM;
    LongPathToQuote(szProgram, TRUE);
    LongPathToShortPath(szParam);

    szCommandLine = szProgram + " " + szParam;

    szWorkingDir = "";
    szIconPath   = "";
    nIcon        = 0;
    szShortCutKey = "";
    nFlag        = REPLACE;

    if (Is(USER_ADMINISTRATOR, szIsData) == TRUE) then
        nType = COMMON;
    else
        nType = PERSONAL;
    endif;

    ProgDefGroupType(nType);

    // The following adds the shortcut.
    if (AddFolderIcon(szProgramFolder, szItemName, szCommandLine, szWorkingDir,
                    szIconPath, nIcon, szShortCutKey, nFlag) < 0) then
        MessageBox("AddFolderIcon failed.", SEVERE);
    else
        sprintfBox(INFORMATION, "AddFolderIcon", "%s created successfully.",
                  szItemName);
    endif;

    szProgramFolder = "";
```

```
szItemName    = "Notepad Example2";  
endprogram  
// Source file: Is5fn613.rul
```

```
{button ,PI("",`comment')} Feedback
```



QueryProgGroup

Syntax

QueryProgGroup (szGroupName, svGroupPath, nvItemCount);

Description

The QueryProgGroup function determines whether a specific program group exists; if it does, QueryProgGroup determines the number of program items contained in that program group. Under Windows 3.x, this function also obtains the path to the group (.GRP) file for the program group. In 32-bit setups under Windows NT, this function determines whether the program group is Common or Personal.



In 32-bit setups, this function works only under program manager shell platforms (Windows NT 3.51). On Explorer Shell platforms (Windows 95 or later and Windows NT 4.0 and later), the function will return -1. In 16-bit setups, this function will work only on Windows 3.x and Windows 95. Under Windows NT 4.0 or later or under Windows 98 or later, the function will return -1. Also, note that under Windows 95, this function can only query root level folders, that is, those folders that are subfolders of the Programs folder.

Parameters

szGroupName

Specifies the name of the program group to query. To use special characters such as commas and parentheses in the names of program item icons, enclose the name string, including its double quotation marks, inside additional single quotation marks, as in the following example:

```
"Add, Edit, or Delete (Line)"
```

svGroupPath

Under Windows 3.x, returns the complete path and filename of the group (.GRP) file. On other platforms, this parameter will not contain usable information.

nvItemCount

Under Windows 3.x, Returns the number of program items in the group. Under Windows NT 3.51, nvItemCount returns the number of program items and the group type. The lower 2 bytes contain the number of program items in the group. The upper 2 bytes contain the group type, where 0 = Personal and 1 = Common. In 16-bit setup running under Windows NT 3.51, the upper two bytes will always be 0 regardless of whether the folder is common or personal.

Return values

0

Indicates that the function successfully queried the program group.

< 0

Indicates that the function was unable to query the program group.

{button ,JI(`LANGREF.HLP>Examples`,`QueryProgGroup_example`)} [Example](#)

{button ,AL(`GetSystemInfo;HIWORD;LOWORD;QueryProgItem`,0,`,`)} [See also](#)

{button ,PI(``,`comment`)} [Feedback](#)



QueryProgGroup example

```
/*-----*\
 *
 * This example illustrates the use of the QueryProgGroup function to
 * query a specific program group.
 *
 *-----*/

    STRING  svResult, svGroupPath, szGroupName, szTitle, szMsg;
    NUMBER  nvResult, nvItemCount, nCount, nType;
    BOOL    bWinNT, bWin31, b16bit;

program

    // Determine whether the target system is running Windows NT.
    GetSystemInfo (OS, nvResult, svResult);

    if (nvResult = IS_WINDOWSNT) then
        bWinNT = TRUE;
    else
        bWinNT = FALSE;
    endif;

    if (nvResult = IS_WINDOWS) then
        bWin31 = TRUE;
    else
        bWin31 = FALSE;
    endif;

    GetSystemInfo (ISTYPE, nvResult, svResult);

    if (nvResult = 16) then
        b16bit=TRUE;
    else
        b16bit=FALSE;
    endif;

    szGroupName = "Accessories";
    szTitle      = "QueryProgGroup Example";

/*-----*\
 *
 * Query the 'Accessories' group.  nvItemCount is set to the number of items
 * in thegroup.  In 32-bit setups under Windows NT 3.51, the high word
 * contains the group type.  If the FOLDER group is of Common type, the
 * variable nType will be set to 1.  If the group is of Personal type, the
 * variable nType will be set to 0.
 *
 * svGroupPath contains the path to the group (.grp) file.  This name of the
 * group will be returned under Windows NT 3.51.
 *
 *-----*/

    if (QueryProgGroup (szGroupName, svGroupPath, nvItemCount) < 0) then
        MessageBox("QueryProgGroup failed.", SEVERE);
        abort;
    endif;

    if (bWinNT = TRUE && b16bit = FALSE) then
        nCount = LOWORD(nvItemCount);
        nType  = HIWORD(nvItemCount);
    endif;
endprogram
```

```
if (nType = 1) then
    szMsg = "" + szGroupName + "" + " Group is Common.\n\nNumber of " +
        "items = %ld";
else
    szMsg = "" + szGroupName + "" + " Group is Personal.\n\nNumber of " +
        "items = %ld";
endif;

// Display the results for 32-bit setups on Windows NT 3.51
sprintfBox(INFORMATION, szTitle, szMsg, nCount);

elseif (bWin31 = TRUE) then
    // Display the results for Windows 3.1
    szMsg = "The '%s' Group has %ld items.\n\nThe path of the group file is:\n%s.";
    sprintfBox (INFORMATION, szTitle, szMsg, szGroupName, nvItemCount,
        svGroupPath);

else
    // Display the results for all other Operating Systems & Setup Types.
    szMsg = "The '%s' Group has %ld items.";
    sprintfBox (INFORMATION, szTitle, szMsg, szGroupName, nvItemCount);
endif;

endprogram

// Source file: Is5fn124.rul
```

{button ,PI("",`comment')} [Feedback](#)



QueryProgItem

Syntax

```
QueryProgItem (szFolderName, szItemName, svCmdLine, svWrkDir, svIconPath, nvIconIndex, svShortCutKey, nvMinimizeFlag);
```

Description

The QueryProgItem function checks for the existence of a specific program item or subfolder name. If InstallShield finds the item or subfolder, QueryProgItem returns its attributes. The attributes include the application's command line, working directory, icon path, shortcut key, and minimize flag.

QueryProgItem checks for the existence of subfolder names successfully only under Windows NT version 4.0 and Windows 95.

To use special characters such as commas and parentheses in the names of program item icons in Windows 3.1, enclose the name string, including its double quotation marks, inside additional single quotation marks, as in the following example:

```
'"Add, Edit, or Delete (Line)'"'
```

On Windows 95 and Windows NT 4.0 platforms no special formatting is required.

To use QueryProgItem, enter information in the parameters szFolderName and szItemName. InstallShield fills the remaining parameters with the program item's attributes.

When querying groups or folders under Windows NT, first call the [ProgDefGroupType](#) function to establish the group as either COMMON or PERSONAL. By default, InstallShield searches under COMMON.

Parameters

szFolderName

Specifies the name of the folder that contains the item or subfolder.

You can enter a fully qualified path for szFolderName, such as

```
"C:\\WINDOWS\\START MENU\\PROGRAMS\\ACCESSORIES\\GAMES"
```

If szFolderName is null, QueryProgItem searches the default Programs directory (see below). If you do not specify an [absolute path](#) for szFolderName, QueryProgItem searches for a subfolder under the default Programs directory, according to the operating system:

- n Windows 95: The . \\Start Menu\\Programs directory under the Windir environment variable.
- n Windows NT 4.0 (if COMMON is selected with ProgDefGroupType): The . \\profiles\\All Users\\Start Menu\\Programs directory under the Windir environment variable.
- n Windows NT 4.0 (if PERSONAL is selected with ProgDefGroupType): The . \\profiles\\<user name>\\Start Menu\\Programs directory under the Windir environment variable. <user name> is the name of the current user from the USERPROFILE environment variable.
- n Windows 3.1: Not applicable.
- n Under Windows 95 and Windows NT 4.0 you can also use an InstallShield system variable:

FOLDER_DESKTOP

Queries items in the Desktop folder.

FOLDER_STARTMENU

Queries items in the Start menu folder.

FOLDER_STARTUP

Queries items in the Startup menu folder.

Or you could use a relative path, such as:

```
FOLDER_STARTMENU ^ "ACCESSORIES\\GAMES"
```

szItemName

Specifies the name of the program item or subfolder to find.

svCmdLine

Returns either the command line of the item's executable file or the complete path of the subfolder (under Windows 95 and Windows NT 4.0).

svWrkDir

Returns the full path of the working directory of the program item. (Not applicable if szItemName is a subfolder.)

svIconPath

Returns the fully qualified filename of the .ico file or .exe file. (Not applicable if szItemName is a subfolder.)

nvIconIndex

Returns the index of the icon used for the program item. (Not applicable if szItemName is a subfolder.)

svShortCutKey

Returns the item's shortcut key. (Not applicable if szItemName is a subfolder.)

nvMinimizeFlag

(Not applicable if szItemName is a subfolder.) Returns one of the following constants indicating whether an application window is minimized when first displayed. You can combine constants with the bitwise OR operator (|). These constants are available:

NULL

Indicates that the application's window is not minimized upon startup.

RUN_MINIMIZED

Indicates that the application's window is minimized upon startup.

RUN_SEPARATEMEMORY

Applies to 16-bit applications on Windows NT 4.0 only. Checks the "Run in Separate Memory Space" check box in the icon properties sheet. This tells the operating system that the 16-bit application must run in its own memory space.

Return values

IS_ITEM

Indicates szItemName is a program item or shortcut in szFolderName.

IS_FOLDER

Indicates szItemName is a subfolder in szFolderName.

< 0

Indicates that the function was unable to find the program item or subfolder name.

Comments

The location of the Start menu is different under different languages. Nonetheless, InstallShield automatically selects the correct path.

{button ,JI('LANGREF.HLP>Examples','QueryProgItem_example')} [Example](#)

{button ,AL('QueryProgGroup',0,'')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



QueryProgItem example

```
/*-----*\
*
* This example illustrates the usage of the QueryProgItem function.
*
* QueryProgItem is called to find the attributes of a target file or folder.
*
* NOTE: This script runs properly only under Win95 and WinNT 4.0. In order
*       for this example to run under Win 3.1, it is necessary to change
*       the FOLDER_NAME and ITEM_NAME constants to a valid folder and
*       item.
*
*-----*/

// Define a constant for the file or folder name.
#define FOLDER_NAME "C:\\WINDOWS\\START MENU\\PROGRAMS\\INSTALLSHIELD"
#define ITEM_NAME   "ONLINE REFERENCE"

#include "Sddialog.h"

STRING  szFolderName, szItemName, svCmdLine, svWrkDir, svIconPath;
STRING  svShortCutKey, svGroupPath, szTitle, szMsg, szInfo, svMinFlag;
STRING  svMinimizeFlag;
NUMBER  nvIconIndex, nvMinimizeFlag, nResult, nvMinFlag;
LIST    listInfo, listID;

program

    // Set variables for QueryProgItem.
    szFolderName = FOLDER_NAME;
    szItemName   = ITEM_NAME;

/*-----*\
*
* The following searches for an item in the FOLDER_NAME folder. It returns
* all attributes of the item in six different variables.
*
*-----*/
nResult = QueryProgItem (szFolderName, szItemName, svCmdLine, svWrkDir,
                        svIconPath, nvIconIndex, svShortCutKey,
                        nvMinimizeFlag);

// Create string list.
listInfo = ListCreate(STRINGLIST);

// Error check QueryProgItem.
if (nResult < 0) then

    MessageBox("QueryProgItem failed.", SEVERE);
    abort;

// Check if the item is an application.
elseif (nResult = IS_ITEM) then

    // Return the command line into the string list.
    Sprintf(szInfo, "The command line of %s: %s", szItemName, svCmdLine);
    ListAddString(listInfo, szInfo, AFTER);

    // Return the working directory into string list.
    Sprintf(szInfo, "The working directory of %s: %s", szItemName, svWrkDir);
```

```

ListAddString(listInfo, szInfo, AFTER);

// Return the icon path into string list.
Printf(szInfo, "The icon path of %s: %s", szItemName, svIconPath);
ListAddString(listInfo, szInfo, AFTER);

// Return icon index into string list.
Printf(szInfo, "The index of the icon: %d", nvIconIndex);
ListAddString(listInfo, szInfo, AFTER);

// Return shortcut key into string list.
Printf(szInfo, "The shortcut key of %s: %s", szItemName,
        svShortCutKey);
ListAddString(listInfo, szInfo, AFTER);

// Check if the item is a folder.
elseif (nResult = IS_FOLDER) then

    // Return a message into string list.
    Printf(szInfo, "The item is a subfolder. QueryProgItem does not " +
        "retrieve very much information about subfolders.");
    ListAddString(listInfo, szInfo, AFTER);
endif;

// Display the string list.
szTitle = "QueryProgItem Example";
szMsg    = "The following are attributes of the item:";
SdShowInfoList(szTitle, szMsg, listInfo);

endprogram

#define SD_SINGLE_DIALOGS 1
#define SD_SHOWINFOLIST 1

#include "Sddialog.rul"

// Source file: Is5fn125.rul

```

{button ,PI("",`comment')} [Feedback](#)



QueryShellMgr

Syntax

QueryShellMgr (svShellMgrName);

Description

The QueryShellMgr obtains the name of the program shell being used by Microsoft Windows. For example, if the program shell is Program Manager, QueryShellMgr returns the string "Progman.exe" in svShellMgrName.

If the Windows 95 shell or Program Manager is not the shell on the target system, you may need to launch one of them with the [LaunchApp](#) function. The InstallShield functions that create program folders and program icons use a DDE conversation with the shell to create the program folders and program items. Most alternate shells such as the Norton Desktop emulate the Windows 95 shell or Program Manager. Therefore, they can create program folders and items.

In shells that do not emulate the Windows 95 shell or Program Manager, InstallShield cannot use the program folder and program item functions to create or modify the program folders and program items. Check with the manufacturer of the shell to determine how it handles the creation of program folders and program items using Microsoft DDE specifications.

Parameters

svShellMgrName

Returns the unqualified name (that is, without the drive designation or path) of the shell manager that is currently running.

Return values

0

Indicates that the function successfully retrieved the name of the program shell.

< 0

Indicates that the function was unable to retrieve the name of the program shell.

{button ,JI('LANGREF.HLP>Examples','QueryShellMgr_example')} [Example](#)

{button ,AL('ExitProgMan',0,'')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



QueryShellMgr example

```
/*-----*\
 *
 * This example illustrates the usage of the QueryShellMgr function.
 *
 * QueryShellMgr is called to find the name of the shell manager and then
 * display it in a message box.  If the name is not retrievable, then an error
 * message is displayed.
 *
 \*-----*/

    STRING svShellMgrName, szTitle, szMsg;
    NUMBER nReturn;

program

/*-----*\
 *
 * Retrieve the name of the program shell.
 *
 \*-----*/
    nReturn = QueryShellMgr(svShellMgrName);

    // If an error occurs, display a message, otherwise display the results.
    if (nReturn < 0) then

        MessageBox("Could not retrieve the program shell.", SEVERE);
        abort;
    else

        szTitle = "QueryShellMgr Example";
        szMsg = "The name of the shell manager: %s";
        sprintfBox(INFORMATION, szTitle, szMsg, svShellMgrName);
    endif;

endprogram

// Source file: Is5fn126.rul
```

{button ,PI("",`comment')} [Feedback](#)



ReloadProgGroup

Syntax

```
ReloadProgGroup (szGroupName);
```

Description

The ReloadProgGroup function instructs the shell to remove and reload a specific folder. If you make modifications to program folder files directly and wish to immediately view the changes, call this function to force the shell to update the folder.

This function is for advanced users who want to directly modify folder files.

Parameters

szGroupName

Specifies the name of the program folder you want to reload. To use special characters such as commas and parentheses in the names of program item icons in Windows NT or Windows 3.1, enclose the name string, including its double quotation marks, inside additional single quotation marks, as in the following example:

```
'"Add, Edit, or Delete (Line)"'
```

On Windows 95 platforms no special formatting is required.

Return values

0

Indicates that the function successfully instructed the Program Manager to remove and reload a specified folder.

< 0

Indicates that the function was unable to instruct the Program Manager to remove and reload a specified folder.

{button ,JI('LANGREF.HLP>Examples','ReloadProgGroup_example')} [Example](#)

{button ,AL('ExitProgMan;QueryProgGroup',0,'')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



ReloadProgGroup example

```
/*-----*\
 *
 * This example illustrates the usage of the ReloadProgGroup function.
 *
 \*-----*/

    STRING szGroup;
    NUMBER nvFileHandle;

program

    szGroup = "MAIN.GRP";

    // Set the file mode to read/write mode.
    OpenFileMode(FILE_MODE_BINARY);

    // Open the group file.
    OpenFile(nvFileHandle, WINDIR, szGroup);

    // Make changes to the folder file here.

