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

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:

; (semicolon)
abort
begin
BOOL
BYREF
case
CHAR
default
downto
else
elseif
end
endfor
endif
endprogram
endswitch
endwhile
exit
for
function
goto
HWND
if
INT
LIST
LONG
NUMBER
POINTER
program
prototype
repeat
return
SHORT
step
STRING
switch
then
to
typedef
until
while

Dialog styles

Four different dialog box styles are available:

{button ,JI(`LangRef.hlp>Dialog',`CHECKBOX_dialog_box_style')} CHECKBOX

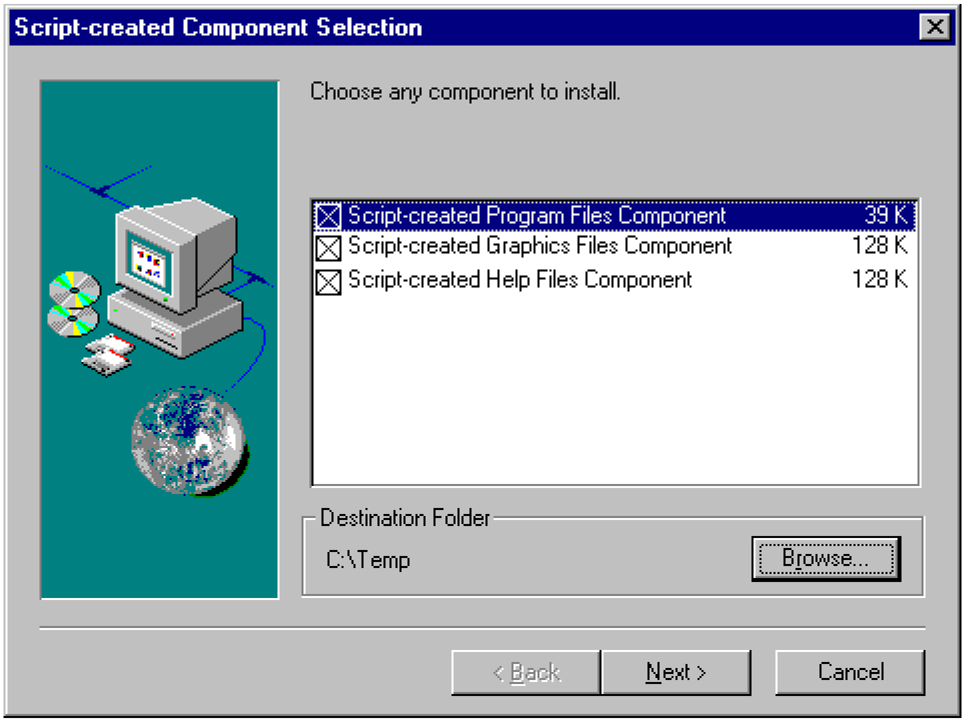
{button ,JI(`LangRef.hlp>Dialog',`CHECKBOX95_dialog_box_style')} CHECKBOX95

{button ,JI(`LangRef.hlp>Dialog',`CHECKMARK_dialog_box_style')} CHECKMARK

{button ,JI(`LangRef.hlp>Dialog',`CHECKLINE_dialog_box_style')} CHECKLINE

InstallShield
www.installshield.com

CHECKBOX dialog box style



Script-created Component Selection

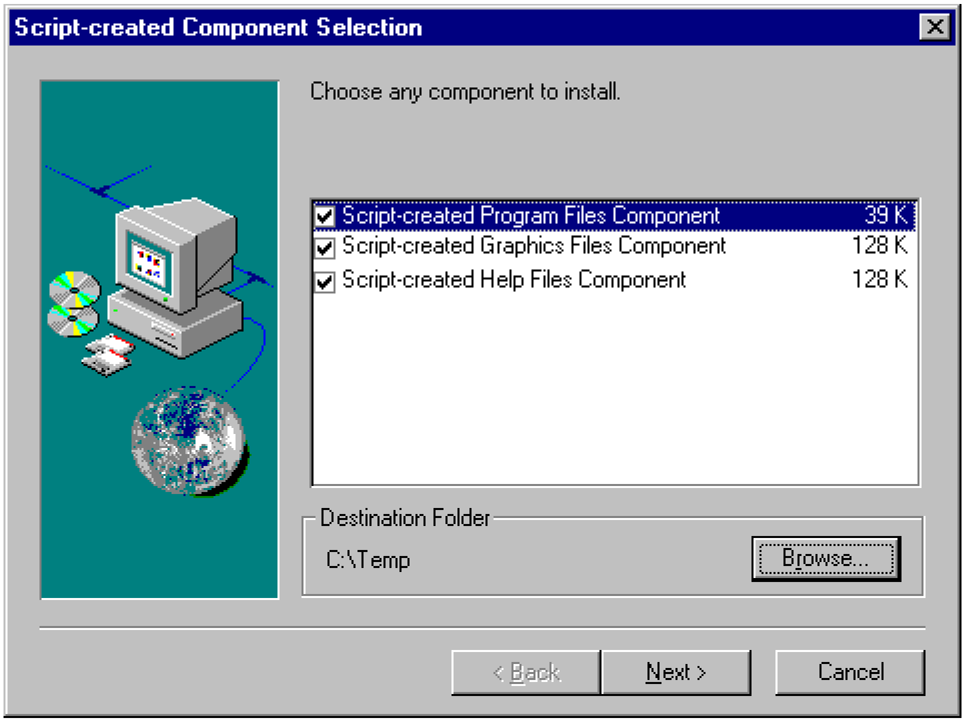
Choose any component to install.

☒	Script-created Program Files Component	39 K
☒	Script-created Graphics Files Component	128 K
☒	Script-created Help Files Component	128 K

Destination Folder

C:\Temp

CHECKBOX95 dialog box style



Script-created Component Selection

Choose any component to install.