    // Save the folder information.
    CloseFile(nvFileHandle);

/*-----*\
 *
 * The following forces the shell to update the altered program group.
 *
 \*-----*/
    if (ReloadProgGroup(szGroup) < 0) then

        MessageBox("ReloadProgGroup failed.", SEVERE);
    endif;

endprogram

// Source file: Is5fn129.rul
```

{button ,PI("",`comment')} [Feedback](#)



ReplaceFolderIcon

Syntax

```
ReplaceFolderIcon (szProgramFolder, szItemName, szNewItem, szCmdLine, szWorkingDir, szIconPath, nIcon, szShortCutKey, nFlag);
```

Description

The ReplaceFolderIcon function replaces an icon in a specified folder. You must specify an existing folder, either one you have created with the [CreateProgramFolder](#) function or one that already exists on the user's system.

Parameters

szProgramFolder

Specifies the name of the folder that contains the icon to replace.

szItemName

Specifies the name of the icon to replace.

szNewItem

Specifies the name of the icon as it should appear after the replacement.

szCmdLine

Specifies the fully qualified filename of the existing icon.

szWorkingDir

Specifies the fully qualified filename of the replacement icon.

szIconPath

Specifies the name of an alternate icon file or a valid Windows executable that contains the new icon.

nIcon

Specifies the icon ordinal if you specified a Window executable icon. Otherwise, pass 0 in this parameter.

szShortCutKey

Specifies the string that contains the shortcut key sequence the user can press to start the program. For example, if you wanted the user to be able to open the application by depressing the "Ctrl," the "Alt," and then the "1" key, enter "Ctrl + Alt + 1" in this parameter.

nFlag

Specifies one or more options. Pass the following predefined constants in this parameter. To specify more than one option, combine constants with the OR (|) operator

NULL

Causes the function to perform none of these options.

REPLACE

Causes the existing icon to be replaced with the new icon.

RUN_MAXIMIZED

Causes the program window to be maximized when the user first starts the program.

RUN_MINIMIZED

Causes the program window to be minimized when the user first starts the application.

RUN_SEPARATEMEMORY

Applies to 16-bit applications on Windows NT 4.0 only. Checks the "Run in Separate Memory Space" check box in the icon properties sheet. This tells the operating system that the 16-bit application must run in its own memory space.

Return values

0

Indicates that the function successfully replaced the icon.

< 0

Indicates that the function was unable to replace the icon.

{button ,JI(`LANGREF.HLP>Examples`,`ReplaceFolderIcon_example`)} [Example](#)

{button ,AL(`AddFolderIcon;CreateProgramFolder;DeleteFolderIcon`,0,`,`)} [See also](#)

{button ,PI("",`comment`)} [Feedback](#)



ReplaceFolderIcon example

```
/*-----*\
*
* This example illustrates the ReplaceFolderIcon function.
*
* NOTE: In order for this script to run properly, it is necessary for the
*       preprocessor constants to be set to valid file names and a path on the
*       target system. To easily create this example folder and icon, run
*       the AddFolderIcon example #3.
*
*-----*/

#define FOLDER      "C:\\WINDOWS\\"
#define NEW_PROGRAM "C:\\WINDOWS\\WRITE.EXE"
#define NEW_PARAM   "C:\\WINDOWS\\README.TXT"

    STRING szProgramFolder, szItemName, szNewItem, szCmdLine, szWorkingDir;
    STRING szShortCutKey, szIconPath, szProgram, szParam;
    NUMBER nIcon, nFlag;

program

    szProgramFolder = FOLDER ^ "Example folder";
    szItemName      = "Notepad Example";
    szNewItem       = "New Wordpad Example";

    // Make sure the space is not seen as a delimiter, see the comments
    // section under AddFolderIcon.
    szProgram       = NEW_PROGRAM;
    LongPathToQuote(szProgram, TRUE);

    szParam         = NEW_PARAM;
    LongPathToShortPath(szParam);

    szCmdLine       = szProgram + " " + szParam;

    szWorkingDir    = "";
    szIconPath      = "";
    nIcon           = 0;
    szShortCutKey   = "";
    nFlag           = REPLACE|RUN_MAXIMIZED;

    // Display the folder on the screen.
    ShowProgramFolder(szProgramFolder, SW_SHOW);

/*-----*\
*
* ReplaceFolderIcon is called to replace the "Notepad Example" icon with
* "New Wordpad Example".
*
*-----*/

    if (ReplaceFolderIcon(szProgramFolder, szItemName, szNewItem, szCmdLine,
        szWorkingDir, szIconPath, nIcon, szShortCutKey,
        nFlag) < 0) then
        MessageBox("ReplaceFolderIcon failed.", SEVERE);
    else
        MessageBox("Icon successfully replaced.", INFORMATION);
    endif;
```

endprogram

// Source file: Is5fn250.rul

{button ,PI("",`comment')} Feedback



SelectFolder

Syntax

SelectFolder (szTitle, szDefFolder, svResultFolder);

Description

The SelectFolder function displays a dialog box that allows the end user to enter the name of a program folder in an edit field or select a program folder from a list. The function automatically displays all program folders on the system. A default folder name passed in svDefFolder is displayed in the edit field. The selected folder name is returned in svResultFolder. If the specified folder does not exist, it is not created.



[View sample dialog](#)

Parameters

szTitle

Specifies the title of the dialog box. To display the default title ("Select Program Folder"), pass a null string ("") in this parameter.

szDefFolder

Specifies the name of the folder to display as the default folder.

svResultFolder

Returns the name of the folder selected or specified by the end user. If the folder does not exist, you must call CreateProgramFolder to create it; SelectFolder will *not* create the folder.

Return values

BACK

Indicates that the end user selected the Back button.

NEXT

Indicates that the end user selected the Next button.

< 0

Indicates that the function was unsuccessful.

```
{button ,JI('LANGREF.HLP>Examples',`SelectFolder_example')}
```

[Example](#)

```
{button ,AL(`CreateProgramFolder;DeleteProgramFolder`,0,`,`)}
```

[See also](#)

```
{button ,PI("",`comment`)} Feedback
```



SelectFolder example

```
/*-----*\
 *
 * The following example uses the SelectFolder function to allow the user to
 * select a folder. It then displays the name of the selected folder.
 *
 \*-----*/

    STRING szTitle, szDefFolder, svResultFolder;
    NUMBER nReturn;

program

/*-----*\
 *
 * The following displays the SelectFolder dialog box. Prompting the user to
 * choose a folder on the target system.
 *
 \*-----*/
    szTitle = "SelectFolder Example";
    szDefFolder = "Main";

    nReturn = SelectFolder(szTitle, szDefFolder, svResultFolder);

    if (nReturn < 0) then
        MessageBox("SelectFolder failed.", SEVERE);
    else
        sprintfBox(INFORMATION, szTitle, "Selected folder: %s", svResultFolder);
    endif;

endprogram

// Source file: Is5fn251.rul
```

{button ,PI("",`comment')} [Feedback](#)



ShowGroup

Syntax

ShowGroup (szGrpName, nCommand);

Description

The ShowGroup function displays or moves a Program Manager folder window. Use this function at the end of the setup to make the new program folder and its program icons visible.

Parameters

szGrpName

Specifies the name of the program folder to display.

nCommand

Specifies how the folder window is to be displayed. Pass one of the following predefined constants in this parameter:

SW_NORMAL

Activates and displays the folder window. It returns to its original position if the window was moved.

SW_MAXIMIZE

Activates and maximizes the folder window.

SW_MINIMIZE

Minimizes the folder window.

SW_SHOW

Activates and displays the folder window in the window's current position.

SW_SHOWMINIMIZED

Activates and minimizes the folder window.

SW_SHOWMINNOACTIVE

Minimizes the folder window. The current active window remains active.

SW_SHOWNA

Displays the folder window in its current state. The current folder window remains active.

SW_SHOWNOACTIVATE

Displays the folder in its most recent position.

Return values

0

Indicates that the function successfully displayed or moved the folder window.

< 0

Indicates that the function was unable to display or move the folder window.

{button ,JI(`LANGREF.HLP>Examples',`ShowGroup_example')}} [Example](#)

{button ,AL(`QueryProgGroup;ReloadProgGroup;ShowGroup',0,`,`')}} [See also](#)

{button ,PI(`",`comment')}} [Feedback](#)



ShowGroup example

```
/*-----*\
 *
 * This example illustrates the usage of the ShowGroup function.
 *
 \*-----*/

    STRING szGrpName;
    NUMBER nCommand;

program

    // This is the name of the folder that is to be displayed.
    szGrpName = "Startup";
    nCommand  = SW_SHOW;

/*-----*\
 *
 * ShowGroup is called to activate and display the folder in its current
 * position.
 *
 \*-----*/

    ShowGroup(szGrpName, nCommand);

endprogram

// Source file: Is5fn167.rul
```

{button ,PI("",`comment')} [Feedback](#)



ShowProgramFolder

Syntax

ShowProgramFolder (szFolder, nCommand);

Description

The ShowProgramFolder function displays a program folder.

Parameters

szFolder

Specifies the name of the folder to display.

nCommand

Specifies the state of the folder. Pass one of the following predefined constants in this parameter:

SW_SHOW

Show folder in normal state.

SW_MAXIMIZE

Maximize the folder.

SW_MINIMIZE

Minimize the folder.

SW_RESTORE

Restore the folder to original size.

Return values

This function does not return a value.

{button ,JI(`LANGREF.HLP>Examples`,`ShowProgramFolder_example`)} [Example](#)

{button ,AL(`CreateProgramFolder;DeleteProgramFolder`,0,`,`)} [See also](#)

{button ,PI(``,`comment`)} [Feedback](#)



ShowProgramFolder example

```
/*-----*\
 *
 * This example illustrates the ShowProgramFolder function.
 *
 * ShowProgramFolder displays a folder, then changes the state of the folder.
 *
 * NOTE: This script will not run properly if the folder "MAIN" does not
 *       exist. It is advised to either create a folder called "MAIN", or to
 *       set the FOLDER constant to a folder that exists on the local
 *       system. Also, in order for this script to execute properly, the
 *       specified folder should be either closed or minimized.
 *
 *-----*/

#define FOLDER "MAIN"

    STRING szFolderName;
    NUMBER nCommand;

program

    szFolderName = FOLDER;
    nCommand     = SW_SHOW;

/*-----*\
 *
 * The following displays and changes the state of the specified folder.
 *
 *-----*/

    ShowProgramFolder(szFolderName, nCommand);
    Delay(3);

    nCommand     = SW_MAXIMIZE;
    // ShowProgramFolder maximizes FOLDER.
    ShowProgramFolder(szFolderName, nCommand);
    Delay(3);

    nCommand     = SW_MINIMIZE;
    // ShowProgramFolder minimizes FOLDER.
    ShowProgramFolder(szFolderName, nCommand);
    Delay(3);

    nCommand     = SW_RESTORE;
    // ShowProgramFolder restores FOLDER to its previous state.
    ShowProgramFolder(szFolderName, nCommand);
    Delay(3);

    nCommand     = SW_RESTORE;
    // ShowProgramFolder restores FOLDER to its original state.
    ShowProgramFolder(szFolderName, nCommand);
    Delay(3);

endprogram

// Source file: Is5fn259.rul
```

{button ,PI("",`comment`)} Feedback



String functions

The String functions provide the ability to manipulate string variables and literals. String functions behave similarly to the standard C language functions. The return values also follow the C language convention.

CopyBytes

Copies a specified number of bytes from one string to another.

GetDir

Deletes the drive designation from a path name or fully qualified filename.

GetDisk

Retrieves the disk drive designation from a path name or fully qualified filename.

NumToStr

Converts a number to a string.

ParsePath

Retrieves the drive, path, filename, or extension from a path.

StrCompare

Compares one string to another.

StrFind

Finds a string in another string.

StrGetTokens

Gets a token from a string based on specified delimiters.

StrLength

Returns the number of characters in a string.

StrRemoveLastSlash

Removes the last backslash in a path string.

StrSub

Returns a substring from a string.

StrToLower

Converts all alphabetic characters in string to lowercase.

StrToNum

Converts a string to a number.

StrToUpper

Converts all alphabetic characters in string to uppercase.

{button ,PI('','comment')} [Feedback](#)



GetDir

Syntax

```
GetDir (szPath, svDir);
```

Description

The GetDir function removes the drive designation from a path name and returns the remainder of the path name in svDir. The parameter szPath must be a path name with a drive designation. It may include a filename. The function will fail if the first two characters of szPath are not a drive letter and a colon. The path name without the drive designation is returned in svDir. In the following example, the path name C:\Windows is returned in svDir as \Windows.

```
GetDir("C:\\Windows", svDir);
```

Parameters

szPath

Specifies a path that includes a drive designation.

svDir

Returns the path without the drive designation.

Return values

0

Indicates that the function successfully returned a path name without a drive designation.

<0

Indicates that the function was unable to return a path name without a drive designations.

{button ,JI('LANGREF.HLP>Examples','GetDir_example')} [Example](#)

{button ,AL('GetDisk',0,'')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



GetDir example

```
/*-----*\
 *
 * This example illustrates the usage of the GetDir function.
 *
 \*-----*/

#define TARGET_DIR "C:\\EXAMPLE"

    STRING szTitle, szMsg, szDefPath, svDir, svResultPath, szPath;

program

    szTitle = "GetDir Example";

    szDefPath = TARGET_DIR;
    szMsg = "Which directory would you like the name of?";
    AskPath(szMsg, szDefPath, svResultPath);

    szPath = svResultPath;

/*-----*\
 *
 * GetDir is called to retrieve only the directory name of the user-specified
 * path.
 *
 \*-----*/

    if (GetDir(szPath, svDir) < 0) then

        MessageBox("GetDir failed.", SEVERE);
    else

        szMsg = "Before: %s\n\nAfter: %s";
        sprintfBox(INFORMATION, szTitle, szMsg, svResultPath, svDir);
    endif;

endprogram

// Source file: Is5fn219.rul
```

{button ,PI("",`comment')} Feedback



GetDisk

Syntax

```
GetDisk (szPath, svDisk);
```

Description

The GetDisk function extracts the disk drive letter and colon from a fully qualified filename, if included.

Parameters

szPath

Specifies a path or qualified filename that includes a drive designation. If a drive designation is not included, GetDisk will fail.

svDisk

Returns the drive designation (which includes the colon).

Return values

0

Indicates that the function successfully returned the drive letter.

< 0

Indicates that the function was unable to return the drive letter.

{button ,JI('LANGREF.HLP>Examples','GetDisk_example')} [Example](#)

{button ,AL('GetDir;GetDiskSpace;GetSystemInfo',0,','')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



GetDisk example

```
/*-----*\
 *
 * This example illustrates the uses of the GetDisk function.
 *
 * This script calls GetDisk to retrieve the drive letter from the path in
 * AskDestPath.
 *
 \*-----*/

#define TDIR "C:\\EXAMPLE"

    STRING  szMsg, szPath, svDisk;
    NUMBER  nResult;

program

start:

    Disable(BACKBUTTON);

    szMsg = "Which drive letter would you like GetDisk to identify?";
    szPath = TDIR;

    // Display a dialog to prompt the user for a target path.
    AskDestPath("Choose Location", szMsg, szPath, 0);

/*-----*\
 *
 * GetDisk is called to retrieve the drive letter of the path returned from
 * AskDestPath.
 *
 \*-----*/

    if (GetDisk(szPath, svDisk) < 0) then
        MessageBox("GetDisk failed", SEVERE);
        goto start;
    endif;

    szMsg = "The drive for that directory is %s.";
    sprintfBox(INFORMATION, "GetDisk", szMsg, svDisk);

endprogram

// Source file: Is5fn220.rul
```

{button ,PI("",`comment')} Feedback



NumToStr

Syntax

NumToStr (svString, nValue);

Description

The NumToStr function converts a number to a string.

Parameters

svString

Returns the string equivalent of nValue.

nValue

Specifies the number to convert to a string.

Return values

0

Indicates that the function successfully converted the number to a string.

< 0

Indicates that the function failed to convert the number to a string.

{button ,JI('LANGREF.HLP>Examples',`NumToStr_example`)} [Example](#)

{button ,AL(`StrToNum`,0,`,`)} [See also](#)

{button ,PI("",`comment`)} [Feedback](#)



NumToStr example

```
/*-----*\
 *
 * This example calls the NumToStr function to convert the numeric value of
 * free disk space available on the system to a string.
 *
 \*-----*/

#define   DISK_DRIVE   "C:\\\"

        STRING   szDrive, svString;
        NUMBER   nSpace, nResult;

program

    szDrive = DISK_DRIVE;
    nSpace  = GetDiskSpace(szDrive);

    nResult = NumToStr(svString, nSpace);

    if (nResult < 0) then
        MessageBox("NumToStr failed.", SEVERE);
        abort;
    endif;

    sprintfBox(INFORMATION, "NumToStr", "Disk Space: %s", svString);

endprogram

// Source file: Is5fn233.rul
```

{button ,PI("",`comment')} [Feedback](#)



StrCompare

Syntax

StrCompare (szStringA, szStringB);

Description

The StrCompare function performs compares two strings. It is not case-sensitive.

Parameters

szStringA

Specifies the first string to compare.

szStringB

Specifies the second string to compare.

Return values

< 0

Indicates that the string in szStringA has a lower value than the string in szStringB.

= 0

Indicates that the two strings are equal.

> 0

Indicates that the string in szStringA has a greater value than the string in szStringB.

Comments

- The StrCompare function compares two strings by checking the first character in each string, the second character in each string, and so on, until it finds an inequality or reaches the ends of the strings.
- The language driver for the language you select determines which string is greater or if the strings are the same. If you do not use a language driver, Windows uses an internal function. With a double-byte character set (DBCS) version of Windows, this function can compare two DBCS strings.

{button ,JI('LANGREF.HLP>Examples','StrCompare_example')}

[Example](#)

{button ,AL('StrFind;StrLength;StrSub;StrToLower;StrToNum;StrToUpper',0,'')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



StrCompare example

```
/*-----*\
 *
 * This examples illustrates the usage of the StrCompare function.
 *
 * StrCompare is called to compare two strings. The result of the comparison
 * is displayed in a message box.
 *
 \*-----*/

#include "Sddialog.h";

STRING szTitle, szMsg, szField1, szField2, szStringA, szStringB;
NUMBER nResult;

program

    szTitle      = "StrCompare Example";
    szMsg        = "Please enter two strings to compare:";
    szField1     = "String A:";
    szField2     = "String B:";
    szStringA    = "This is String A";
    szStringB    = "This is String B";

    SdShowDlgEdit2(szTitle, szMsg, szField1, szField2, szStringA, szStringB);

/*-----*\
 *
 * StrCompare is called to compare the two user-defined strings.
 *
 \*-----*/

    nResult = StrCompare(szStringA, szStringB);

    // Display the appropriate result.
    if (nResult = 0) then

        MessageBox ("Both strings are equal.", INFORMATION);
    endif;

    if (nResult < 0) then

        MessageBox ("String A has a lower value than String B", INFORMATION);
    endif;

    if (nResult > 0) then

        MessageBox ("String A has a greater value than String B", INFORMATION);
    endif;

endprogram

#define SD_SINGLE_DIALOGS 1
#define SD_SHOWDLGEDIT2 1

#include "Sddialog.rul"

// Source file: Is5fn261.rul
```

{button ,PI("", `comment`)} Feedback



StrFind

Syntax

StrFind (szString, szFindMe);

Description

The StrFind function determines if one string contains another string. StrFind will tell you if the string passed to the parameter szFindMe is located anywhere in the szString string. StrFind searches for the first occurrence of szFindMe and, if found, returns the numeric position of the character where szFindMe starts. The position of the first character in szString is zero. This function is not case sensitive.

If all you need is a TRUE or FALSE return value telling you whether one string contains another string, and the location is unimportant to you, you can also use the string find operator (%) as shown below:

```
if (szString % szFindMe) then ...
```

The above statement is similar to the StrFind function, but StrFind also returns the location where the substring starts. You can use the string find operator (%) only in Boolean expressions, and only in if statements. You cannot use it in repeat statements or while statements.

Parameters

szString

Specifies the string to search.

szFindMe

Specifies the string to find in szString.

Return values

X

If szString contains szFindMe, X is the numeric position of the first character in szFindMe. The first character in szString is in position 0 (zero).

< 0

Indicates szString does not contain szFindMe.

{button ,JI(^LANGREF.HLP>Examples', 'StrFind_example')} [Example](#)

{button ,AL(^StrCompare;StrLength;StrSub;StrToLower;StrToNum;StrToUpper',0,',')} [See also](#)

{button ,PI('', 'comment')} [Feedback](#)



StrFind example

```
-----*\
*
* This example illustrates the StrFind function.
*
* StrFind is called to search for a string within another string.  If
* successful, StrFind returns the location of the string.
*
\*-----*/

    STRING szString, szFindMe, szTitle, szMsg;
    NUMBER nLocation;

program

    szString = "This is a sample string to be searched.";
    szFindMe = "Sample String";

/*-----*\
*
* The following searches for szFindMe in szString.
*
\*-----*/
    nLocation = StrFind(szString, szFindMe);

    // Print out location of text if it was found.
    if (nLocation < 0) then

        MessageBox("StrFind failed.", SEVERE);
    else

        szTitle = "StrFind Example";
        szMsg = "'%s' was found beginning in byte: %d of string '%s'";
        sprintfBox(INFORMATION, szTitle, szMsg, szFindMe, nLocation, szString);
    endif;

endprogram

// Source file: Is5fn262.rul
```

{button ,PI("",`comment')} Feedback



StrLength

Syntax

StrLength (szString);

Description

Use the StrLength function to find the length of a string.

Parameters

szString

Specifies the string whose length is to be determined.

Return values

X

Where X is the length of the string.

< 0

Indicates that the function was unable to determine the length of the string.

{button ,JI('LANGREF.HLP>Examples','StrLength_example')} [Example](#)

{button ,AL('StrCompare;StrFind;StrSub;StrToLower;StrToNum;StrToUpper',0,'')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



StrLength example

```
/*-----*\
 *
 * This example illustrates the StrLength function.
 *
 * StrLength is called to retrieve the length of EXAMPLE_STRING.
 * The results are displayed in a message box.
 *
 \*-----*/

#define EXAMPLE_STRING "This is a test string."

    STRING szString, szTitle, szMsg;
    NUMBER nLength;

program

    szString = EXAMPLE_STRING;

/*-----*\
 *
 * The following will return the length of the string in bytes.
 *
 \*-----*/

    nLength = StrLength(szString);

    if (nLength < 0) then
        MessageBox("StrLength failed.", SEVERE);
    else
        // Display the string length.
        szTitle = "StrLength Example";
        szMsg = "String '%s' is %d bytes long.";
        sprintfBox(INFORMATION, szTitle, szMsg, szString, nLength);
    endif;

endprogram

// Source file: Is5fn263.rul
```

{button ,PI("",`comment')} [Feedback](#)



StrRemoveLastSlash

Syntax

```
StrRemoveLastSlash (svString);
```

Description

The StrRemoveLastSlash function removes the last backslash from a string.

Parameters

svString

Specifies the string from which the backslash is to be removed; returns the string without the last backslash.

Return values

0

Indicates that the function successfully removed the backslash or that the string no longer contains a final backslash.

< 0

Indicates that the function was unable to remove the backslash.

Comments

StrRemoveLastSlash does not remove the last backslash from a string that contains a colon before the backslash, such as "C:\\\" or "C:\\TEST\\ONE:". You will receive a return value of 0 (zero), indicating success, but the trailing backslash nonetheless remains. If necessary, use the following code as a workaround to remove the backslash:

```
// Get length of string, so you can extract the last three characters.
nReturn = StrLength(svString);
StrSub(svSubStr, svString, nReturn - 2, 3);

// Check if last three characters are colon and backslash escape sequence.
// If so, remove them by assigning svString to the substring (svSubStr).
// If not, call StrRemoveLastSlash as usual.

if svSubStr = ":\\" then
    StrSub(svSubStr, svString, 0, nReturn - 1);
    svString = svSubStr;
else
    StrRemoveLastSlash(svString);
endif;
```

{button ,JI('LANGREF.HLP>Examples','StrRemoveLastSlash_example')}

[Example](#)

{button ,AL('StrCompare;StrFind;StrGetTokens;StrLength;StrSub;StrToLower;StrToNum;StrToUpper',0,',')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



StrRemoveLastSlash example

```
/*-----*\
 *
 * This example illustrates the usage of the StrRemoveLastSlash function.
 *
 * The AskPath dialog box is displayed, prompting the user for a directory
 * with a backslash in the end. StrRemoveLastSlash is then called to remove it.
 *
 \*-----*/

    STRING szDefPath, svString, szMsg, szTitle;
    NUMBER nStyle;

program

    szDefPath = "C:\\EXAMPLE\\";
    szMsg      = "Please insert the directory string which should have the last slash
removed.";
    szTitle    = "StrRemoveLastSlash Example";
    nStyle     = 0;
    AskPath(szMsg, szDefPath, svString);

/*-----*\
 *
 * StrRemoveLastSlash removes the last slash from directory set to the
 * svString variable.
 *
 \*-----*/

    if (StrRemoveLastSlash(svString) < 0) then
        MessageBox("StrRemoveLastSlash failed.", SEVERE);
    else
        szMsg = "The following string had the last slash removed:\n\n%s";
        SprintfBox(INFORMATION, szTitle, szMsg, svString);
    endif;

endprogram

// Source file: Is5fn174.rul
```

{button ,PI("",`comment')} [Feedback](#)



StrSub

Syntax

StrSub (svSubStr, szString, nStart, nLength);

Description

The StrSub function copies part of the string specified by szString, beginning at the location specified by nStart. The parameter nLength specifies the number of characters to copy.



The nStart and nLength values used by StrSub are numbers of bytes. When using double-byte characters, you must multiply the number of characters by 2 in order to get correct values for nStart and nLength.

Parameters

svSubStr

Returns the substring copied from szString.

szString

Specifies the string from which the substring is to be copied.

nStart

Specifies an offset into szString that identifies the first character to be copied. Note that the position of the first character in szString is 0 (zero). If the value passed in nStart is equal to or greater than the length of szString, a null string ("") is returned in svSubStr.

nLength

Specifies the number of characters to copy from szString. If this value specifies more characters than exist between nStart and the end of szString, all characters from nStart to the end of the string are returned in svSubStr.

Return values

X

Where X equals the number of characters in svSubStr.

{button ,JI('LANGREF.HLP>Examples','StrSub_example')} [Example](#)

{button ,AL(' StrCompare;StrFind;StrLength;StrToLower;StrToNum;StrToUpper',0,',')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



StrSub example

```
/*-----*\
 *
 * This example illustrates the usage the StrSub function.
 *
 * StrSub is called to retrieve 10 bytes of the sample string.
 *
 \*-----*/

    STRING svSubStr, szString, szQuestion, szTitle, szMsg;
    NUMBER nStart, nLength;

program

    szQuestion = "Please enter the date in this form MM/DD/YYYY:";
    AskText(szQuestion, "07/19/1996", szString);

/*-----*\
 *
 * StrSub is called to retrieve only the last four bytes of the user-defined
 * string.
 *
 \*-----*/

    nStart = 6;
    nLength = 4;
    if (StrSub(svSubStr, szString, nStart, nLength) < 0) then

        MessageBox("StrSub failed.", SEVERE);
    endif;

    szTitle = "StrSub Example";
    szMsg = "Original string: %s\n\nSub string: %s";

    // Display the edited string.
    sprintfBox(INFORMATION, szTitle, szMsg, szString, svSubStr);

endprogram

// Source file: Is5fn264.rul
```

{button ,PI("",`comment')} [Feedback](#)



StrToLower

Syntax

StrToLower (svTarget, szSource);

Description

The StrToLower function converts all the letters in a string to lowercase. This function does not affect non-alphabetic characters.

Parameters

svTarget

Returns the string in szSource converted to all lowercase characters.

szSource

Specifies the string to convert to all lowercase characters.

Return values

0

Indicates that the function successfully changed the case of the string.

< 0

Indicates that the function was unable to change the case of the string.

{button ,JI('LANGREF.HLP>Examples','StrToLower_example')}[Example](#)

{button ,AL('StrCompare;StrFind;StrLength;StrSub;StrToNum;StrToUpper',0,','')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



StrToLower example

```
/*-----*\
 *
 * This example illustrates the usage of the StrToUpper and StrToLower
 * functions.
 *
 * StrToUpper is called to convert the characters in the szSource string from
 * lowercase to uppercase.
 *
 * StrToLower is then called to convert the uppercase characters to lowercase.
 *
 \*-----*/

    STRING szSource, svTarget, szTitle, szMsg;

    NUMBER nReturn;

program

/*-----*\
 *
 * The following will change the lowercase letters to uppercase.
 *
 \*-----*/
    szSource = "aBcDeF";
    nReturn = StrToUpper(svTarget, szSource);

    if (nReturn < 0) then

        MessageBox("StrToUpper failed.", SEVERE);
    endif;

    szTitle = "StrToUpper & StrToLower";
    szMsg = "Original: %s\n\nModified: %s";

    // Display the altered string.
    sprintfBox(INFORMATION, szTitle, szMsg, szSource, svTarget);

/*-----*\
 *
 * The following will change the uppercase letters to lowercase
 *
 \*-----*/
    szSource = "ABC123*&?d";
    nReturn = StrToLower(svTarget, szSource);

    if (nReturn < 0) then

        MessageBox("StrToLower failed.", SEVERE);
    endif;

    // Display the altered string.
    szMsg = "Original: %s\n\nModified: %s\n(Note--Non-alphabetic chars " +
        "are not changed)";
    sprintfBox (INFORMATION, szTitle, szMsg, szSource, svTarget);

endprogram

// Source file: Is5fn265.rul
```

{button ,PI("", `comment`)} Feedback



StrToNum

Syntax

StrToNum (nvVar, szString);

Description

The StrToNum function converts a string to a number, much like the C function atol(). It inspects svString, starting with the character at position 0 and continuing through the string until it reaches the end of the string or encounters a character that is not in the range "0".."9". (The first character in the string may be a plus or minus sign.) Then one of the following processes occurs:

- n If all of the characters in the string are in the range "0".."9", the number represented by the string is assigned to nvVar.
- n If the string begins with one or more characters in the range "0".."9" but also contains one or more non-numeric characters, a number based on the characters to the left of the first non-numeric character is assigned to nvVar. For example, if szString is "-123ABC456", nvResult will be -123.
- n If the first character in the string is not in the range "0".."9" and is not a plus or minus sign, the function fails.
- n If the first character in the string is a plus or minus sign and the second character is not in the range "0".."9", the function fails.

Parameters

nvVar

Returns the number created from the string in szString.

szString

Specifies the string to convert to a number.

Return values

0

Indicates that the function successfully converted the string into a numeric value.

< 0

Indicates that the function was unable to convert the string into a numeric value.

{button ,JI('LANGREF.HLP>Examples','StrToNum_example')} [Example](#)

{button ,AL(' StrCompare;StrFind;StrLength;StrSub;StrToLower;StrToUpper',0,',')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



StrToNum example

```
*-----*\
*
* This example illustrates the StrToNum.
*
* StrToNum is called to convert "1222240" to 1222240.
* and "1222ABC40" to 1222
*
\*-----*/
    STRING szString, szTitle, szMsg;
    NUMBER nVar;

program

/*-----*\
*
* Convert szString to a number and store in nVar.
*
\*-----*/
    szString = "1222240";
    if (StrToNum(nVar, szString) < 0) then

        MessageBox("StrToNum failed.", SEVERE);
    else

        szTitle = "StrToNum Example";
        szMsg = "String: %s\n\nNumber: %d";
        sprintfBox(INFORMATION, szTitle, szMsg, szString, nVar);
    endif;

/*-----*\
*
* Convert szString (which contains non-numeric character)
* to a number and store in nVar.
*
\*-----*/
    szString = "1222ABC40";
    if (StrToNum(nVar, szString) < 0) then

        MessageBox("StrToNum failed.", SEVERE);
    else

        szTitle = "StrToNum Example";
        szMsg = "String: %s\n\nNumber: %d";
        sprintfBox(INFORMATION, szTitle, szMsg, szString, nVar);
    endif;

endprogram

// Source file: Is5fn266.rul
```

{button ,PI("",`comment')} [Feedback](#)



StrToUpper

Syntax

StrToUpper (svTarget, szSource);

Description

The StrToUpper function converts all the letters in a string to uppercase. This function does not affect non-alphabetic characters.

Parameters

svTarget

Returns the string in szSource converted to all uppercase characters.

szSource

Specifies the string to convert to all uppercase characters.

Return values

0

Indicates that the function successfully changed the case of the string.

< 0

Indicates that the function was unable to change the case of the string.

{button ,JI('LANGREF.HLP>Examples','StrToUpper_example')}[Example](#)

{button ,AL('StrCompare;StrFind;StrLength;StrSub;StrToLower;StrToNum',0,',')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



StrToUpper example

```
/*-----*\
 *
 * This example illustrates the usage of the StrToUpper and StrToLower
 * functions.
 *
 * StrToUpper is called to convert the characters in the szSource string from
 * lowercase to uppercase.
 *
 * StrToLower is then called to convert the uppercase characters to lowercase.
 *
 \*-----*/

    STRING szSource, svTarget, szTitle, szMsg;

    NUMBER nReturn;

program

/*-----*\
 *
 * The following will change the lowercase letters to uppercase.
 *
 \*-----*/
    szSource = "aBcDeF";
    nReturn = StrToUpper(svTarget, szSource);

    if (nReturn < 0) then

        MessageBox("StrToUpper failed.", SEVERE);
    endif;

    szTitle = "StrToUpper & StrToLower";
    szMsg = "Original: %s\n\nModified: %s";