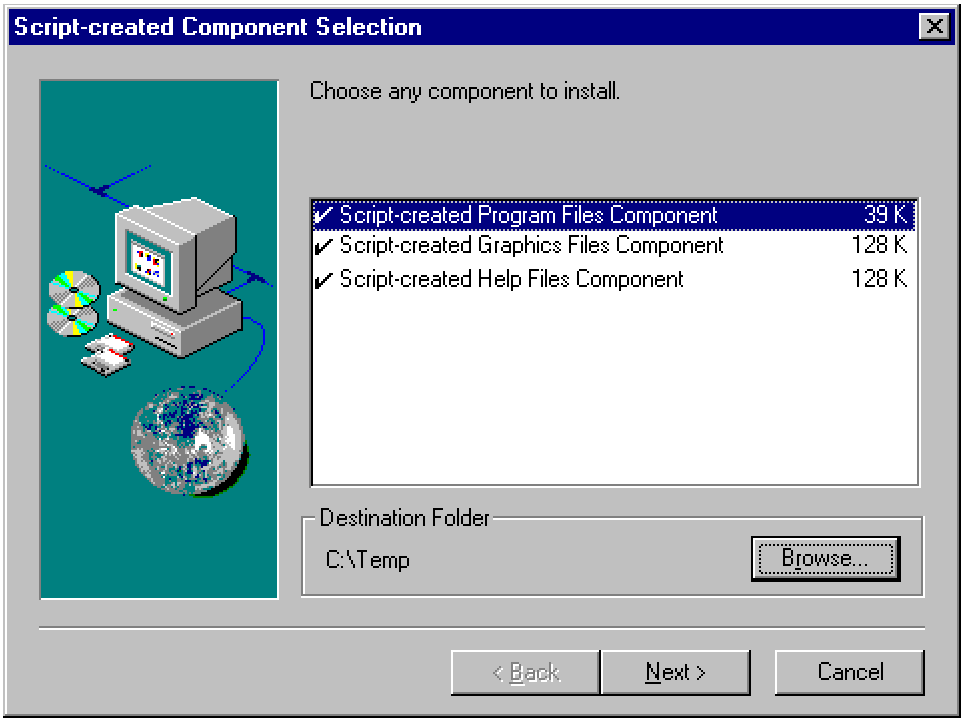
☑	Script-created Program Files Component	39 K
☑	Script-created Graphics Files Component	128 K
☑	Script-created Help Files Component	128 K

Destination Folder

C:\Temp

InstallShield
www.installshield.com

CHECKMARK dialog box style



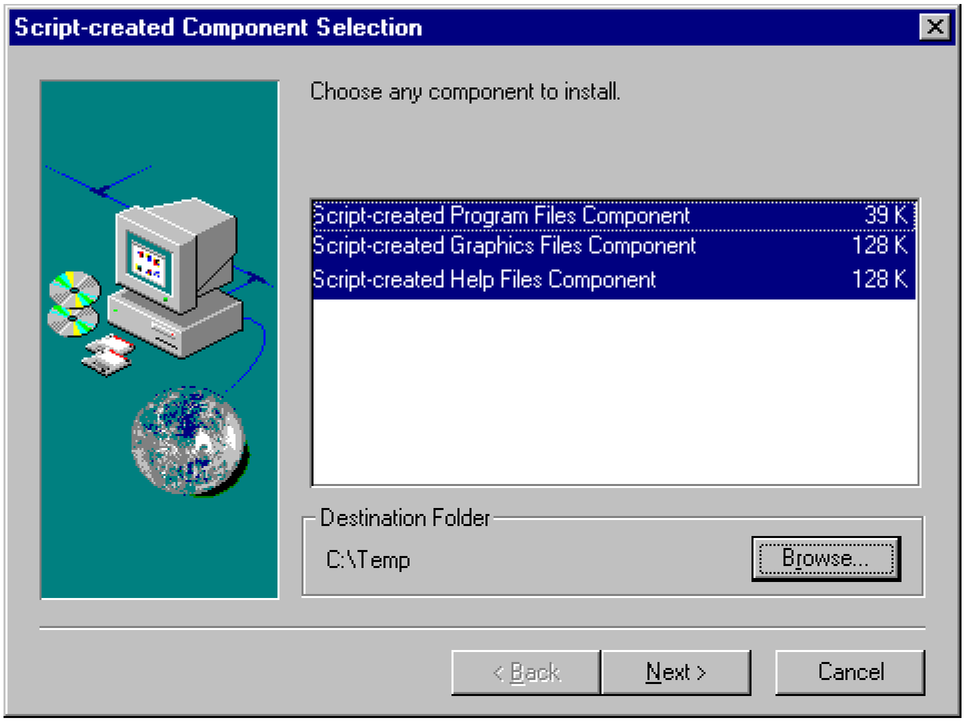
Script-created Component Selection

Choose any component to install.

✓	Script-created Program Files Component	39 K
✓	Script-created Graphics Files Component	128 K
✓	Script-created Help Files Component	128 K

Destination Folder
C:\Temp

CHECKLINE dialog box style



Script-created Component Selection

Choose any component to install.

Script-created Program Files Component	39 K
Script-created Graphics Files Component	128 K
Script-created Help Files Component	128 K

Destination Folder

C:\Temp

Warning messages

The compiler issues warning messages to alert you to irregularities in the script that do not prevent compilation, but which may result in inefficiencies or run-time errors.

W7501	macro redefinition
W7502	string/array size exceeds recommended limit
W7503	function defined but never called
W7504	incompatible function pointer assignment
W7505	typedef definition differs from previous

Error messages

Error messages do not stop the compiler.

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
C8018	typedef object illegal in this context
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
C8029	too many operands
C8030	unbalanced parentheses
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
C8041	function type required
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
C8056	#elif follows #else
C8057	#else not preceded by #if
C8058	#endif not preceded by #if
C8059	unrecognized preprocessor command
C8060	missing '(' after 'defined'
C8061	missing ')' after 'defined'
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
C8078	label illegal in this 'for' loop

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
C8094	label must be defined in function
C8095	label must be defined in main program
C8096	number too big for character
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
C8102	cannot specify DLL for function pointer
C8103	cannot specify DLL for ISObject
C8107	must use POINTER for function pointer
C8110	internal function required for pointer
C8111	function call not legal as lvalue
C8112	typedef includes instance of self
C8113	local variables cannot be external
C8114	preprocessor user define error

Fatal errors

Fatal error messages stop the compiler.

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
C8507	indirection not implemented yet
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

Internal errors

I9001	internal error

Constant data

A constant is a named data item with a defined value. InstallShield supports two types of constants:

- 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 an InstallShield constant will result in a compiler error.
- 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 statement 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

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:

- You cannot assign the contents of one structure to another structure with the assignment operator, as in `newstruct = struct1`.
- You must specify the size of all STRING declarations in a structure—InstallShield's autosizing feature does not work in typedef statements.
- You cannot declare a structure within a function.
- 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.
    SprintfBox(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.
    SprintfBox(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

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 Character data: a single 8-bit signed character. When a literal character appears in a script, it must be enclosed within single or double quotes. Note that you can assign a numeric ASCII value to a character value.To display a CHAR variable as a character, use the format specifier "%c" with the function SprintfBox. To display the numeric value of a CHAR variable use the specifier "%d".

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.

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 CmdGetHwndDlg or GetWindowHandle functions.

INT Equivalent to the NUMBER type; provided for convenience.

LIST Pointer to an InstallShield list. LIST variables are always initialized and uninitialized using the ListCreate and ListDestroy functions.

LONG Equivalent to the NUMBER type; provided for convenience.

NUMBER Signed four-byte integer. NUMBER is the recommended data type for storing numeric data. The data type is similiar to the LONG variable type in C++. It can hold any value between -2147483648 and +2147483647. Note that all numeric data types in InstallShield are equivalent to the NUMBER variable type.

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.

STRING An array of characters. STRING variables, which are similiar 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 diplay a STRING variable you can 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

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
\n	Inserts a carriage return and a line feed.
\’	Inserts a single quotation mark in the string.
\”	Inserts a double quotation mark in the string.
\r	Inserts a carriage return only. Does not insert a line feed.
\t	Inserts a tab character.
\ooo	Indicates an ASCII character—not an integer—in octal notation.
\xhh	Indicates an ASCII character—not an integer—in hexadecimal notation.
\\	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"
```

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: "nvNumber = 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

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 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

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 the Language reference, Hungarian notation is used in example scripts to indicate the data type of all variables. In the InstallScript Language Reference, Hungarian notation is used for parameter names in function syntax descriptions 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 InstallScript Function Syntax
c	Character (CHAR)	Character variable, constant, or literal.
const	Constant	Constant or literal value.
sz	String (STRING)	String variable, constant or literal.
sv	String (STRING)	String variable only
n	Number (NUMBER)	Number variable, constant or literal.
nv	Number (NUMBER)	String variable only
b	Boolean (BOOL)	Boolean variable, literal, or Boolean constant
bv	Boolean variable	Boolean variable only
p	POINTER	Poiner variable
l	Long integer (LONG)	Long integer variable, literal, or constant
lv	Long integer (LONG)	Long integer variable only
s	Short integer (SHORT)	Short integer variable, literal, or constant
i	Integer (INT)	Integer variable, literal, or constant
list	List (LIST)	List variable
h	Handle (HWND)	Handle variable
struct	Defined structure type	Not used
pstruct	Pointer to a defined structure type	Not used

{button ,AL(`Data Types;Data structures;Pointers;String indexing;String size',0,`',`')} 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

Passing data by reference in a call to a DLL requires the use of the UseDLL and UnUseDLL functions, which are available only in the InstallShield Professional 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 BlankLeadingZeroes uses the string indexing technique to replace the leading zeroes in a string representation of a number with blank characters.

```
prototype BlankLeadingZeroes(BYREF STRING);
STRING szString;

program
    szString = "00001234";
    BlankLeadingZeroes(szString);
    MessageBox(szString, INFORMATION);
endprogram

function BlankLeadingZeroes(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

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 under Windows 3.1 and up to 1,024 bytes under Windows 95 and NT). 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 as 129 to make 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 for which you have manually declared 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

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, six variables are declared. Note that the last declaration creates three numeric variables.

```
BOOL   bValidEntry;
LONG   lPopulation;
STRING szUserName[128];
NUMBER nFileSize, nDirSize, nDiskSpace;
```

Note that you may declare the size of a string variable by typing its length in brackets following the variable name. Maximum string size is 512 bytes under Windows 3.1 and 1,024 bytes under Windows 95 and Windows NT. If you do note specify a string length, InstallShield adjusts the memory allocation as necessary to fit the length of the string assigned to the variable.

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

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

_ A B C D E F G H I L M N O P R S T U V W X Y

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_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

_ A B C D E F G H I L M N O P R S T U V W X Y

_ A B C D E F G H I L M N O P R S T U V W X Y

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

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_UNITEDARABEMIRATES
ISLANG_ARABIC_BAHRAIN
ISLANG_ARABIC_QATAR
ISLANG_AFRIKAANS
ISLANG_ALBANIAN
ISLANG_BASQUE
ISLANG_BULGARIAN
ISLANG_BYELORUSSIAN
ISLANG_CATALAN
ISLANG_CHINESE
ISLANG_CHINESE_TAIWAN
ISLANG_CHINESE_PRCHINA
ISLANG_CHINESE_HONGKONG
ISLANG_CHINESE_SINGAPORE

ISLANG_CROATIAN
ISLANG_CZECH
ISLANG_DANISH
ISLANG_DUTCH
ISLANG_DUTCH_DUTCH
ISLANG_DUTCH_BELGIAN
ISLANG_ENGLISH
ISLANG_ENGLISH_USA
ISLANG_ENGLISH_UK
ISLANG_ENGLISH_AUSTRALIAN
ISLANG_ENGLISH_CANADIAN
ISLANG_ENGLISH_NEWZEALAND
ISLANG_ENGLISH_IRISH
ISLANG_ENGLISH_SOUTHAFRICA
ISLANG_ENGLISH_JAMAICA
ISLANG_ENGLISH_CARRIBEAN
ISLANG_ESTONIAN
ISLANG_FAEROESE
ISLANG_FARSI
ISLANG_FINNISH
ISLANG_FRENCH
ISLANG_FRENCH_FRENCH
ISLANG_FRENCH_BELGIAN
ISLANG_FRENCH_CANADIAN
ISLANG_FRENCH_SWISS
ISLANG_FRENCH_LUXEMBOURG
ISLANG_GERMAN
ISLANG_GERMAN_GERMAN
ISLANG_GERMAN_SWISS
ISLANG_GERMAN_AUSTRIAN
ISLANG_GERMAN_LUXEMBOURG
ISLANG_GERMAN_LIECHTENSTEIN
ISLANG_GREEK
ISLANG_HEBREW
ISLANG_HUNGARIAN
ISLANG_ICELANDIC
ISLANG_INDONESIAN
ISLANG_ITALIAN
ISLANG_ITALIAN_ITALIAN
ISLANG_ITALIAN_SWISS
ISLANG_JAPANESE
ISLANG_KOREAN
ISLANG_KOREAN_KOREAN
ISLANG_KOREAN_JOHAB
ISLANG_LATVIAN
ISLANG_LITHUANIAN
ISLANG_NORWEGIAN
ISLANG_NORWEGIAN_BOKMAL
ISLANG_NORWEGIAN_NYNORSK