    // Display the altered string.
    sprintfBox(INFORMATION, szTitle, szMsg, szSource, svTarget);

/*-----*\
 *
 * The following will change the uppercase letters to lowercase
 *
 \*-----*/
    szSource = "ABC123*&?d";
    nReturn = StrToLower(svTarget, szSource);

    if (nReturn < 0) then

        MessageBox("StrToLower failed.", SEVERE);
    endif;

    // Display the altered string.
    szMsg = "Original: %s\n\nModified: %s\n(Note--Non-alphabetic chars " +
        "are not changed)";
    sprintfBox (INFORMATION, szTitle, szMsg, szSource, svTarget);

endprogram

// Source file: Is5fn265.rul
```

{button ,PI("", `comment`)} Feedback



Version checking functions

The following functions allow you to access version information present in Windows 3.1 or higher files. In order to use the functions you need to know background information about version resources. Review the Microsoft Windows Programmer's Reference, Volume 4: Resources manual to gain a better understanding of the version resource. The descriptions of the functions assume that you are completely familiar with the concepts behind the version resource.

The following functions obtain a specific file's version, find a file and get its version, or search for an existing file and try to install a newer version of the file. The functions work with either compressed or uncompressed files.

VerCompare

Compares two strings containing version information.

VerFindFileVersion

Searches for the specified file and retrieves its version and location.

VerGetFileVersion

Retrieves the version of a specified file.

VerSearchAndUpdateFile

Replaces an existing file with a more recent version. If the specified file does not exist, the more recent version is installed.

VerUpdateFile

Replaces an existing file with a more recent version. If the specified file does not exist, the more recent version is *not* installed.

{button ,PI("",`comment')} [Feedback](#)



VerCompare

Syntax

VerCompare (szVersionInfo1, szVersionInfo2, nCompareFlag);

Description

The VerCompare function compares two strings that contain version information.

Parameters

szVersionInfo1

Specifies the first version string in the following format:

`Major version number.Minor version number`

If szVersionInfo1 is 2.1.2.0, the major version number is 2.1 and the minor version number is 2.0.

szVersionInfo2

Specifies the second version string in the same format.

nCompareFlag

Pass the predefined constant VERSION to specify the comparison of version numbers. No other values are allowed in this parameter.

Return values

EQUALS

Indicates that the two strings are of equal value.

GREATER_THAN

Indicates that the first string contains a value greater than the second.

LESS_THAN

Indicates that the first string contains a value less than the second.

{button ,JI('LANGREF.HLP>Examples','VerCompare_example')} [Example](#)

{button ,AL('VerFindFileVersion;VerGetFileVersion;VerUpdateFile',0,'')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



VerCompare example

```
/*-----*\
*
* This example illustrates the VerFindFileVersion and VerCompare
* functions.
*
* This script calls VerFindFileVersion to find the target file and retrieve
* its version information. If the specified target file is not found, the
* source file is copied to TARGETDIR. If found, VerCompare is called to
* compare the version number of the EXAMPLE file found on the target
* system (the target file) to the version number of the file in SRCDIR (the
* source file). If the source file version number is newer than the target
* file, the target file is overwritten with the source file.
*
* NOTE: In order for this script to run properly, it is necessary for the
*       preprocessor constants to be set to valid a file name and paths on
*       the target system.
*
\*-----*/

#define EXAMPLE      "STIRINFC.DLL"
#define SOURCE_DIR  "C:\\EXAMPLE\\SOURCE"
#define TARGET_DIR  "C:\\EXAMPLE\\TARGET"
#define SOURCE_VER  "2.0.1.0"

    STRING szFileName, svPath, svVersionNumber, szVersionInfo1, szVersionInfo2;
    STRING szTitle, szMsg;
    NUMBER nResult, nCompareFlag;

program

    // Set the file to be updated.
    szFileName = EXAMPLE;

    // Set the SRCDIR and TARGETDIR locations.
    SRCDIR     = SOURCE_DIR;
    TARGETDIR  = TARGET_DIR;
    szTitle    = "VerCompare and VerFindFileVersion";

/*-----*\
*
* Find szFileName on the target system and retrieve its version number.
*
\*-----*/
    nResult = VerFindFileVersion(szFileName, svPath, svVersionNumber);

    if(nResult = FILE_NOT_FOUND) then

        // If szFileName not found, copy source file to TARGETDIR.
        szMsg = "Unable to locate %. Copying %s to %s.";
        sprintfBox(INFORMATION, szTitle, szMsg, szFileName,
                  TARGETDIR);

        CopyFile(szFileName, szFileName);
        abort;
    elseif (nResult = FILE_NO_VERSION) then

        TARGETDIR = svPath;

        // If no version number found, copy source file to svPath and exit.
```

```

        szMsg = "%s version number not found. Copying %s to %s.";
        sprintfBox(INFORMATION, szTitle, szMsg, szFileName, szFileName,
                  TARGETDIR);

        CopyFile(szFileName, szFileName);
        abort;
    elseif (nResult < 0) then

        MessageBox("VerFindFileVersion failed.", SEVERE);
        abort;
    endif;

/*-----*\
*
* Compare the versions of the two files. It is assumed that the source
* version number is known.
*
\*-----*/
szVersionInfo1 = svVersionNumber;
MessageBox(szVersionInfo1, INFORMATION);
szVersionInfo2 = SOURCE_VER;
MessageBox(szVersionInfo2, INFORMATION);
nCompareFlag = VERSION;

nResult = VerCompare(szVersionInfo1, szVersionInfo1, nCompareFlag);

// If the source file is a more current version, install it.
if (nResult = GREATER_THAN) then

    TARGETDIR = svPath;
    szMsg = "%s updated into the %s directory.";
    sprintfBox(INFORMATION, szTitle, szMsg, szFileName, TARGETDIR);

    CopyFile(szFileName, szFileName);

// If the target file is a more current version, do not install.
elseif (nResult = LESS_THAN) then

    szMsg = "No need for an upgrade, the most current version of %s is " +
           "installed.";
    sprintfBox(INFORMATION, szTitle, szMsg, szFileName);

// If both the target and source versions are the same, do not install.
elseif (nResult = EQUALS) then

    MessageBox("Versions are equal. No update needed.", INFORMATION);
endif;

endprogram

// Source file: Is5fn269.rul

```

{button ,PI("",`comment')} [Feedback](#)



VerFindFileVersion

Syntax

VerFindFileVersion (szFileName, svPath, svVersionNumber);

Description

The VerFindFileVersion function searches for a specified file and retrieves the file version and location. VerFindFileVersion uses the following search algorithm to find the file:

1. First, it searches the Windows directory
2. Next, the Windows\System directory
3. Next, the directory specified by the TARGETDIR system variable
4. Next, the directories specified by the PATH environment variable
5. Finally, the directory from which Setup.exe is running

Parameters

szFileName

Specifies the unqualified name of the file whose version is to be obtained. Do not specify a drive designation or path in this parameter.

svPath

Returns the complete path (including the drive designation) to the folder in which the file resides.

svVersionNumber

Returns the version number of the file in the following format:

Major version number.Minor version number

For example, if svVersionNumber returns 2.1.2.0, the major version number is 2.1 and the minor version number is 2.0.

Return values

0

Indicates that the function successfully returned the version information.

FILE_NO_VERSION

Indicates that the file was found but did not contain version information.

FILE_NOT_FOUND

Indicates that the file was not found.

Comments

When using VerFindFileVersion, you may need to set a value for TARGETDIR other than the value automatically set by InstallShield. Since the function looks for the file in the TARGETDIR directory, you may need to reset the value of the system variable temporarily to ensure that VerFindFileVersion will find the file. If you need to do this, use [VarSave](#) to save the value of TARGETDIR before temporarily setting it to another directory. After the VerFindFileVersion function call, reset TARGETDIR using [VarRestore](#).

{button ,JI('LANGREF.HLP>Examples','VerFindFileVersion_example')} [Example](#)

{button ,AL('VerCompare;VerGetFileVersion;VerUpdateFile',0,'')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



VerFindFileVersion example

```
/*-----*\
*
* This example illustrates the VerFindFileVersion and VerCompare
* functions.
*
* This script calls VerFindFileVersion to find the target file and retrieve
* its version information. If the specified target file is not found, the
* source file is copied to TARGETDIR. If found, VerCompare is called to
* compare the version number of the EXAMPLE file found on the target
* system (the target file) to the version number of the file in SRCDIR (the
* source file). If the source file version number is newer than the target
* file, the target file is overwritten with the source file.
*
* NOTE: In order for this script to run properly, it is necessary for the
*       preprocessor constants to be set to valid a file name and paths on
*       the target system.
*
\*-----*/

#define EXAMPLE      "STIRINFC.DLL"
#define SOURCE_DIR  "C:\\EXAMPLE\\SOURCE"
#define TARGET_DIR  "C:\\EXAMPLE\\TARGET"
#define SOURCE_VER  "2.0.1.0"

    STRING szFileName, svPath, svVersionNumber, szVersionInfo1, szVersionInfo2;
    STRING szTitle, szMsg;
    NUMBER nResult, nCompareFlag;

program

    // Set the file to be updated.
    szFileName = EXAMPLE;

    // Set the SRCDIR and TARGETDIR locations.
    SRCDIR     = SOURCE_DIR;
    TARGETDIR  = TARGET_DIR;
    szTitle    = "VerCompare and VerFindFileVersion";

/*-----*\
*
* Find szFileName on the target system and retrieve its version number.
*
\*-----*/
    nResult = VerFindFileVersion(szFileName, svPath, svVersionNumber);

    if(nResult = FILE_NOT_FOUND) then

        // If szFileName not found, copy source file to TARGETDIR.
        szMsg = "Unable to locate %s. Copying %s to %s.";
        sprintfBox(INFORMATION, szTitle, szMsg, szFileName,
                  TARGETDIR);

        CopyFile(szFileName, szFileName);
        abort;
    elseif (nResult = FILE_NO_VERSION) then

        TARGETDIR = svPath;

        // If no version number found, copy source file to svPath and exit.
```

```

        szMsg = "%s version number not found. Copying %s to %s.";
        sprintfBox(INFORMATION, szTitle, szMsg, szFileName, szFileName,
                  TARGETDIR);

        CopyFile(szFileName, szFileName);
        abort;
    elseif (nResult < 0) then

        MessageBox("VerFindFileVersion failed.", SEVERE);
        abort;
    endif;

/*-----*\
*
* Compare the versions of the two files. It is assumed that the source
* version number is known.
*
\*-----*/
szVersionInfo1 = svVersionNumber;
MessageBox(szVersionInfo1, INFORMATION);
szVersionInfo2 = SOURCE_VER;
MessageBox(szVersionInfo2, INFORMATION);
nCompareFlag = VERSION;

nResult = VerCompare(szVersionInfo1, szVersionInfo1, nCompareFlag);

// If the source file is a more current version, install it.
if (nResult = GREATER_THAN) then

    TARGETDIR = svPath;
    szMsg = "%s updated into the %s directory.";
    sprintfBox(INFORMATION, szTitle, szMsg, szFileName, TARGETDIR);

    CopyFile(szFileName, szFileName);

// If the target file is a more current version, do not install.
elseif (nResult = LESS_THAN) then

    szMsg = "No need for an upgrade, the most current version of %s is " +
           "installed.";
    sprintfBox(INFORMATION, szTitle, szMsg, szFileName);

// If both the target and source versions are the same, do not install.
elseif (nResult = EQUALS) then

    MessageBox("Versions are equal. No update needed.", INFORMATION);
endif;

endprogram

// Source file: Is5fn269.rul

```

{button ,PI("",`comment')} [Feedback](#)



VerGetFileVersion

Syntax

VerGetFileVersion (szFileName, svVersionNumber);

Description

The VerGetFileVersion function retrieves the version of a specified file.

Parameters

szFileName

Specifies the fully qualified name of the file whose version number is to be retrieved.

svVersionNumber

Returns the version number in the following format:

Major version number.Minor version number

For example, if svVersionNumber returns 2.1.2.0, the major version number is 2.1 and the minor version number is 2.0.

Return values

0

Indicates that the function successfully returned the version information.

FILE_NOT_FOUND

Indicates that the specified file was not found.

FILE_NO_VERSION

Indicates that the file was found but did not contain version information.

{button ,JI(`LANGREF.HLP>Examples`,`VerGetFileVersion_example`)} [Example](#)

{button ,AL(`VerCompare;VerFindFileVersion;VerUpdateFile`,0,`,`)} [See also](#)

{button ,PI(``,`comment`)} [Feedback](#)



VerGetFileVersion example

```
/*-----*\
 *
 * This example illustrates the usage of the VerGetFileVersion function.
 *
 * The script below calls VerGetFileVersion to retrieve the version number of
 * the STIRINFC.DLL file. This string is displayed in a message box.
 *
 \*-----*/

#define EXAMPLE_FILE1 "STIRINFC.DLL"
#define SOURCE_DIR    "WINDOWS\\SYSTEM"

    NUMBER nResult;
    STRING szFile, szPath, szTitle, szMsg, svVersionNumber;

program

    // Identify the target file.
    szFile = EXAMPLE_FILE1;

    // Identify the SRCDIR location.
    SRCDIR = TARGETDISK ^ SOURCE_DIR;

    szPath = SRCDIR ^ szFile;
    nResult = VerGetFileVersion(szFile, svVersionNumber);

    if (nResult = FILE_NO_VERSION) then
        szMsg = "Target file does not contain version information.";
        MessageBox(szMsg, INFORMATION);
    elseif (nResult = FILE_NOT_FOUND) then
        szMsg = "Source file could not be found.";
        MessageBox(szMsg, INFORMATION);
        abort;
    else
        szTitle = "VerGetFileVersion Example";
        szMsg    = "The version number for %s is %s";
        sprintfBox(INFORMATION, szTitle, szMsg, szFile, svVersionNumber);
    endif;

endprogram

// Source file: Is5fn271.rul
```

{button ,PI("",`comment')} [Feedback](#)



VerSearchAndUpdateFile

Syntax

VerSearchAndUpdateFile (szFileName, nUpdateFlag, svInstalledFile);

Description

The VerSearchAndUpdateFile function searches for the specified file and installs a newer version of the file if necessary. If the function finds the file, it compares the version number of the existing file to the version number of the new file. If the existing file is older, it is replaced with the new file. The new file must be in the directory specified by the system variable SRCDIR. If the function does not find an existing file, it copies the new file to the target system. Microsoft Windows decides where the file is installed depending on the type of the file. For example, .dlls and system drivers are installed to the Windows\System directory.

VerSearchAndUpdateFile uses the following search algorithm to find the existing file:

1. First, it searches the Windows or Win95 directory
2. Next, the System directory
3. Next, the directory specified by the TARGETDIR system variable
4. Next, the directories specified by the PATH environment variable
5. Finally, the directory from which Setup.exe is running

Parameters

szFileName

Specifies the unqualified name of the file to install. Do not specify a drive designation or path in this parameter.

nUpdateFlag

Specifies whether the file should be updated unconditionally or only if the version of the file found on the target system is older than the version of the file that you have shipped. Pass one of the following predefined constants in this parameter:

VER_UPDATE_COND

Updates the existing file only if it is an older version.

VER_UPDATE_ALWAYS

Updates the existing file even if it is a newer version.

svInstalledFile

Returns the fully qualified name of the file installed by the function. If the file you want to replace is in use, the file is installed to the same directory with a slightly different name. The file is renamed with a tilde (~) character in the first character of the extension. For example, if you are installing the file Shell.dll and the file is locked, the file is copied as Shell.~ll. The name is returned in this variable.

Return values

FILE_INSTALLED

Function was successful in installing the file.

FILE_IS_LOCKED

Indicates that the file is in use by Windows and cannot be replaced. The new file is copied to the same directory with a new name.

FILE_NO_VERSION

Indicates that the file was found, but it did not contain version information. File update is not performed.

FILE_RD_ONLY

Indicates that the existing file is write-protected. The script should reset the read-only flag of the destination file before proceeding with the setup and then attempt to install the file again.

FILE_SRC_OLD

Indicates that the file to install has the same date or is older than the preexisting file.

OUT_OF_DISK_SPACE

Indicates that the function cannot create the file due to insufficient disk space on the destination drive. File update is not performed.

VER_DLL_NOT_FOUND

Indicates Ver.dll was not found. File update is not performed.

OTHER_FAILURE

Indicates an unspecified error occurred. File update is not performed.

Comments

- n When using VerSearchAndUpdateFile, you may need to set values for SRCDIR and/or TARGETDIR other than those automatically set by InstallShield. Since the function looks for the files in the SRCDIR and TARGETDIR directories, you may need to temporarily reset the value of the system variables to ensure that VerSearchAndUpdateFile will find the files. If you need to do this, use [VarSave](#) to save the values of SRCDIR and TARGETDIR before temporarily setting them to another value. After the VerSearchAndUpdateFile function call, reset SRCDIR and TARGETDIR using [VarRestore](#).
- n For file transfer, possible alternatives to VerSearchAndUpdateFile may be [XCopyFile](#), which can do version checking, mark locked .dll and .exe files for update after system reboot, and increment registry reference counters for shared .dll and .exe files.

{button ,JI('LANGREF.HLP>Examples','VerSearchAndUpdateFile_example')} [Example](#)

{button ,AL('VerGetFileVersion;VerFindFileVersion;VerUpdateFile',0,','')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



VerSearchAndUpdateFile example

```
/*-----*\
 *
 * This example illustrates the usage of the VerSearchAndUpdateFile function.
 *
 * The first call replaces the UPDATE_FILE1 file regardless of the version on
 * the target system.
 *
 * The second call replaces the UPDATE_FILE2 only if the version in the source
 * directory is newer than the version in the target directory. This example
 * checks for and displays errors.
 *
 * NOTE: The preprocessor constants should be set to valid files and paths in
 * order for this script to run properly.
 *
 * The first time this script runs, target files within the TDIR
 * constant directory should not exist. It is advised to run this
 * script multiple times.
 *
 *-----*/

#define UPDATE_FILE1 "EXAMPLE.TXT"
#define UPDATE_FILE2 "README.TXT"
#define SDIR "C:\\EXAMPLE"
#define TDIR "C:\\EXAMPLE\\TARGET"

    STRING szFileName, svInstalledFile, szTitle, szMsg;
    NUMBER nUpdateFlag, nResult;
    BOOL bDone;

program

    // Set source and target directories.
    SRCDIR = SDIR;
    TARGETDIR = TDIR;

    szTitle = "VerSearchAndUpdateFile Example";
    szMsg = UPDATE_FILE1 + " was successfully updated.";

/*-----*\
 *
 * UPDATE_FILE1 is updated regardless of version number and a message is
 * displayed.
 *
 *-----*/

    if (VerSearchAndUpdateFile(UPDATE_FILE1, VER_UPDATE_ALWAYS,
        svInstalledFile) = 0) then

        SprintfBox(INFORMATION, szTitle, szMsg);
    endif;

    bDone = FALSE;

    // Begin the while loop.
    while (bDone = FALSE)

/*-----*\
 *
 * VerSearchAndUpdateFile is called within a while loop. If a specific error
```

```

* occurs, a message is displayed.
*
\*-----*/

nResult = VerSearchAndUpdateFile(UPDATE_FILE2, VER_UPDATE_COND,
                                svInstalledFile);

switch (nResult)
    case 0:

        // VerSearchAndUpdate successful.
        sprintfBox(INFORMATION, szTitle, szMsg);
        bDone = TRUE;

        // The target file does not have a version number.
        case FILE_NO_VERSION:

            // Ask the user if the file should be updated regardless.
            if (AskYesNo("Version number was not found.\nDo you still wish " +
                        "to update " + UPDATE_FILE2 + "?", YES) = YES) then

/*-----*\
*
* Update the file UPDATE_FILE2 regardless of version number.
*
\*-----*/

        VerSearchAndUpdateFile(UPDATE_FILE2, VER_UPDATE_ALWAYS,
                                svInstalledFile);

        bDone = TRUE;
    else

        bDone = TRUE;
    endif;

// The target file is locked.
case FILE_IS_LOCKED:

    MessageBox("The target file is locked.\n\nPlease close all " +
                "programs and run Setup again.", INFORMATION);
    bDone = TRUE;

// The target file is read-only.
case FILE_RD_ONLY:

    // Ask the user if Setup should
    if(AskYesNo("File is read-only.\nShould Setup remove the " +
                "write-protection of " + UPDATE_FILE2 + "?", YES) = YES) then

        // Change the attribute of the target file to normal.
        SetFileInfo(svInstalledFile, FILE_ATTRIBUTE, FILE_ATTR_NORMAL, "");
        bDone = FALSE;
    else

        bDone = TRUE;
    endif;

// The target disk does not have enough space.
case OUT_OF_DISK_SPACE:

    MessageBox("You need more free space for this update.", SEVERE);
    bDone = TRUE;

// The required VER.DLL file was not found.

```

```
case VER_DLL_NOT_FOUND:

    MessageBox("VER.DLL was not found.", SEVERE);
    bDone = TRUE;

// Some other error occurred.
case OTHER_FAILURE:

    MessageBox("Update has failed.", SEVERE);
    bDone = TRUE;
default:

    bDone = TRUE;
endswitch;
endwhile;

endprogram

// Source file: Is5fn183.rul
```

{button ,PI("",`comment')} Feedback



VerUpdateFile

Syntax

VerUpdateFile (szFileName, nUpdateFlag, svInstalledFilePath);

Description

The VerUpdateFile function uses the version information of a specified file to determine whether or not to install the file on the target directory. VerUpdateFile gets the filename specified in szFileName. If a fully qualified filename is not specified in szFileName, VerUpdateFile uses the following search algorithm to find the file (the target file):

1. First, it searches the Windows or Win95 directory
2. Next, the Windows\System directory
3. Next, the directory specified by the system variable TARGETDIR
4. Next, the directories specified by the PATH environment variable
5. Finally, the directory from which Setup.exe is running

VerUpdateFile then compares the version of the file with the same name in SRCDIR (the source file), if it exists, with the version of the target file. If the source file has a more recent version number than the target file, the target file is replaced by the source file. If the target file does not exist, InstallShield copies the source file to the target location.

When the SHAREDFILE or LOCKEDFILE option is used in the parameter nUpdateFlag and .dll or .exe files to be updated are in use by the system, renamed copies of the source files are transferred to the target system and the system variable BATCH_INSTALL is set to TRUE. Then, when RebootDialog or SdFinishReboot is called at the end of the setup and the system is restarted, the locked files are updated. Refer to [RebootDialog](#) and [SdFinishReboot](#) for more information on updating locked files. The system variable BATCH_INSTALL can be tested to determine if locked .dll or .exe files were encountered. You cannot use the SHAREDFILE and LOCKEDFILE options simultaneously—you must use one or the other.

Parameters

szFileName

Specifies the qualified or unqualified name of the file to update. If the name is unqualified (that is, if it does not include a drive designation or path), InstallShield searches the Windows or Win95 directory, the System directory, the directories specified by the PATH environment variable, and then the path of the InstallShield executable file for a matching file. VerUpdateFile takes the filename portion of szFileName and uses it to identify the file in SRCDIR that is used as the source file.

nUpdateFlag

Specifies whether the file is updated unconditionally or updated only if the version of the target file found is older than the version of the source file. Pass one of the following predefined constants in this parameter. You can combine the constant SHAREDFILE with one of the other constants using the bitwise OR operator (|). However, you cannot combine SHAREDFILE and LOCKEDFILE.

LOCKEDFILE

Causes VerUpdateFile to record locked .dll and .exe files for update when Windows or the system is rebooted. A locked file is a file that is in use by an application or the system when InstallShield attempts to access or update the file. The LOCKEDFILE option works like SHAREDFILE except that LOCKEDFILE does not create registry entries or modify the registry reference counter. You cannot use the LOCKEDFILE option when using the SHAREDFILE option. There are some unshared files (such as shell extensions) for which the script writer does not want a registry entry and reference counter. These files should never be uninstalled, except by the application itself. LOCKEDFILE allows VerUpdateFile to handle locked files that are not shared.

SHAREDFILE

Combines shared and locked file handling by causing VerUpdateFile to treat all files as shared, and to record locked .dll and .exe files for update when Windows or the system restarts. See [RebootDialog](#) and [SdFinishReboot](#).

Under Windows 95 and Windows NT, the SHAREDFILE option causes VerUpdateFile to treat all files as shared files and increment the registry reference counter by one when the file exists in the target directory and it has a reference count greater than 0. If the shared file does not exist in the target directory and it has no reference counter, InstallShield creates the counter and sets it to 1. If the shared file already exists in the target directory but has no reference counter, InstallShield creates the counter and initializes it to 2 as a precaution against accidental removal during uninstallation.

Under Windows 3.1, there is no registry reference counter, so no files updated using the SHAREDFILE option are logged for uninstallation.

SELFREGISTER

Carries out the self-registration process immediately, when using the "non-batch method" of installing self-registering files.

If you have called Enable(SELFREGISTERBATCH), this option queues up self-registering files for registration. The files are registered once Do(SELFREGISTRATIONPROCESS) is called, when using the "batch method" of installing self-registering files.

Always use SELFREGISTER together with the constant SHAREDFILE, using the bitwise OR operator (|).

VER_UPDATE_ALWAYS

Updates the file regardless of the version numbers.

VER_UPDATE_COND

Updates the file only if the file being replaced is an older version.

svInstalledFilePath

Returns the fully qualified name of the file that was installed. If the file you want to replace is in use, the file is installed to the same directory with a modified name. InstallShield uses a tilde (~) character to replace the first character of the file's extension.

For example, if you are updating the file Shell.dll and the target file is locked, the source file is copied to the target directory as Shell.~ll. This name is returned in the parameter svInstalledFilePath.

If the SHAREDFILE option is used in the parameter nUpdateFlag and locked files are properly committed for update when Windows or the system restarts, the ~ modified name versions of the files are deleted when the update takes place.

Return values

FILE_INSTALLED

Indicates that the function successfully installed the file. This constant is equal to 0 (zero). All other return values are less than zero (<0).

FILE_IS_LOCKED

Indicates that the existing file is in use by Windows and cannot be replaced. The new file is copied to the same directory with a new name, as described above.

FILE_NO_VERSION

Indicates that the file was found, but it did not contain version information in it. File update is not performed.

FILE_RD_ONLY

Indicates that the existing file is write-protected. You must reset the read-only bit in the destination file before proceeding with the setup, and then attempt to install the file again.

FILE_SRC_EQUAL

Indicates that the file you want to install has the same version as the existing file. File update is not performed if the VER_UPDATE_COND flag is set.

FILE_SRC_OLD

Indicates that the file you want to install is older than the existing file. File update is not performed if the VER_UPDATE_COND flag is set.

OUT_OF_DISK_SPACE

Indicates that the function cannot create the file due to insufficient disk space on the destination drive. File update is not performed.

-51

A self-registering file did not register successfully.

OTHER_FAILURE

Indicates an unspecified error occurred. File update is not performed.

Comments

- n Before calling any of the functions that use the SHAREDFILE option, and before calling [CommitSharedFiles](#), the application information must be set using [InstallationInfo](#), and uninstallShield information must be set using the [DeinstallStart](#) function.
- n When using VerUpdateFile, you may need to set a value for SRCDIR other than the value automatically set by InstallShield. Since the function looks for the file in the SRCDIR directory, you may need to temporarily reset the value of the system variable to ensure that VerUpdateFile will find the file. If you need to do this, use [VarSave](#) to save the value of SRCDIR before temporarily setting it to another directory. After the VerUpdateFile function call, reset SRCDIR using [VarRestore](#).
- n For file transfer, preferable alternatives to VerUpdateFile may be [XCopyFile](#), which can perform version checking, mark locked .dll and .exe files for update after system reboot, and increment registry reference counters for shared .dll and .exe files.

```
{button ,JI('LANGREF.HLP>Examples','VerUpdateFile_example')} Example  
{button ,AL('CommitSharedFiles;DeinstallStart;InstallationInfo;SdFinishReboot;VerFindFileVersion;VerGetFileVersio  
n;Installing shared files with XCopyFile and  
VerUpdateFile;Installing_self_registering_files_with_XCopyFile_and_VerUpdateFile Installing locked files with  
XCopyFile and VerUpdateFile',0,'')} See also  
{button ,PI('','comment')} Feedback
```



VerUpdateFile example

```
/*-----*\
 *
 * This example illustrates the VerUpdateFile function.
 *
 \*-----*/

#define TDIR "C:\\EXAMPLE"
#define SDIR "C:\\EXAMPLE\\SOURCE"
#define FILE "NOTEPAD.EXE"

STRING szFileName, svInstalledFilePath, szTitle, szMsg;
NUMBER nUpdateFlag, nResult;
BOOL bDone;

program

    TARGETDIR = TDIR;
    SRCDIR = SDIR;

    szFileName = FILE;
    nUpdateFlag = VER_UPDATE_ALWAYS;

    szTitle = "VerUpdateFile Example";
/*-----*\
 *
 * VerUpdateFile will update the file regardless of file version.
 *
 \*-----*/

    nResult = VerUpdateFile(szFileName, nUpdateFlag, svInstalledFilePath);
    sprintfBox(INFORMATION, "", "%d", nResult);

    if (nResult < 0) then
        MessageBox("First call to VerUpdateFile failed.", SEVERE);
    else
        szMsg = "%s successfully updated.";
        sprintfBox(INFORMATION, szTitle, szMsg, szFileName);
    endif;

    nUpdateFlag = VER_UPDATE_COND;

/*-----*\
 *
 * VerUpdateFile will update the files only if the target files are more
 * recent.
 *
 \*-----*/

    nResult = VerUpdateFile(szFileName, nUpdateFlag, svInstalledFilePath);

    if(nResult < 0) then
        MessageBox("Second call to VerUpdateFile failed.", SEVERE);
    endif;

endprogram

// Source file: Is5fn272.rul
```

{button ,PI("",`comment')} Feedback



User interface functions

User interface functions allow you to customize certain error messages and titles of error boxes. However, several internal error messages that may be encountered during the development of the setup cannot be modified using user interface functions.

Disable

Disables the display of a user interface object.

Enable

Enables the display of a user interface object.

FindWindow

Retrieves the handle to a window.

PlaceBitmap

Inserts an image into the installation window.

PlaceWindow

Sets the position of user interface objects.

PlayMMedia

Plays an audio or video file.

RGB

Returns a custom color value based on the specified red, green and blue values.

SetColor

Changes the color of various user-interface elements.

SetDialogTitle

Creates custom dialog box titles.

SetDisplayEffect

Sets the display effect for bitmap and metafile images.

SetErrorMsg

Changes the text of error messages.

SetErrorTitle

Changes the title of error message boxes.

SetFont

Sets the font type and style.

SetStatusWindow

Sets the text for the status window and sets the initial percentage completed value that is displayed in the progress indicator.

SetTitle

Sets the text and color of the title in the main window.

SizeWindow

Specifies the size of most user interface objects.

StatusUpdate

Links the information gauges to the progress indicator and sets the percent complete to display in the progress indicator after the next file transfer operation.

{button ,PI("",`comment')} [Feedback](#)



Disable

Syntax

Disable (nConstant);

Description

The Disable function deactivates the user interface object or setup feature specified by the parameter nConstant.

Parameters

nConstant

Specifies the user interface object or operational feature to disable. Pass one of the following predefined constants in this parameter:

BACKBUTTON

Disables (grays out) the Back button that is displayed in some built-in dialog boxes. The Back button is enabled by default.

BACKGROUND

Disables and hides the setup's main background window. Note that this parameter has no effect when the setup is in full-screen mode.

BITMAP256COLORS

Disables 256-color palette handling so that bitmaps will use only those color palette entries that are currently allocated. For more information on color palette handling, refer to [PlaceBitmap](#). 256-color palette handling is enabled by default.

CORECOMPONENTHANDLING

Disables core component handling. For more information about core component handling, refer to [Enable](#).

DIALOGCACHE

Disables the dialog cache mechanism. For more information about dialog caching, refer to [Enable](#).

FEEDBACK_FULL

Disables and hides the feedback indicator (information gauges), which is displayed automatically when the status bar is enabled. If the feedback indicator is disabled by passing this constant to Disable, that indicator will not be displayed when the status bar is enabled.

HOURGLASS

Causes the mouse cursor to change from the "Busy" cursor (an hourglass by default), to a normal cursor (a pointer by default).

INDVFILESTATUS

Causes individual paths and filenames not to be displayed in the progress indicator (status bar) during file transfer.

LOGGING

Disables the logging of uninstallation information so that no information will be recorded in the uninstallation log file. Note that logging is enabled automatically by the function DeInstallStart. Disabling the logging of uninstallation information before calling DeInstallStart will have no effect since logging will be re-enabled automatically when DeInstallStart is called. You must disable logging manually immediately before performing operations that should not be logged for uninstallation.

NEXTBUTTON

Disables (grays out) the Next button that is displayed in some built-in dialog boxes. The Next button is enabled by default.

SELFREGISTERBATCH

Disables the batch method of registering self-registering files. For more information, refer to [Enable](#) or click [here](#).

STATUS

Disables and hides the standard progress indicator (Status bar). Disabling the progress indicator will also disable the feedback indicator.

STATUSDLG

Disables and hides the dialog style progress indicator (Status bar). Disabling the progress indicator will also disable the feedback indicator.

STATUSOLD

Disables and hides the old style progress indicator (Status bar). Disabling the progress indicator will also disable the feedback indicator.

Return values

0

Indicates that the function successfully disabled the user interface object or setup feature specified by the parameter nConstant.

< 0

Indicates that the function was unable to disable the user interface object or setup feature specified by the parameter nConstant.

Comments

- n If your script calls the Disable function to disable the Next or Back button, that button will be disabled in all dialog boxes displayed after that function call. To re-enable the Next or Back button, you must call Enable with the corresponding constant.
- n DIALOGCACHE has no effect on dialog boxes that do not have a Next or Back button.
- n InstallShield displays the feedback indicator (information gauges) by default when the progress indicator (status bar) is enabled. The feedback indicator is the set of feedback gauges displayed to the left of the progress indicator (status bar). To prevent these gauges from being displayed when the progress indicator is displayed, call Disable with the FEEDBACK_FULL parameter before enabling the progress indicator with the Enable function.

{button ,JI('LANGREF.HLP>Examples','Disable_example')} [Example](#)

{button ,AL('Enable;Prepare my setup for uninstallation;The information gauges',0,'','')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



Disable example