ISLANG_POLISH
ISLANG_PORTUGUESE
ISLANG_PORTUGUESE_PORTUGUESE
ISLANG_PORTUGUESE_BRAZILIAN
ISLANG_ROMANIAN
ISLANG_RUSSIAN
ISLANG_SLOVAK
ISLANG_SLOVENIAN
ISLANG_SORBIAN
ISLANG_SPANISH
ISLANG_SPANISH_CASTILIAN
ISLANG_SPANISH_MEXICAN
ISLANG_SPANISH_MODERN
ISLANG_SPANISH_GUATEMALA
ISLANG_SPANISH_COSTARICA
ISLANG_SPANISH_PANAMA
ISLANG_SPANISH_DOMINICANREPUBLIC
ISLANG_SPANISH_VENEZUELA
ISLANG_SPANISH_COLOMBIA
ISLANG_SPANISH_PERU
ISLANG_SPANISH_ARGENTINA
ISLANG_SPANISH_ECUADOR
ISLANG_SPANISH_CHILE
ISLANG_SPANISH_URUGUAY
ISLANG_SPANISH_PARAGUAY
ISLANG_SPANISH_BOLIVIA
ISLANG_SWEDISH
ISLANG_THAI
ISLANG_TURKISH
ISLANG_UKRAINIAN
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

_ A B C D E F G H I L M N O P R S T U V W X Y

_ A B C D E F G H I L M N O P R S T U V W X Y

_ A B C D E F G H I L M N O P R S T U V W X Y

_ A B C D E F G H I L M N O P R S T U V W X Y

_ A B C D E F G H I L M N O P R S T U V W X Y

_ A B C D E F G H I L M N O P R S T U V W X Y

_ A B C D E F G H I L M N O P R S T U V W X Y

_ A B C D E F G H I L M N O P R S T U V W X Y

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.

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.

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.

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

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

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:

- A boolean or integer constant, variable or literal.
- An expression that produces a boolean or integer result.
- 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

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

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

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 “exit”, 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;if-then structure;if-then-else structure;Nested if-then-else structure;elseif structure;if structure with goto',0,`',`')}   See also

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

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:

- The condition must be followed by a goto statement.
- The keyword then is not used.
- 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

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:

- The repeat statement must loop at least once. A while statement may not loop at all.
- 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

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 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.

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

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
```

Functions

A function is a named set of instructions that operate together to perform a certain task. Every function has the following characteristics:

- 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.
- A function is independent. In most cases, any function can perform its instructions without interfering with other parts of the program.
- 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.
- 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:

- Built-in functions, which are supplied by InstallShield or included for Sd dialog boxes.
- User-defined functions, which you create.
- 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

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.

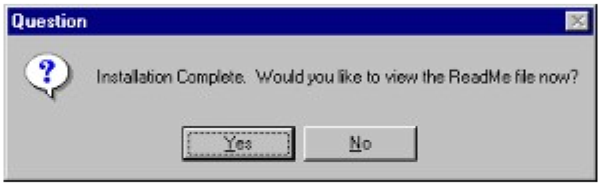
Like C, InstallScript is case sensitive. Be sure to pay close attention to the capitalization of letters in the built-in function names.

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:



Installation Complete. Would you like to view the ReadMe file now?

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 Functions;Advanced Batch File Functions;Built in Dialog Box Functions;EZ CONFIG SYS File Functions;Advanced Config File Functions;Custom Dialog Box Functions;Data Structure Memory Access Functions;Extensibility Functions;File Directory Functions;Information Functions;List Processing Functions;Long Filename Functions;Miscellaneous Functions;Path String Modification Functions;INI File Profile File Functions;Registry Functions;Sd Dialog Box Functions;Shell Functions;String Functions;Version Checking Functions;Visual Interface Functions',0,`',`')} See also

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.

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, BACKGROUNDCAPTION); // 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.

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 exit 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 exceed 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(`BYREF operator;return',0,`',`')}   See also

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);
```

Calling a DLL Function

professional edition only

Follow the steps below to call a function that resides in a DLL.

1. Declare the DLL function, using the following format:

   ```
   prototype <DLL filename>.<function name> ( <parameter types> );
   ```

2. Before calling the DLL function, call UseDLL to load the DLL into memory.
3. Call the DLL function (do not include the DLL filename):

   ```
   <function name> ( <parameters> );
   ```

4. Call UnUseDLL to release the DLL from memory.

{button ,AL(`UseDLL;UnuseDLL;CALLDLLFx;BYREF operator;Call a custom DLL function from unInstallShield',0,`',`')} See also

Call a custom DLL function from unInstallShield

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:

- Functions ending in -String that work with string lists only
- Functions ending in -Item that work with number lists only
- 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

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.

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.

InstallShield
www.installshield.com

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

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;
```

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.

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.
SprintfBox (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);
```

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.

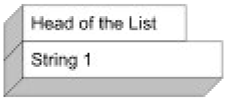
{button ,JI(`',`Adding_an_element_after_the_current_element')}Adding an element after the current element.

{button ,JI(`',`Adding_elements_before_and_after_the_current_element')} Adding elements before and after the current element.

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:

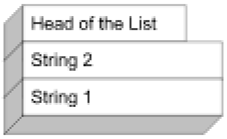
```
// Create the empty list of strings.
listID = ListCreate (STRINGLIST);

// 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);
```



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:

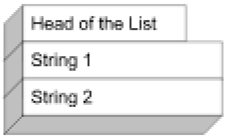
```
// Create the empty list of strings.
listID = ListCreate (STRINGLIST);

// 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);
```



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:

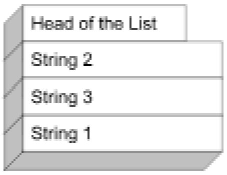
```
// Create the empty list of strings.
listID = ListCreate (STRINGLIST);

// 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);
```



Adding elements before and after the current element

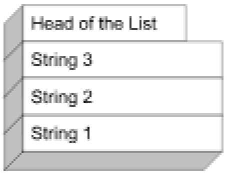
As another example, the code segment shown below creates a new list and puts "String 1" in the first position. "String 2" is then added before "String 1", leaving "String 2" in the first position as the current string. Next, "String 3" is added after the current string, resulting in the list depicted 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."
```

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);
```

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.