```
/*-----*\
*
* This example illustrates using the Disable and Enable functions.
*
* The first dialog box that appears has the Back button disabled.
*
* After the Next button is pressed, the second dialog box appears. This box
* will have the Next button disabled, and the Back button enabled.
*
*\-----*/

program

start:

    // Back button is disabled.
    Disable(BACKBUTTON);

    // The following displays a dialog box with the Back button disabled.
    SetupType("", "", "", TYPICAL, 0);

    // Enable the Back button,
    Enable(BACKBUTTON);

    // Next button is disabled.
    Disable(NEXTBUTTON);

    // The following displays a dialog box with only the Back button enabled.
    if (SetupType("", "", "", TYPICAL, 0) = BACK) then

        // If the Back button is pressed, the Next button is enabled.
        Enable(NEXTBUTTON);
        goto start;
    endif;

    // The Next and Back buttons stay disabled until Enable is called.
    Enable(NEXTBUTTON);
    Enable(BACKBUTTON);

endprogram

// Source file: Is5fn208.rul
```

{button ,PI("",`comment')} Feedback



Enable

Syntax

Enable (nConstant);

Description

The Enable function activates the user interface object or setup feature specified by the parameter nConstant.

Parameters

nConstant

Specifies the user interface object or operational feature you want to enable. Pass one of the following predefined constants in this parameter:

BACKBUTTON

Enables the Back button that is displayed in some built-in dialog boxes. The Back button is enabled by default, but it can be disabled by calling the Disable function.

BACKGROUND

Displays the setup's main background window when the setup is in window mode. When setup is in full screen mode, which is the default mode, this constant has no effect. To enable window mode, you must call Enable with the constant DEFWINDOWMODE or FULLWINDOWMODE.

BITMAP256COLORS

Activates 256-color palette handling so that bitmaps will allocate and use any available color palette entries automatically. For more information on color palette handling, refer to [PlaceBitmap](#). 256-color palette handling is enabled by default.

CORECOMPONENTHANDLING

Enables core component handling so that in accordance with Windows logo requirements, any files listed in Corecomp.ini will not be logged for uninstallation or reference counted by the setup. Corecomp.ini is added to the setup automatically by the Media Build Wizard. Core component handling is not enabled by default.

Note that core component handling is not supported in 16-bit setups under any platform or in 32-bit setups under Windows NT 3.51. Passing CORECOMPONENTHANDLING to Enable in a 16-bit setup does not produce an error, but it has no effect on the setup. Under Windows NT 3.51, the function will fail when called to enable core component handling; however, the error should be ignored on this platform.

DEFWINDOWMODE

Configures the main background window to be a normal, resizable window with a title bar. If the background window is enabled, its appearance will change immediately. If the background window is not enabled, no change to the screen will occur until Enable is called with the constant BACKGROUND.

DIALOGCACHE

Enables the dialog Cache mechanism, which eliminates the screen flash that appears between the display of dialog boxes. This screen flash is most noticeable in the title bar of setups running in window mode. Note that the dialog caching mechanism works only for dialogs that have BACK and NEXT buttons. Dialog caching is disabled by default.

FEEDBACK_FULL

Displays the feedback indicator (information gauges). These gauges are also displayed automatically when the status bar is enabled; therefore, it is not necessary to enable the gauges manually unless you want to display them while the status bar is not displayed.

FULLWINDOWMODE

Configures the main background window to be a maximized, resizable window with a title bar. If the background window is enabled, its appearance will change immediately. If the background window is not enabled, no change to the screen will occur until Enable is called with the constant BACKGROUND.

HOURLASS

Causes the mouse cursor to change to the "Busy" cursor, an hourglass by default.

INDVFILESTATUS

Enables the display (on the second line of the progress indicator) of the fully qualified filename of each file as it is transferred when ComponentMoveData, CopyFile, or XCopyFile is called and the progress indicator is enabled.

LOGGING

Enables the logging of uninstallation information. When logging is enabled, results of operations that InstallShield logs for uninstallation will be recorded in the uninstallation log file and will be reversed during uninstallation. Calling DeinstallStart automatically enables logging; therefore, you do not need to enable logging manually unless you have previously disabled it with the Disable function.

NEXTBUTTON

Enables the Next button that is displayed by some built-in dialog boxes. The Next button is enabled by default.

SELFREGISTERBATCH

Enables the batch method of registering self-registering files. This method is used to avoid dependency problems related to the self-registering of files. Normal (non-batch) self-registering is the default. For more information, click [here](#).

STATUS

Enables the display of the standard progress indicator (status bar) and, if it has not been disabled, the feedback indicator.

STATUSDLG

Enables the display of the dialog style progress indicator (status bar) and, if it has not been disabled, the feedback indicator.

STATUSOLD

Enables the display of the old style progress indicator (status bar), which does not have a Cancel button. Also enables the feedback indicator if it has not been disabled.

Return values**0**

Indicates that the function successfully enabled the user interface object or setup feature specified by the parameter nConstant.

< 0

Indicates that the function was unable to enable the user interface object or setup feature specified by the parameter nConstant.

CC_ERR_OUTOFMEMORY

Not enough memory was available to enable core component handling. Close any open applications and restart the setup.

CC_ERR_NOCOMPONENTLIST

No entries could be found in Corecomp.ini. Verify that the file Corecomp.ini is valid.

CC_ERR_FILEFORMATERROR

A formatting error was detected in Corecomp.ini. Verify that the file Corecomp.ini is valid.

CC_ERR_FILEREADERROR

InstallShield is unable to read the file Corecomp.ini. Verify that the file is valid.



The file Corecomp.ini can be found in the following folder:
<INSTALLSHIELD LOCATION>\Redistributable\Compressed
Files\Language Independent\OS Independent.

Comments

- n By default, setup runs in full screen mode; the main background window is displayed as a full-screen window without a title bar. In this mode the main installation window cannot be resized and does not have a visible title bar. To enable window mode, you must call Enable with the constant DEFWINDOWMODE or FULLWINDOWMODE

- n If your script calls the Disable function to disable the Next or Back button, that button will be disabled in all dialog boxes displayed after that function call. To re-enable the Next or Back button, you must call Enable with the corresponding constant.
- n DIALOGCACHE has no effect on dialog boxes that do not have a Next or Back button.
- n After calling Enable to initialize core component handling, you should check the return value to verify that core component handling was enabled.
- n Core component handling is not supported in 16-bit setups under any platform or in 32-bit setups under Windows NT 3.51. Passing CORECOMPONENTHANDLING to Enable in a 16-bit setup does not produce an error, but it has no effect on the setup. Under Windows NT 3.51, the function will fail when called to enable core component handling; however, the error should be ignored on this platform.

{button ,JI('LANGREF.HLP>Examples','Enable_example')} [Example](#)

{button ,AL('Disable;PlaceBitmap;Prepare my setup for uninstallation;Install shared files as core components;The information gauges;Displaying the progress indicator;Preventing color distortion;Which graphic formats InstallShield supports',0,'')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



Enable example

```
/*-----*\
*
* This example illustrates using the Disable and Enable functions.
*
* The first dialog box that appears has the Back button disabled.
*
* After the Next button is pressed, the second dialog box appears. This box
* will have the Next button disabled, and the Back button enabled.
*
\*-----*/

program

start:

    // Back button is disabled.
    Disable(BACKBUTTON);

    // The following displays a dialog box with the Back button disabled.
    SetupType("", "", "", TYPICAL, 0);

    // Enable the Back button,
    Enable(BACKBUTTON);

    // Next button is disabled.
    Disable(NEXTBUTTON);

    // The following displays a dialog box with only the Back button enabled.
    if (SetupType("", "", "", TYPICAL, 0) = BACK) then

        // If the Back button is pressed, the Next button is enabled.
        Enable(NEXTBUTTON);
        goto start;
    endif;

    // The Next and Back buttons stay disabled until Enable is called.
    Enable(NEXTBUTTON);
    Enable(BACKBUTTON);

endprogram

// Source file: Is5fn208.rul
```

{button ,PI('','comment')} [Feedback](#)



PlaceBitmap

Syntax

PlaceBitmap (szName, nID_BITMAP, nDx, nDy, nDrawOp);

Description

The PlaceBitmap function inserts an image into the installation window. The image source is specified by szName; it can be a bitmap file (.bmp), metafile (.wmf file), or dynamic link library (.dll). InstallShield supports 2-color, 16-color, 256-color and true color (24-bit) bitmaps. Two-color, 16-color and 256-color bitmaps can have transparent portions.

Transparent bitmaps are useful for displaying images that appear to be integrated with the background window. Pixels in the bitmap that match a specified transparent color are not displayed; the background pixel at that location remains visible. In setups, transparent bitmaps that incorporate the company name and its logo in an artful design are often used as titles in the installation window.

To specify a transparent bitmap, you must pass the constant BITMAPICON in the parameter nDrawOp. You must also consider which color in the bitmap is to be transparent. The default transparent color is purple (RGB(255,0,255)). To specify a different transparent color, use the parameter szName as described below.

Because metafiles are drawn rather than placed, they are intrinsically transparent. If BITMAPICON is specified for a metafile, that parameter is ignored. For more information about metafiles, click [here](#).



Many special display effects are available for non-transparent bitmaps by using the [SetDisplayEffect](#) function. That function also provides limited display effects for metafiles.

The location of the bitmap within the window can be specified in one of two ways:

- n By passing one of the location constants in the parameter nDrawOp.
- n By passing a vertical and horizontal offset from the edge of the installation window in nDx and nDy.

Always remove any bitmaps or metafiles that are no longer needed by calling PlaceBitmap with the constant REMOVE as the parameter nDrawOp. Removing an unneeded bitmap is recommended even if another bitmap covers that bitmap completely because the palette entries for the first bitmap will not be released until the bitmap is removed.



A true color bitmap that is displayed on a 16-bit system or on a system running in 16-color or 256-color mode will use only those colors available in the color palette; no additional colors will be allocated for the bitmap, even when additional color palette entries are available. If you anticipate that a setup with 24-bit bitmaps will be run on 16- or 256-color systems, include 16- or 256-color versions of the bitmaps. Then call [GetSystemInfo](#) with the COLORS parameter to determine the current color mode before selecting the bitmap to display.

Parameters

szName

Specifies the fully qualified name of the bitmap file (.bmp), metafile (.wmf file), or dynamic link library (.dll) of the image to be displayed. InstallShield recognizes bitmap files and metafiles by their file extension. Bitmap files must have the extension .bmp. Metafiles must have the extension .wmf. Dynamic link libraries must have the extension .dll. If a filename is specified with no extension, InstallShield assumes the extension .dll.

To specify an alternate transparent color, place a semicolon after the filename and follow it with a set of RGB color values. (RGB color is specified with three numeric values, separated by commas.) That color is then used as the transparent color for the bitmap specified by szName. Note that it does *not* affect bitmaps that are already displayed; nor does it become the default transparent color for bitmaps displayed by subsequent calls to PlaceBitmap.

The parameter below specifies white as the transparent color:

```
SUPPORTDIR ^ "Bitmap.bmp;255,255,255"
```

When nDrawOptions is set to REMOVE, this parameter is ignored.

nID_BITMAP

Specifies the bitmap's resource ID if the bitmap resides in a .dll. If the bitmap source is a metafile or bitmap file, specifies a value that is not in use for an image currently on display; images that are displayed simultaneously must have unique ID numbers. When nDrawOptions is set to REMOVE, this parameter must contain the ID of a displayed image.

nDx

Specifies the horizontal distance in pixels between the edge of installation window and the edge of the image. When nDrawOp is set to CENTERED, this parameter is ignored.

nDy

Specifies the vertical distance in pixels between the edge of installation window and the edge of the image. When nDrawOp is set to CENTERED, this parameter is ignored.

nDrawOp

Specifies the bitmap's placement location, sets placement options, or removes a previously placed bitmap. Pass one of the following predefined constants in this parameter:

BITMAPICON

Indicates that the bitmap has transparent parts. You can use the bitwise OR operator (|) to combine this constant with any of the other constants except TILED, FULLSCREEN or FULLSCREENSIZE. When BITMAPICON is ORed with one of those constants, the bitwise operation is ignored and BITMAPICON is used.

BITMAPICON has no effect when szName specifies a metafile or a 24-bit bitmap. Note that when you specify BITMAPICON, the bitmap is displayed normally, even if a special effect has been enabled with SetDisplayEffect.

TILED

The bitmap is tiled across the main installation window. This constant is normally used to create an installation background. When this constant is specified, location options are ignored and the bitmap is displayed normally, even if a special effect has been enabled with SetDisplayEffect.

FULLSCREEN

Draw the image to fill the entire installation window. The image is not resized when its drawn. If a bitmap image is smaller than the InstallShield main window, it is centered in the window and the background is filled with the current background color. The default value is teal; it can be changed using the [SetColor](#) function. When this constant is specified, location options are ignored and the bitmap is displayed normally, even if a special effect has been enabled with SetDisplayEffect.

FULLSCREENSIZE

Draw and stretch the image to fill the entire installation window. When this constant is specified, location options are ignored and the bitmap is displayed normally, even if a special effect has been enabled with SetDisplayEffect.

CENTERED

Places the bitmap in the center of the InstallShield installation window.

LOWER_LEFT

Places the bitmap in the lower left corner of the InstallShield installation window.

LOWER_RIGHT

Places the bitmap in the lower right corner of the InstallShield installation window.

UPPER_LEFT

Places the bitmap in the upper left corner of the InstallShield installation window.

UPPER_RIGHT

Places the bitmap in the upper right corner of the InstallShield installation window.

REMOVE

Removes a previously placed bitmap or metafile. Any special display effects that have been enabled with

SetDisplayEffect are ignored.

Return values

0

Indicates that the function successfully found and placed the image.

< 0

Indicates that the function was unable to find or place the image.

Comments

- n Call SetDisplayEffect to set special effects for non-tiled, full-screen, transparent bitmaps; you can also set limited special effects for metafiles.
- n InstallShield does not support 24-bit transparent bitmaps. If you include a transparent color in a 24-bit bitmap and specify the BITMAPICON constant, the color will be displayed normally.
- n When you place a 256-color bitmap on a system running in 256-color mode, InstallShield will attempt to allocate the bitmap's color palette into the system color palette. If multiple 256-color bitmaps are placed, InstallShield will attempt to merge the color palettes of all visible bitmaps into the system color palette, giving precedence to the most recently placed bitmap. This behavior may cause previously placed bitmaps to change colors when additional bitmaps are displayed.

You can prevent InstallShield from merging additional color palette entries for newly-placed bitmaps by calling the [Disable](#) function with the BITMAP256COLORS options. Note that this will cause subsequent bitmap displays to use the previously allocated palette entries only.

- n On a system running in 256-color mode with a 256-color dithered background, bitmaps that include many colors may cause some of the color palette entries used for the background to be reallocated; this can cause a gradient effect to appear in the background. Setups with bitmaps that use many colors should not use a 256-color dithered background if they will run on 256-color systems.
- n System color palettes exist only on systems that are running in 256-color mode. Windows 95 systems running in high color (16-bit) or true color (24-bit) modes or Windows NT systems running in 65535 (16-bit) color mode do not have a system color palette. On these systems there are no color palette handling issues to consider; colors are displayed directly using the RGB color value. For more information about preventing color distortion, click [here](#).
- n Because metafiles are rendered, they do not include a custom color palette. When a metafile is displayed on a 256-color system, no color palette handling takes place; the metafile is drawn with the colors currently available in the color palette. For that reason, you should not use metafiles that display colors other than the standard 16 colors in setups that will run on 256-color systems.

{button ,JI('LANGREF.HLP>Examples','PlaceBitmap_example')} [Example](#)

{button ,AL('Disable;Enable;PlaceWindow;SetDisplayEffect;SetColor;Preventing color distortion;Using metafiles as images;Which graphic formats InstallShield supports;Using transparent bitmaps;Troubleshooting problems with loading bitmaps;Place bitmaps according to target screen resolution;Access bitmaps during setup',0,','')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



PlaceBitmap example

```
/*-----*\
 *
 * This example illustrates the uses of the PlaceBitmap function.
 *
 * PlaceBitmap is called three times to display a bitmap on the screen.
 * The SetDisplayEffect function sets the display effect for the bitmap.
 *
 * NOTE: In order for this script to run properly, it is advised to set the
 *       BMP_PATH constant to a valid bitmap file on the target system.
 *
 \*-----*/

#define BMP_PATH "C:\\EXAMPLE\\SOURCE\\PROGRAM\\DEMO.BMP"

STRING szBitmap;
NUMBER nID_BITMAP, nDx, nDy, nCorner;

program

szBitmap = BMP_PATH;
nID_BITMAP = 12;
nDx = 10;
nDy = 10;
nCorner = UPPER_LEFT;

// Position the first bitmap on the screen.
if (PlaceBitmap(szBitmap, nID_BITMAP, nDx, nDy, nCorner) < 0) then

    MessageBox("First call to PlaceBitmap failed.", SEVERE);
endif;
Delay(3);

// Remove the bitmap with ID of 12.
if (PlaceBitmap("", 12, 0, 0, REMOVE) < 0) then

    MessageBox("Second call to PlaceBitmap failed.", SEVERE);
endif;

// Set bit map fade in effect.
SetDisplayEffect(EFF_FADE);

// Position the second bitmap on the screen.
PlaceBitmap(szBitmap, 13, CENTERED, CENTERED, 0);
Delay(3);

// Remove the bitmap with ID of 13.
PlaceBitmap("", 13, 0, 0, REMOVE);

// Set bit map reveal in effect.
SetDisplayEffect(EFF_REVEAL);

// Position the third bitmap on the screen.
PlaceBitmap(szBitmap, 14, 10, 10, LOWER_RIGHT);
Delay(3);

// Remove the bitmap with ID of 14.
PlaceBitmap("", 14, 0, 0, REMOVE);
Delay(3);
```

endprogram

// Source file: Is5fn122.rul

{button ,PI("",`comment')} Feedback



PlaceWindow

Syntax

PlaceWindow (nObject, nDx, nDy, nCorner);

Description

The PlaceWindow function changes the position of user interface objects, including billboards. Specify the distance between the sides of the object and the edges of the screen in nDx and nDy.

Parameters

nObject

Specifies the object whose position is to be changed. Pass one of the following predefined constants in this parameter:

ASKOPTIONS

Moves the AskOptions dialog box.

ASKPATh

Moves the AskPath dialog box.

ASKTEXT

Moves the AskText dialog box.

BACKGROUND

Moves the background window.

BILLBOARD

Sets the location of billboards used during the file transfer process.

ENTERDISK

Moves the EnterDisk dialog box.

FEEDBACK

Moves the Feedback control.

MMEDIA_AVI

Sets the window position for next AVI file to be played. By default, the AVI file will be played in a window in the left hand corner of the InstallShield client screen, 10 pixels from the left and 10 pixels from the top.

STATUS

Moves the progress indicator.

STATUSDLG

Moves the dialog style progress indicator.



If you call Enable (STATUS) in your script, make sure you use the constant STATUS. If you call Enable (STATUSDLG), make sure you use the constant STATUSDLG.

nDx

Specifies the distance in pixels between the appropriate edge of the object and the edge of the screen on the horizontal axis. When passing FEEDBACK as the parameter nObject, you can use CENTERED to center the information gauge horizontally. (All other objects represented by nObject options are centered by default.)

nDy

Specifies the distance in pixels between the appropriate edge of the object and the edge of the screen on vertical axis. When passing FEEDBACK as the parameter nObject, you can use CENTERED to center the information gauge vertically. (All other objects represented by nObject options are centered by default.)

nCorner

Specifies from which corner to measure the distances, expressed in nDx and nDy. Pass one of the following

predefined constants in this parameter:

LOWER_LEFT

Measures nDx from the left and nDy from the bottom of the main InstallShield window.

LOWER_RIGHT

Measures the nDx from the right and nDy from the bottom of the main InstallShield window.

UPPER_LEFT

Measures the nDx from the left and nDy from the top of the main InstallShield window.

UPPER_RIGHT

Measures the nDx from the right and nDy from the top of the main InstallShield window.

CENTERED

Centers the object in the main InstallShield window. For use when passing FEEDBACK as the parameter nObject. (All other objects represented by nObject options are centered by default.) Causes the information gauge to be centered in the main window.

Return values

0

Indicates that the function successfully changed the position of the object.

< 0

Indicates that the function was unable to change the position of the object.

Comments

- n When using this function be aware that a setup program runs on a variety of screen resolutions. You may want to determine the extents of the screen before you position the objects. The distance is measured in pixels and is between the edge of the object and the edge of the corner of the screen specified.
- n You cannot position message boxes with this function because they are created using the native Windows/PM API. A message box's position is determined by Windows/PM and is not under the control of InstallShield. In addition, you cannot position a custom dialog box with this function.
- n To center the information gauge, call PlaceWindow as shown below.

```
PlaceWindow(FEEDBACK, CENTERED, CENTERED, CENTERED);
```
- n You must also call PlaceWindow to move the progress indicator (passing STATUS or STATUSDLG as the parameter nObject) because the progress indicator is centered by default, and it will obscure the centered information gauge.

{button ,JI('LANGREF.HLP>Examples','PlaceWindow_example')} [Example](#)

{button ,AL(' Enable;GetExtents;Displaying billboards;Name my billboards;Showing video during setup;Playing sound during setup;Place video on the main window',0,','')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



PlaceWindow example

```
/*-----*\
 *
 * This example illustrates the uses of the PlaceWindow function.
 *
 * PlaceWindow is called to place the background window 50 pixels to the
 * right and below the upper left-hand corner of the display.
 *
 \*-----*/

program

/*-----*\
 *
 * The following changes the position of the background window.
 *
 \*-----*/
    if (PlaceWindow(BACKGROUND, 50, 50, UPPER_LEFT) < 0) then
        MessageBox("PlaceWindow failed.", SEVERE);
    else
        MessageBox("This is the new position of the background window.",
            INFORMATION);

        Delay(3); // Delay exiting script for three seconds.
    endif;

endprogram

// Source file: Is5fn123.rul
```

{button ,PI("",`comment')} [Feedback](#)



RGB

Syntax

RGB (constRed, constGreen, constBlue);

Description

The RGB function creates a custom color value that can be used with [SetColor](#) and [SetTitle](#).

Parameters

constRed

Specifies a numeric constant, ranging in value from 0..255, indicates the amount of red in the custom color.

constGreen

Specifies a numeric constant, ranging in value from 0..255, indicates the amount of green in the custom color.

constBlue

Specifies a numeric constant, ranging in value from 0..255, indicates the amount of blue in the custom color.

Return values

This function returns a number value for a custom color that can be used when calling SetColor and SetTitle.

{button ,JI('LANGREF.HLP>Examples','RGB_example')} [Example](#)

{button ,AL('SetColor;SetTitle',0,',')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



RGB example

```
/*-----*\
 *
 * This example illustrates the use of the RGB function.
 *
 * The first call to RGB returns the value for the background color to Grey; the
 * second call sets the background color to red; the third call sets the
 * background color to blue.
 *
 \*-----*/

program

    // Change the background color using RGB value.
    SetColor(BACKGROUND, RGB(198,198,198));

    Delay(3);

    SetColor(BACKGROUND, RGB(255,0,0));

    Delay(3);

    SetColor(BACKGROUND, RGB(0, 0, 255));

    Delay(3);

endprogram

// Source file: Is5fn605.rul
```

{button ,PI("",`comment')} [Feedback](#)



SetColor

Syntax

SetColor (nObject, nColor);

Description

The SetColor function sets the color of two user interface objects.

Parameters

nObject

Specifies which user interface object to change. Pass one of the following predefined constants in this parameter:

BACKGROUND

Indicates the background of the InstallShield main window. The default color is solid teal: RGB (0,128,128).

STATUSBAR

Indicates the color of the progress bar in the progress indicator. The default color is BLUE.

nColor

Specifies the color to apply to the user interface objects. Pass the function RGB or pass one of the following predefined constants in this parameter.

n For background color, pass one of the following constants:

BK_BLUE

Produces a gradient blue background color.

BK_GREEN

Produces a gradient green background color.

BK_MAGENTA

Produces a gradient magenta background color.

BK_RED

Produces a gradient red background color.

BK_YELLOW

Produces a gradient yellow background color.

BK_SOLIDBLUE

Produces a solid blue color background.

BK_SOLIDGREEN

Produces a solid green color background.

BK_SOLIDMAGENTA

Produces a solid magenta color background.

BK_SOLIDRED

Produces a solid red color background.

BK_SOLIDYELLOW

Produces a solid yellow color background.

BK_SMOOTH

Can be bitwise ORed with any custom color to achieve a smoothing effect (gradient) when the background is painted. It should be noted that the smoothing effect will be better with 256 colors enabled.

n For the progress bar , pass one of the following constants:

GREEN

Produces a green progress bar.

RED

Produces a red progress bar.

BLUE

Produces a blue progress bar.

MAGENTA

Produces a magenta progress bar.

YELLOW

Produces a yellow progress bar.

Return values

0

Indicates that the function successfully set the color of the object.

< 0

Indicates that the function was unable to set the color of the user interface object.

Comments

When using an RGB value, you can apply the same method as described in the programming manuals for Microsoft Windows. You specify the mixture of RED, GREEN, and BLUE colors to "mix" the color. Use a number from 0 to 255 to represent the amount of color to use. You must use literal numbers in the parameters of the RGB macro. You can also use a long value that represents the RGB color instead of the RGB statement.

{button ,JI('LANGREF.HLP>Examples','SetColor_example')} [Example](#)

{button ,AL('Disable;Enable;Displaying the progress indicator;Preventing color distortion',0,','')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



SetColor example

```
/*-----*\
 *
 * This example illustrates the use of the SetColor function.
 *
 * The first call to SetColor sets the background color to solid blue; the
 * second call sets the background color to red; the third call sets the
 * background color to an RGB value.
 *
 \*-----*/

program

    // Change the background color and delay for three seconds.
    if (SetColor(BACKGROUND, BK_SOLIDBLUE) < 0) then
        MessageBox("SetColor failed.", SEVERE);
    endif;

    Delay(3);

    if (SetColor(BACKGROUND, BK_RED) < 0) then
        MessageBox("SetColor failed.", SEVERE);
    endif;

    Delay(3);

    if (SetColor(BACKGROUND, RGB(100, 50, 150)) < 0) then
        MessageBox("SetColor failed.", SEVERE);
    endif;

    Delay(3);

endprogram

// Source file: Is5fn163.rul
```

{button ,PI("",`comment')} [Feedback](#)



SetDialogTitle

Syntax

```
SetDialogTitle (nDialogId, szTitle);
```

Description

The SetDialogTitle function changes the titles that appear in the title bars of some common built-in dialog boxes. Specify the dialog box using the parameter nDialogId. If you do not use SetDialogTitle, InstallShield displays the default title. Once you set the title for a particular dialog box, InstallShield uses that title for every instance of that type of dialog until you use SetDialogTitle to change the title again.

Parameters

nDialogId

Identifies the built-in dialog box whose title is to be changed. Pass one of the following predefined constants in this parameter:

DLG_ASK_OPTIONS

Changes the title of the AskOptions dialog box.

DLG_ASK_PATH

Changes the title of the AskPath dialog box.

DLG_ASK_TEXT

Changes the title of the AskText dialog box.

DLG_ASK_YESNO

Changes the title of the AskYesNo dialog box.

DLG_ENTER_DISK

Changes the title of the EnterDisk dialog box.

DLG_MSG_INFORMATION

Changes the title of the Information-style MessageBox.

DLG_MSG_SEVERE

Changes the title of the Severe-style MessageBox.

DLG_STATUS

Changes the title of the dialog-style progress indicator. You must re-enable the dialog-style progress indicator by calling Enable(STATUSDLG) after calling SetDialogTitle with the DLG_STATUS option in order for the title change to take effect.

DLG_MSG_WARNING

Changes the title of the Warning-style MessageBox.

DLG_USER_CAPTION

Changes the MessageBox caption when you use the user-defined message box styles.

szTitle

Specifies the new title.

Return values

0

Indicates that the function successfully changed the title of the dialog box.

< 0

Indicates that the function was unable to change the title of the dialog box.

Comments

- n You must call SetDialogTitle separately for each dialog box title you wish to change.
- n InstallShield creates message boxes using standard Windows message box functions. Windows determines the OK and Cancel button text for these message boxes. InstallShield has no control over the text used in the buttons inside Windows message boxes.

{button ,JI('LANGREF.HLP>Examples','SetDialogTitle_example')} [Example](#)

{button ,AL('AskOptions;AskPath;AskText;AskYesNo;MessageBox;EnterDisk;Displaying the progress indicator',0,'')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



SetDialogTitle example

The sample code below displays the AskOptions dialog box with the title "My Title."

```
/*-----*\
 *
 * This example illustrates the SetDialogTitle function.
 *
 * SetDialogTitle is called to change the title of the AskOptions dialog box.
 *
 \*-----*/

    STRING szTitle;
    NUMBER nDialogId, nCheck1, nCheck2;

program

    nDialogId = DLG_ASK_YESNO;
    szTitle   = "SetDialogTitle Example";
    if (SetDialogTitle (nDialogId, szTitle) < 0) then
        MessageBox("SetDialogTitle failed.", SEVERE);
    else
        AskYesNo("Did SetDialogTitle change this title?", YES);
    endif;

endprogram

// Source file: Is5fn253.rul
```

{button ,PI("",`comment')} [Feedback](#)



SetErrorMsg

Syntax

SetErrorMsg (nErrorID, szText);

Description

The SetErrorMsg function customizes InstallShield default error messages. You can use this function to specify the caption text of the message boxes in which these error messages are displayed.

Parameters

nErrorID

Specifies which error message to replace. Pass one of the following predefined constants in this parameter:

ERR_BOX_BADPATH

This message appears when EnterDisk detects a bad path entered by the user.

ERR_BOX_BADTAGFILE

This message appears when EnterDisk detects the specified tag file does not exist on the disk.

ERR_BOX_DISKID

This message appears when EnterDisk detects the drive specified by the user does not exist.

ERR_BOX_DRIVEOPEN

This message appears when EnterDisk detects the disk drive door is open.

szText

Specifies the error message to display in the message box.

Return values

0

Indicates that the function successfully changed the error message.

< 0

Indicates that the function was unable to change the error message.

{button ,JI('LANGREF.HLP>Examples','SetErrorMsg_example')} [Example](#)

{button ,AL('EnterDisk;SetErrorTitle',0,','')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



SetErrorMsg example

```
/*-----*\
 *
 * This example illustrates the usage of the SetErrorMsg function.
 *
 \*-----*/

    STRING szText;
    NUMBER nErrorID;

program

    nErrorID = ERR_BOX_DRIVEOPEN;
    szText   = "You have specified an open drive door.";
/*-----*\
 *
 * The following sets the messages used for the error boxes of the EnterDisk
 * function.
 *
 \*-----*/

    SetErrorMsg(nErrorID, szText);

    nErrorID = ERR_BOX_BADPATH;
    szText   = "You have specified a bad path.";

    SetErrorMsg(nErrorID, szText);

    nErrorID = ERR_BOX_BADTAGFILE;
    szText   = "You have specified a bad tag file.";

    SetErrorMsg(nErrorID, szText);

    nErrorID = ERR_BOX_DISKID;
    szText   = "You have specified a drive which does not exist.";

    SetErrorMsg(nErrorID, szText);

    // Prompt the user to enter a disk to test the messages.
    EnterDisk("Please enter any disk:", "EXAMPLE.EXE");

endprogram

// Source file: Is5fn164.rul
```

{button ,PI('','comment')} [Feedback](#)



SetErrorTitle

Syntax

```
SetErrorTitle (nErrorID, szText);
```

Description

The SetErrorTitle function specifies custom text for the caption bar of InstallShield's built-in error message boxes. You can use the [SetErrorMsg](#) function to customize the error message text.

Parameters

nErrorID

Specifies the error message box whose title is to be replaced. Pass one of the following predefined constants in this parameter:

ERR_BOX_BADPATH

This message appears when [EnterDisk](#) detects a bad path.

ERR_BOX_BADTAGFILE

This message box appears when EnterDisk detects that the specified tag file does not exist on the disk.

ERR_BOX_DISKID

This message box appears when EnterDisk detects that the specified drive does not exist.

ERR_BOX_DRIVEOPEN

This message box appears when EnterDisk detects the disk drive door is open.

szText

Specifies the error message to display in the error message box.

Return values

0

Indicates that the function successfully changes the text in the caption bar.

< 0

Indicates that the function was unable to change the text in the caption bar.

{button ,JI('LANGREF.HLP>Examples','SetErrorTitle_example')} [Example](#)

{button ,AL('SetErrorMsg',0,'')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



SetErrorTitle example

```
/*-----*\
 *
 * This example illustrates using the SetErrorTitle function.
 *
 \*-----*/

    STRING szText;
    NUMBER nErrorID;

program

    szText = "Drive Open Error";
    nErrorID = ERR_BOX_DRIVEOPEN;