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);
```

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

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;
```

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

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

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

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 must include a space both before and after an arithmetic operator in InstallShield. Although you do not necessarily have to do this with other types of operators, we recommend that you do so in order to make the script more readable and maintain a consistent look to your code.

{button ,AL(`Unary Arithmetic Operators;Binary Arithmetic Operators;Arithmetic Operator Precedence',0,`',`')}See also

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:

- Negative (-) unary has first precedence.
- Multiplication, and division have second precedence.
- Addition and subtraction have third precedence.

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

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);
```

If you declared a size for a string variable in the declare block, 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;
```

InstallShield will automatically size a string buffer for you if you do not specify a size. By default, InstallShield will autosize 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 under Windows 3.1 and up to 1,024 bytes under Windows 95 and NT).

BYREF operator

To force InstallShield to pass data by reference when you are not using pointers, use the BYREF operator before non-pointer parameter(s) in the declaration of the user-defined function. The BYREF operator informs the InstallShield script compiler that the following parameter must be passed by reference, not by value.

For example, to pass a string by reference to a user-defined function called Tester, use the following function declaration in the declare block:

```
   prototype Tester( BYREF STRING );
```

You can also pass data by reference to functions in DLLs. You must prototype the DLL function, load the DLL into memory, and unload the DLL after you are done using it. It is not necessary to use the BYREF operator when prototyping string parameters of DLL functions in InstallScript. All string parameters are ALWAYS passed to DLL functions by reference.

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.

{button ,AL(`Declaring functions;Returning a value from a function;Calling a function;Calling a DLL function',0,`',`')} See also

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	Determines whether operand 1 equal to operand 2.
>	Greater than	x > y	Determines whether operand 1 greater than operand 2.
<	Less than	x < y	Determines whether operand 1 less than operand 2.
>=	Greater than or equal to	x >= y	Determines whether operand 1 greater than or equal to operand 2.
<=	Less than or equal to	x <= y	Determines whether operand 1 less than or equal to operand 2.
!=	Not equal to	x != y	Determines whether operand 1 not equal to operand 2.

When you use a relational operator in an if or while statement, the program will follow one of two actions:

- If the expression is TRUE, the command that follows it is executed.
- If the expression is FALSE, some other action takes place.

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

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. It 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:

Less than (<), less than or equal to (<=), greater than (>), and greater than or equal to (>=) have first precedence.

Equal to (=) and not equal to (!=) are evaluated next.

Therefore, when combining arithmetic and relational expressions, 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. Less than (<), less than or equal to (<=), greater than (>), and greater than or equal to (>=) have fourth

precedence.

5. Equal to (**=**) and not equal to (!=) have fifth precedence.

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, and if it is TRUE, y is added to 6. If 7 is not greater than y (FALSE), the program will not add y 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:

<table>
<tr><th>Symbol</th><th>Operation</th><th>Example</th><th>Description</th></tr>
<tr><td>&&</td><td>AND</td><td><code>exp1 && exp2</code></td><td>True only if both exp1 and exp2 are true; otherwise, false.</td></tr>
<tr><td>||</td><td>OR</td><td><code>exp1 || exp2</code></td><td>True if either exp1 or exp2 is true; false (0) only if both are false.</td></tr>
<tr><td>!</td><td>NOT</td><td><code>!exp1</code></td><td>False if exp1 is true; true if exp1 is false.</td></tr>
</table>

Logical operators have a lower precedence than arithmetic or relational operators. Among logical operators, the AND operator has higher precedence than the OR operator.

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;
```

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

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 run-time error (“Error 401”).

{button ,AL(`Append to path;Concatenate;Find string',0,`',`')}   See also

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

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. Right-shifting an integer by n has the same effect as dividing the number by 2.

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:

- The indirection operator can be used with number pointers only.
- The indirection operator cannot be used to assign a value to a memory location.
- The indirection operator cannot be used as an argument to a function whose parameter has been defined with the BYREF operator.
- The indirection operator cannot be used to declare a pointer.
- The indirection operator cannot be used to declare a variable parameter in a function declaration.

{button ,AL(`Address operator;Pointers',0,`',`')} See also

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

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.Btptr->Cptr->Cmember is invalid in InstallScript.

{button ,AL(`Data structures;Pointers',0,`',`')} See also

Preprocessor statements

Preprocessor statements, or directives, are interpreted by the InstallShield Script Compiler as the script is being compiled. Preprocessor statements are called compile-time statements because they are executed only when the InstallShield Script Compiler compiles the script.

Preprocessor statements can tell the InstallShield Script Compiler to define constants, to include or not to include script statements, and to determine whether certain compile-time conditions are true or false. InstallShield directives are very similar to directives in the C language, but not exactly the same.

The variables you use in preprocessor statements are different from those used in run-time statements. You must declare the variables with a preprocessor directive, which enables InstallShield to recognize the variables as compile-time variables.

You can place preprocessor statements in any block of the script. Preprocessor statements are always preceded by a number sign (#). Never end a preprocessor statement with a semicolon (;).

The following Boolean operators are supported in #if, #ifdef, #ifndef, and #elif statements:

- Logical OR (||)
- Logical AND (&&)
- Relational (=, !=, >, >=, <, <=)

InstallShield supports the following preprocessor statements:

#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).

#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.

A preprocessor statement does not end with a semicolon.
Preprocessor statements 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

#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:

- 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.
- Constants you declare with a #define statement cannot begin with numbers.
- 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.
- InstallShield assigns a value of zero (0) to an undefined constant.

{button ,AL(`Defining Constants from the Command Line',0,`',`')}   See also

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 /DMAX_BUF=150 /DMAX_LENGTH=255 SETUP.RUL is equivalent to the define statements shown below:

```
#define  MAX_BUF     150
#define  MAX_LENGTH  255
```

There following restrictions apply when defining constants from the command line:

- If you define a constant at the command line that you defined with a #define statement in the script, you will generate an error.
- You can define only numeric constants from the command line.
- If you declare a constant at the command line that you declared as a variable in the script, the value of the constant will be lost at run time.

#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.

#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.
   . . .
#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:

- You cannot place comments on the same line as #ifdef and #ifndef directives.
- 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.
- Do not test a constant that has a value of 0 (zero) with an #ifdef or #ifndef statement.
- You can test only numeric constants with an #ifdef or #ifndef statement.

#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.

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.

#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:

- InstallShield does not handle paths with more than 260 characters, including the filename.
- 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.
- 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"
```

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;Building a function',0,`',`')} See also

Structure of a setup script

Every setup script includes data declarations, a program block, and function block(s).