/*-----*\
 *
 * Set the titles of the error boxes used for 'EnterDisk'
 *
 \*-----*/

    SetErrorTitle(nErrorID, szText);

    szText = "Specified drive doesn't exist";
    nErrorID = ERR_BOX_DISKID;

    SetErrorTitle(nErrorID, szText);

    szText = "Tag file doesn't exist";
    nErrorID = ERR_BOX_BADTAGFILE;

    SetErrorTitle(nErrorID, szText);

    szText = "Bad path error";
    nErrorID = ERR_BOX_BADPATH;

    SetErrorTitle(nErrorID, szText);

    // Prompt the user to enter a disk to test the messages.
    EnterDisk("Please insert the disk labeled 'System Disk' ", "MyApp.exe");

endprogram

// Source file: Is5fn165.rul
```

{button ,PI(,"",`comment')} [Feedback](#)



SetFont

Syntax

```
SetFont (nItemID, nFontStyle, szFontName);
```

Description

The SetFont function sets the font and style when displaying text strings. You may use standard Windows fonts with this function.

Parameters

nItemID

Specifies the item whose font and font style is to be set. Pass one of the following predefined constants in this parameter:

FONT_TITLE

Specifies the main title of the setup procedure, which is displayed in the top left corner of the setup window.

nFontStyle

Specifies font styles. Pass one or more of the following predefined constants in this parameter. All but STYLE_NORMAL can be bitwise ORed to specify multiple styles.

STYLE_NORMAL

No bold, italic, or shadow (cannot be ORed)

STYLE_BOLD

Bold Font

STYLE_ITALIC

Italic Font

STYLE_SHADOW

Font with Shadow drawn

STYLE_UNDERLINE

Underlined Font

szFontName

Specifies the name of an available Windows font. Valid font names include Courier, Helv, Helvetica, Modern, Roman, Script, Terminal, Times, and TmsRmn. If the specified font is not found with the styles specified, Arial is used.

Return values

0

Indicates that the function successfully set the font.

< 0

Indicates that the function was unable to set the font.

{button ,JI('LANGREF.HLP>Examples','SetFont_example')} [Example](#)

{button ,AL('SetTitle',0,',')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



SetFont example

```
/*-----*\
 *
 * This example illustrates the usage of the SetFont function.
 *
 * SetFont is called to change the font and style of the title in the
 * InstallShield main window. SetFont is called first to set the title
 * display in 24-point red Helvetica. After a three-second pause, the same
 * title is displayed in 48-point Terminal font. After another three-second
 * delay, the title is redisplayed in 72-point Script.
 *
\*-----*/

    NUMBER nItemID;
    STRING szFont;

program

    nItemID = FONT_TITLE;
    szFont = "Helv";

/*-----*\
 *
 * Set font style to 24-point red Helvetica.
 *
\*-----*/

    if (SetFont(nItemID, 0, szFont) < 0) then
        MessageBox("SetFont failed.", SEVERE);
    endif;

    SetTitle("SetFont Example 1", 24, RGB(255, 0, 0));
    Delay(3);

    nItemID = FONT_TITLE;
    szFont = "Terminal";

/*-----*\
 *
 * Set font style to 48-point yellow Terminal.
 *
\*-----*/

    if (SetFont(nItemID, 0, szFont) < 0) then
        MessageBox("SetFont failed.", SEVERE);
    endif;

    SetTitle("SetFont Example 2", 48, RGB(255, 0, 0));
    Delay(3);

    nItemID = FONT_TITLE;
    szFont = "Script";

/*-----*\
 *
 * Set font style to 72-point black Script.
 *
\*-----*/

    if (SetFont(nItemID, 0, szFont) < 0) then
```

```
    MessageBox("SetFont failed.", SEVERE);
endif;

SetTitle("SetFont Example 3", 72, RGB(255, 0, 0));
Delay(3);

endprogram

// Source file: Is5fn166.rul
```

{button ,PI("",`comment')} [Feedback](#)



SetStatusWindow

Syntax

SetStatusWindow (nPercent, szString);

Description

The SetStatusWindow function sets an initial or current value for the percentage complete indicator of the progress indicator (Status bar) and specifies the current message to display on the top line of the progress indicator (Status bar).

Setups that install files using the ComponentMoveData function must call SetStatusWindow before calling ComponentMoveData in order to set the percentage complete indicator to 0% and clear the top line of the indicator. No additional calls to SetStatusWindow are required. The 'Status Text' string for each component will be displayed on the top line of the status bar automatically while the files of the component are being installed.

Before calling ComponentMoveData, your setup should also call the StatusUpdate function to enable automatic updating of the percentage complete indicator during file transfer. To enable the display of the name and path of the file being installed on the second line of the status bar, call Enable with the INDIVFILESTATUS parameter before calling ComponentMoveData. After these calls, the percentage complete indicator will be updated smoothly during file transfer and the names of individual files will be displayed as they are transferred when the ComponentMoveData function is called.

Setups that install files using the CopyFile or XCopyFile functions may need to make multiple calls to SetStatusWindow in order to change the message on the top line of the indicator between successive calls to CopyFile or XCopyFile. Note that StatusUpdate and Enable (with the parameter INDIVIDUALFILESTATUS) work normally with CopyFile and XCopyFile. Generally, it is not necessary to call SetStatusWindow to change the percentage complete indicator between successive calls to CopyFile or XCopyFile if you have called StatusUpdate to enable automatic updating.

Parameters

nPercent

Specifies a value between 0 and 100 that represents the percentage to be displayed by the percentage complete indicator. To reset the indicator before calling ComponentMoveData, enter 0. To change the message displayed on the top line of the status bar without changing the percentage complete indicator, enter -1 in this parameter.

szString

Specifies a string to display on the top line of the status bar. Note that if a 'DisplayText' parameter has been specified for this component (in the IDE), that string will automatically overwrite any text specified in this parameter when ComponentMoveData is called.

Return values

This function does not return a value.

{button ,JI('LANGREF.HLP>Examples','SetStatusWindow_example')} [Example](#)

{button ,AL('Disable;Enable;StatusUpdate;The information gauges;Displaying the progress indicator',0,',')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



SetStatusWindow example

```
/*-----*\
 *
 * This example illustrates the uses of SetStatusWindow.
 *
 * SetStatusWindow is called to set the progress bar and
 * to display text in the progress indicator.
 *
\*-----*/

#define SOURCE_DIR "C:\\EXAMPLE\\SOURCE"
#define TARGET_DIR "C:\\EXAMPLE\\TARGET"
#define TARGET_FILE "SOMEFILE.TXT"

program

    // Set up source and target directories
    SRCDIR = SOURCE_DIR;
    TARGETDIR = TARGET_DIR;

    // Enable progress indicator.
    Enable(STATUS);

/*-----*\
 *
 * Set the progress bar to 33% and display a message.
 *
\*-----*/
    SetStatusWindow(33, "Now copying file...");

    // Delay for two seconds to assure the window is displayed correctly.
    Delay(2);

    // Copy the file in the source directory to the target directory
    CopyFile(TARGET_FILE, TARGET_FILE);

/*-----*\
 *
 * Set the progress bar to 66% and display a message.
 *
\*-----*/
    SetStatusWindow(66, "Now deleting file...");
    Delay(2);

    // Delete copied file in the Target directory.
    DeleteFile(TARGET_FILE);

/*-----*\
 *
 * Set the percent indicator to 100% and display a message.
 *
\*-----*/
    SetStatusWindow(100, "SetStatusWindow example completed.");
    Delay(2);

endprogram

// Source file: Is5fn255.rul
```

{button ,PI("",`comment')} Feedback



SetTitle

Syntax

```
SetTitle (szString, nPointSize, nColor);
```

Description

The SetTitle function displays a title in large text in the top-left corner of the main window. You can also use the SetTitle function to set the caption on the title bar of the main window. The default title of the main window is "Setup."

Parameters

szString

Specifies a title of the main window. This title must be a single line of text, no more than 80 characters in length.

nPointSize

Specifies the size, in points, of the title. The recommended size is 28 points.

nColor

Specifies the color of the title. Pass one of the following predefined constants in this parameter: RED, GREEN, BLUE, MAGENTA, YELLOW, and WHITE. You can also use RGB() macro to specify a custom color. The following example uses the custom color (with R=78, G=125 and B=161), to display the title text.

```
SetTitle("This is a CUSTOM color title", 32, RGB(78, 125, 161));
```

To set a caption for the title bar of the main window when full window mode is enabled, pass the constant BACKGROUNDCAPTION in this parameter, as shown in the example below. Note that when you use this option, the value of nPointSize is ignored.

```
Enable (FULLWINDOWMODE);  
Enable (BACKGROUND);  
SetTitle("Title bar caption", 0, BACKGROUNDCAPTION);
```

Return values

0

Indicates that the function successfully set the title of the setup.

< 0

Indicates that the function was unable to set the title of the setup.

Comments

If your setup plays an AVI file while displaying a setup title (created with the [SetTitle](#) function), call [Disable](#) with the BITMAP256COLORS parameter to disable 256-color palette handling before you call PlayMMedia to play the AVI file. Disabling 256-color palette handling will not affect the display of the AVI file, but it will prevent color shifting in the setup title on some systems. After playing the AVI file, call [Enable](#) to re-enable 256-color palette handling so that 256-color bitmaps are displayed properly for the remainder of the setup.

{button ,JI('LANGREF.HLP>Examples','SetTitle_example')} [Example](#)

{button ,AL('Enable',0,'')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



SetTitle example

```
/*-----*\
 *
 * This example illustrates the SetTitle function.
 *
 \*-----*/

program

/*-----*\
 *
 * The following displays the title string in yellow, 24-point type. The \n
 * escape character represents a carriage return.
 *
 \*-----*/

    if(SetTitle("SetTitle\nExample", 24, YELLOW) < 0) then
        MessageBox("SetTitle failed.", SEVERE);
    endif;

    Delay (3);

endprogram

// Source file: Is5fn257.rul
```

{button ,PI("",`comment')} Feedback



SizeWindow

Syntax

SizeWindow (nObject, nDx, nDy);

Description

Use the SizeWindow function to change the size of a specific user interface element. Specify the new size in pixels.

Parameters

nObject

Specifies the object to resize. Pass one of the following predefined constants in this parameter:

BACKGROUND

Indicates the main background window.

METAFILE

Indicates a billboard used during the file transfer process. SizeWindow does not support bitmap (.bmp) files.



This parameter does not have any effect on metafiles that are displayed with [SdBitmap](#) function. SdBitmap automatically resizes metafiles when they are displayed.

MMEDIA_AVI

Sets the window size for next AVI file to be played. All the AVI videos are displayed with a default size. Changing the size may change the effective resolution and crispness of the video.

nDx

Specifies the horizontal size of the object in pixels.

nDy

Specifies the vertical size of the object in pixels.

Return values

0

Indicates that the function successfully sized the window.

< 0

Indicates that the function was unable to size the window.

Comments

n The setup may run under many different screen resolutions. In order to account for this, you will need to use the [GetExtents](#) function to determine the overall size of the screen, then use ratios in the parameters of your SizeWindow function call to specify the size of your user interface object.

n This function is recommended for advanced developers only.

{button ,JI('LANGREF.HLP>Examples','SizeWindow_example')} [Example](#)

{button ,AL('GetExtents;Using metafiles as images;Which graphic formats InstallShield supports;Displaying billboards;Name my billboards;Showing video during setup;Playing sound during setup;Place video on the main window',0,','')} [See also](#)

{button ,PI('','comment')} [Feedback](#)



SizeWindow example

```
/*-----*\
 *
 * This example illustrates the usage of the SizeWindow function.
 *
 * GetExtents is called to retrieve the extents of the screen. SizeWindow is
 * then called to resize the background to one-half the original size.
 *
 \*-----*/

    NUMBER nDx, nDy, nObject;

program

    // Enable the background.
    Enable (BACKGROUND);

    MessageBox("Background at original size.", INFORMATION);
    Delay(3);

    // Get the extents of the screen.
    GetExtents (nDx, nDy);

    // Set the object to be resized.
    nObject = BACKGROUND;

    // Divide the x and the y dimensions by two
    nDx = nDx/2;
    nDy = nDy/2;

/*-----*\
 *
 * Resize the background window to half of its original size.
 *
 \*-----*/

    if (SizeWindow(nObject, nDx, nDy) < 0) then
        MessageBox("SizeWindow failed.", SEVERE);
    endif;

    MessageBox("Background after SizeWindow.", INFORMATION);
    Delay(3);

endprogram

// Source file: Is5fn169.rul
```

{button ,PI("",`comment')} [Feedback](#)



StatusUpdate

Syntax

StatusUpdate (bLink, nFinalPercent);

Description

The StatusUpdate function enables or disables the link between file transfer operations and the progress indicator of the status bar. When bLink is ON, the link is enabled and nFinalPercent specifies a final percentage to be displayed at the end of the next file transfer. During that file transfer, the status bar is updated smoothly from its current value to the value specified by nFinalPercent. When bLink is OFF, the link is disabled and the progress indicator of the status bar is not updated automatically during subsequent file transfers.



If the status bar will be enabled during file transfer, call StatusUpdate before each call to [CopyFile](#) or [XCopyFile](#)). Before calling [ComponentMoveData](#) to transfer files, call StatusUpdate with the parameters ON and 100; that will set the status bar to update smoothly to 100% during the file transfer stage of the setup.

Parameters

bLink

Specifies whether to enable or disable the link between file transfer operations and the progress indicator of the status bar. Pass one of the following predefined constants in this parameter:

ON

Specifies that the progress indicator of the status bar should be linked to file transfer operations.

OFF

Specifies that link between file transfer operations and the progress indicator of the status bar should be disabled. The link remains disabled until it is reestablished by a subsequent call to StatusUpdate with bLink set to ON.



The status bar can be updated manually by calling [SetStatusWindow](#).

nFinalPercent

Specifies the final percentage value that the progress indicator of the status bar should reach at the end of the next file transfer operation if bLink is ON. If the value passed in nFinalPercent is less than the current value in the progress indicator of the status bar, the progress indicator will not change. When bLink is OFF, this parameter is ignored.

Return values

0

Indicates that the function was successful.

< 0

Indicates that the function was not successful.

Comments

- ⓘ This function works by computing the total number of bytes to be transferred by any of the file transfer functions. It then computes how often it will increment the progress bar starting from its current location to the maximum value in nFinalPercent.
- ⓘ The StatusUpdate function does not work with the functions [VerUpdateFile](#) and [VerSearchAndUpdateFile](#). When calling those functions, you should disable the status bar or update it manually.

n To set the status bar to an initial percentage, call `SetStatusWindow` before calling `StatusUpdate`.

`{button ,JI('LANGREF.HLP>Examples','StatusUpdate_example')}` [Example](#)

`{button ,AL('SetStatusWindow;Enable;Disable;The information gauges;Displaying the progress indicator',0,'')}`
[See also](#)

`{button ,PI('','comment')}` [Feedback](#)



StatusUpdate example

```
/*-----*\
 *
 * The following examples illustrate the StatusUpdate function.
 *
 * StatusUpdate is called to set the limit of the progress bar, and to
 * regulate the progress indicator.
 *
\*-----*/

#define SOURCE_DIR "C:\\EXAMPLE"
#define TARGET_DIR "C:\\EXAMPLE\\TARGET";

program

    // Enable the progress bar.
    Enable(STATUS);

    SRCDIR = SOURCE_DIR;
    TARGETDIR = TARGET_DIR;

/*-----*\
 *
 * StatusUpdate sets the limit of the progress bar to 99% completion.
 *
\*-----*/

    StatusUpdate(ON, 99);

    if (XCopyFile("*.*", "*.*", COMP_NORMAL) < 0 ) then
        MessageBox("ERROR in Copying Files", SEVERE);
    endif;

    // The following displays a message without affecting the progress bar.
    SetStatusWindow(-1, "File Copying completed at 99%.");
    Delay(3);

    // The following sets the progress bar to 100% and displays a message.
    SetStatusWindow(100, "StatusUpdate example completed, now exiting...");
    Delay(3);

endprogram

// Source file: Is5fn260.rul
```

{button ,PI("", 'comment')} Feedback



Undocumented InstallScript elements

This InstallScript feature is supported in InstallShield 5.0 for backward compatibility. However, because it may not be supported in future versions of InstallShield, InstallShield Corporation does not recommend its use.

`{button ,PI("",`comment')}_Feedback`

Help could not run Dialog Sampler

InstallShield Help found the Dialog Sampler but could not run it.

Dialog Sampler not found

InstallShield Help could not find the Dialog Sampler. You may not have installed this component; check to see if there is a Dialog Sampler folder in your InstallShield program folder. The Dialog Sampler is not available in the Visual C++ Edition.

Filter_language_dependent_files_based_on_the_target_system_s_language

! Back(); ExecFile("Winhelp.exe", '-I Install_shared_files_as_core_components -W (w95sec) GetRes.hlp') **Install
shared files as core components**

t