- 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
- The program block must appear before the function blocks.
- 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;Building a function',0,`',`')}   See also

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

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;Building a function',0,`',`')}   See also

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:

- Most statements end with a semicolon (;). This includes many one-word statements, such as end;, exit;, return;, and so on.
- Preprocessor statements, such as #define and #include, never end with a semicolon.
- 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.
- End a label, such as start: or starthere:, with a colon (:).
- Enclose parameter lists within parentheses. When you have more than one parameter, separate them with commas.

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 single quotes, and then type the double quotes:

```
//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 installation 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 quotes.

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
```

Writing comments

Comments can be a useful programming tool. They are very helpful if you have to come back to an installation script at a later date. They can also be used to “comment out” portions of your script you want to test it for debugging purposes.

In InstallShield you can designate comments in two ways. You can enclose the comment between /* and */. This allows you 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 format, //, causes the compiler to ignore 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.

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")} BATCH_INSTALL
Indicates whether or not a locked file was encountered while transferring files with the LOCKEDFILE or SHAREDFILE option.

{button ,JI("LangRef>main","CMDLINE")} CMDLINE
Specifies a command line parameter to be passed to Setup.exe.

{button ,JI("LangRef>main","COMMONFILES")} COMMONFILES
Stores the fully-qualified path to the Common files folder.

{button ,JI("LangRef>main","ERRORFILENAME")} ERRORFILENAME
Stores the name of the file at which an error occurred.

{button ,JI("LangRef>main","FOLDER_DESKTOP")} FOLDER_DESKTOP
Stores the fully-qualified path to the Windows 95 or Windows NT 4.0 Desktop folder.

{button ,JI("LangRef>main","FOLDER_PROGRAMS")} FOLDER_PROGRAMS
Stores the fully-qualified path to the Windows 95 or Windows NT 4.0 Start menu\Programs folder.

{button ,JI("LangRef>main","FOLDER_STARTMENU")} FOLDER_STARTMENU
Stores the fully-qualified path to the Windows 95 or Windows NT 4.0 Start menu folder.

{button ,JI("LangRef>main","FOLDER_STARTUP")} FOLDER_STARTUP
Stores the fully-qualified path to the Windows 95 or Windows NT 4.0 StartUp folder.

{button ,JI("LangRef>main","INFOFILENAME")} INFOFILENAME
Specifies the name of a backup file created by InstallShield.

{button ,JI("LangRef>main","ISRES")} ISRES
Contains the fully-qualified filename of the _isres.dll loaded for the current instance of InstallShield.

{button ,JI("LangRef>main","ISUSER")} ISUSER
Contains the fully-qualified filename of the _isuser.dll loaded for the current instance of InstallShield.

{button ,JI("LangRef>main","ISVERSION")} ISVERSION
Contains the version of Setup.exe that is running.

{button ,JI("LangRef>main","MEDIA")} MEDIA
Stores the name of the current media library.

{button ,JI("LangRef>main","MODE")} MODE
Indicates whether the program Setup is running in normal or silent mode.

{button ,JI("LangRef>main","PROGRAMFILES")} PROGRAMFILES
Stores the fully-qualified path to the Windows 95 or Windows NT 4.0 Program files folder.

{button ,JI("LangRef>main","SELECTED_LANGUAGE")} SELECTED_LANGUAGE
This system variable contains the numeric language code of the language the setup is using.

{button ,JI("LangRef>main","SRCDIR")} SRCDIR
Contains the fully-qualified path to the source folder.

{button ,JI("LangRef>main","SRCDISK")} SRCDISK
Contains the name of the drive with the source disk.

{button ,JI("LangRef>main","SUPPORTDIR")} SUPPORTDIR
Contains the fully-qualified path to the folder used for temporary files.

{button ,JI("LangRef>main","TARGETDIR")} TARGETDIR
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.

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.

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.

COMMONFILES

This system variable stores the fully-qualified path to the common files folder under Windows 95 and Windows NT version 4.0. COMMONFILES is initialized only once—at the beginning of the setup process.

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

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.

- 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.
- 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 Window 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.

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 Window 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.

FOLDER_STARTMENU

This system variable stores the fully-qualified path to the Start Menu\Programs 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.

- 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.
- 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 Window 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.

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.

- 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.
- 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 Window 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.

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.

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.

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.

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.

MEDIA

This system variable holds the name of the current media library. The default value is 'DATA'.

MODE

The system variable MODE holds one of the following constant values:

- SILENTMODE—Indicates that Setup.exe is running in silent mode.
- 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;
```

PROGRAMFILES

Under Windows 95 and Windows NT 4.0, this system variable stores the fully-qualified path to the Program Files folder. PROGRAMFILES is initialized only once, at the beginning of the setup process.

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.

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.

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.

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 _user.cab. InstallShield assigns to SUPPORTDIR the fully-qualified path to that folder.

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;VerSearchAndUpdateFile;VerFindFileVersion;VerUpdateFile;XCopyFile',0,`',`')} See also

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.

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);
```

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.

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.

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.

WINSYSDISK

During setup initialization, InstallShield initializes the system variable WINSYSDISK by assigning to it the name of the disk drive that contains Windows\System folder, usually the C: drive. Note that InstallShield includes the colon (:) with the drive letter.

This feature is available only in the InstallShield Professional Edition. Click here for a list of differences between the Free and Professional editions.

InstallShield Help is attempting to send you to the InstallShield Web site (www.installshield.com). If your preferred Web browser is not launched, click here , then restart InstallShield help.

InstallShield Help found Setbrows.exe but could not run it. Double-click the <InstallShield location>\Program\Setbrows.exe icon.

InstallShield Help could not find Setbrows.exe. You may have deleted this file; check to see if Setbrows.exe is in your <InstallShield location>\Program folder.

Get the latest versions of InstallShield help files

For the latest versions of InstallShield help files, visit our documentation Web page.

Visit the InstallShield Knowledge Base

For answers to many commonly asked questions and new information about InstallShield that may not appear in the documentation, visit the InstallShield Knowledge Base.

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 a 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.

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

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.

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 desired.

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.

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

The name of the environment variable you want to modify. For example, if you are modifying the PATH statement, enter "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

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

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

Specify 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

- EzBatchAddPath may fail if the default batch file is hidden or read-only.
- 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

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
```

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

The line of text you want to add to 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.

szRefKey

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

The following constants specify the options to use:

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.

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 nOption 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

- 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.
- The EzBatchAddString function may fail if the default batch file is hidden or read-only.
- 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

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
```

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

The new string you are inserting to replace 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

- 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.
- Some common keys in a batch file are PATH, COMSPEC, TEMP, Smartdrv.exe, Win.com, and Share.exe.
- The EzBatchReplace function may fail if the default batch file is hidden or read-only.
- 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

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
```

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 intitializes, 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.

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

The keyword you are adding to the batch file. PATH, TEMP, and MYENV are examples of valid keys you can use for this parameter.

szValue

The value of the key you are adding 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

The reference key relative to which you are adding szKey in the batch file.

nOptions

Use the following constants to specify where in the file to insert the line:

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 nOption 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

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);

   SprintfBox (INFORMATION, "BatchAdd", "Successful.\n\nThe target batch"  +
               "file has been altered.  The original was saved as " +
               EXAMPLE_BAK + ".");

endprogram

// Source file: Is5fn007.rul
```

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

The reference keyword that identifies the line or lines you are deleting.

nOptions

Indicate whether szKey contains a command statement or an environment variable. The following constants are available:

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

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
```

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

The fully-qualified name of the batch file you are loading into memory. When you enter 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

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;

/*--------------------------------------------------------------------------*\
 *
 * 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
```

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

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

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;

/*--------------------------------------------------------------------------*\
 *
 * 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
```

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

The reference key you are searching for. The reference key can be either 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 enter Win.com, the search looks for this reference key only. If you enter Win, the reference keys Win.exe, Win.dll, Win.sys, and so on will be returned if they exist in the batch file.

svResult

The value of the reference key that was found in the batch file.

nOptions

The following constants indicate where to start the search:

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

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
```

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

BatchGetFileName 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

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
```

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

The string that identifies the line you are moving.

szRefKey

The key that identifies the reference line used to position the line you are moving. 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

The following options indicate where to move the line:

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

Specify whether szMove is a command or an environment variable. These constants are available:

0

Specifies szMove is an environment variable.

COMMAND

Specifies 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

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
```

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:

- It does verify that the specified file exists.
- 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
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

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
```

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 default 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.

AskDestPath

Syntax

AskDestPath (szTitle, szMsg, svDir, nReserved);

Description

The AskDestPath function presents the Choose Destination Location dialog box, which displays the default destination location and allows the end user either to accept that location or open the Choose Folder dialog box to specify an alternate destination location.

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.

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.

Parameters

szTitle

Enter the text you want to set as the title of the dialog box. To display the default title, "Choose Destination Location," enter a null string ("").

szMsg

Enter the message you want to display. You can enter multiple lines of static text in this parameter. Use the \n escape sequence to insert a line break. To display the default instructions for this dialog box, enter a null string ("").

svDir

Enter the default path you want to display. When the function returns, this parameter contains the path to the selected folder.

nReserved

Enter 0 (zero) in this parameter. No other value is allowed.

Return values

BACK

Indicates that the Back button was selected.

NEXT

Indicates that the Next button was selected.

Comments

You cannot use a custom exit handler with the AskDestPath dialog box. It is a built-in dialog box and will use the default exit routine if the end user presses the Cancel button.

{button ,JI(`LANGREF.HLP>Examples',`AskDestPath_example')}   Example

{button ,AL(`AskOptions;AskYesNo;Welcome',0,`',`')}   See also

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
```

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

Enter one of the following constants to specify the type of controls:

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

Enter the text 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 the escape sequence \n to insert a line break.

szText1

Enter 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

When AskOptions returns, this parameter is TRUE if the first check box or radio button was selected, FALSE if it was deselected. To set the initial state of the first option, assign TRUE or FALSE to this variable before calling AskOptions.

szText2

Enter 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

When AskOptions returns, this parameter is TRUE if the second check box or radio button was selected, FALSE if it was deselected. To set the initial state of the second option, assign TRUE or FALSE to this variable before calling AskOptions.

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 AskOption 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.

Comments

You cannot use a custom exit handler with the AskOptions dialog box. It is a built-in dialog box and will use the default exit routine if the user presses the Cancel button.

{button ,JI(`LANGREF.HLP>Examples',`AskOptions_example')} Example

{button ,AL(`AskText;AskYesNo;SetDialogTitle',0,`',`')} See also

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
```

AskPath

Syntax

AskPath (szMsg, szDefPath, svResultPath);

Description

The AskPath function presents 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:

- Accept the default path.
- Edit the default path.
- 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
The message to display. To display the default instructions for this dialog box, enter a null string ("").

szDefPath
The default path to display in the edit field. The end user can modify this string.

svResultPath
AskPath returns the resulting path name in svResultPath, 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

- When you declare the variable svResultPath, you can can declare its length or let InstallShield autosize it. When a string length has been specified for svResultPath, the string entered by the end user is inspected and if it is longer than the declared length, the following error message is displayed.

  ```
  String variable is not large enough for string.
  Please check string declarations.
  ```

- To accommodate both the last slash added by AskPath and the string null terminator, the path string the user enters must be two characters shorter than the length of the variable svResultPath. You may want to include a message in this dialog box telling the user the maximum number of characters to enter.
- The edit field scrolls to accommodate long strings.

{button ,JI(`LANGREF.HLP>Examples',`AskPath_example')} Example

{button ,AL(`AskOptions;AskText;AskYesNo;SetDialogTitle',0,`',`')} See also

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
```

AskText

Syntax

AskText (szQuestion, szDefault, svResult);

Description

The AskText function presents a dialog box that displays a question or a statement along with a single-line edit field in which the end user can type an answer or response. This edit field can contain a default string for the user to accept or modify. The default title is Enter Information. To change the contents of the title bar, call SetDialogTitle before calling AskText.

View sample dialog

Parameters

szQuestion

Enter the question or statement to display. If the statement is too long for one line, use the \n escape sequence to insert a line break.

szDefault

Enter the default text, enclosed by quotation marks, that you want to appear in the edit field.

svResult

When the user clicks the Next button, svResult returns the text string the user has entered.

Return values

BACK

Indicates that the Back button was selected.

NEXT

Indicates that the Next button was selected.

Comments

- The edit field will scroll if the user types a string too long to fit in the window. AskText can return a string of up to 512 characters under Windows 3.1 and 1,024 characters under Windows NT and Windows 95. If the string entered by the end user is too large, InstallShield truncates it.
- The size of the string variable in svResult should be large enough to accommodate the string in svDefault.

{button ,JI(`LANGREF.HLP>Examples',`AskText_example')} Example

{button ,AL(`AskOptions;AskYesNo;SetDialogTitle',0,`',`')} See also

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
```

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:

- Question mark icon
- Question text
- Yes button
- 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 Windows API and is a system modal message box, retaining focus until it receives a response from the user. You cannot change the text in the Yes or No buttons. This text is displayed automatically in the Microsoft Windows operating system language of the target system. For example, if the target system is using the French version of Windows, these buttons contain "Oui" and "Non." If you require more flexibility, create a custom dialog box.

Parameters

szQuestion

Enter the question you want to display in the message box. If the message is too large to fit on one line, embed \n escape characters in the message to insert line breaks.

nDefault

Pass one of the following constant to preselect one of the buttons:

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;AskYesNo;SetDialogTitle',0,`',`')} See also

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
```

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
The message that prompts the user to insert the proper disk. To display the default instructions for this dialog box, enter a null string ("").

szTagFile
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 enter 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',0,`',`')} See also

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
```

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

Enter the message you want 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 the \n escape character at the appropriate place in the string.

nType

Use this parameter to specify the type of message box to create and the type of icon you want to appear in the message box. These constants are available (Windows 95 icons are shown):

INFORMATION

WARNING

SEVERE

Return values

The return value is insignificant 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',0,`',`')} See also

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
```

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

Enter the text you want to display in the title of the dialog box. To display the default title, "Restarting Windows", enter a null string ("").

szMsg

Enter the message you want to display. To display the default instructions for this dialog box, enter a null string ("").

nDefChoice

Use this parameter to specify the default radio button selection. These constants are available:

SYS_BOOTWIN

Use this constant to make 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

Use this constant to make rebooting the machine ("Yes, I want to restart my computer now.") the default radio button selection.

0

Use this constant to make 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

- 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.
- The first radio button ("Yes, I want to restart Windows now.") is hidden on 32-bit systems because it does not apply.
- 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.

- 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.
- 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',0,`',`')} See also

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
```

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
Enter the text you want to display as the caption of this dialog box. To display the default title, "Choose Folder," enter a null string ("").

szMsg
Enter the message you want to display. To display the default instructions for this dialog box, enter a null string ("").

svDir
Enter the name of a folder that will appear as the default selection. When SelectDir returns, this parameter contain the fully-qualified name of the selected folder.

bCreate
Use this parameter to specify whether or not you want InstallShield to create the specified folder if it does not already exist. These constants are available:

TRUE
Indicates that you want InstallShield to create the folder if it does not exist.

FALSE
Indicates that you do not want InstallShield to create the folder 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',0,`',`')}   See also

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
```

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

Parameters

szTitle

Enter the text you want to use as the title of this dialog. To display the default title, "Setup Type," enter a null string ("").

szMsg

Enter the message you want to display at the top of the dialog box. To display the default instructions for this dialog box, enter a null string ("").

szReserved

Enter a null string ("")—no other value is allowed.

nType

Use one of the predefined constants listed below to set the default setup type when the dialog box is first displayed:

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

Enter 0—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(`Welcome',0,`',`')} See also

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 SETUPTYPETITLE     "Choose Setup Type"
#define SETUPTYPEMSG       "Select a setup type."
#define COMPONENT          ""
#define SDCMPDLGTITLE      "Component Selection"
#define SDCMPDLGMSG        "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(SETUPTYPETITLE, SETUPTYPEMSG, "", TYPICAL, 0);

    // If Custom setup type chosen, let user select components
    // and set change location if desired.
    if (nResult = CUSTOM) then
        SdComponentDialogAdv( SDCMPDLGTITLE, SDCMPDLGMSG,
                              TARGETDIR, COMPONENT);
    endif;

    // 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 );

    // 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
```

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

Enter the type of icon to you want to appear in the message box. These constants are available (Windows 95 icons are shown):

INFORMATION

WARNING

SEVERE

szTitle

Enter the title you want to display in the message box. To display the default title, "Error," enter a null string ("").

szFormat

Enter a string that includes a format specifier for each argument to be included in the message.

arg

This parameter specifies up to ten 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. SprintfBox will fail under the following conditions:

- More than ten format specifiers and arguments are specified.
- The number of arguments does not match the number of format specifications.
- 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',0,`',`')} See also

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
```

Welcome

Syntax

Welcome (szTitle, nReserved);

Description

Use the Welcome function to create and display the Welcome dialog box.

View sample dialog

Parameters

szTitle
Enter the text you want to appear as the title of this dialog box. To display the default title, "Welcome," enter a null string ("").

nReserved
Enter 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 present 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. Most setup scripts call InstallationInfo for other reasons anyway, but you must ensure that it is called before calling the Welcome function.

{button ,JI(`LANGREF.HLP>Examples',`Welcome_example')} Example

{button ,AL(`SdWelcome',0,`',`')} See also

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
```

Component functions

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.

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

The media name of a script-created component set. If it does not exist, ComponentAddItem will create it.

szComponent

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

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

Specify the default selection setting of the component. These constants are available:

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;ComponentSelectItem;SdAskOptions;SdAskOptionsList;SdComponentDialog;SdComponentDialog2;SdComponentDialogAdv;SdComponentMult',0,`',`')}   See also

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
```

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 may be either the file media library or a script-created component set. 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 is a script-created component or a 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
The name of the media.

svTarget
If the target drive doesn’t have enough space, the target path will be returned.

nvSize
If the target drive doesn’t have enough space, the size required will be returned.

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

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
```

ComponentDialog

Syntax

ComponentDialog (szTitle, szMsg, svDir, szComponents);

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.

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.

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.

Parameters

szTitle

The text you want to appear as the dialog box title. To display the default title, "Select Components," enter a null string ("").

szMsg

The message you want 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, enter a null string ("").

svDir

The default destination location. When the function returns, svDir will contain the selected folder. The location that is 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

The component whose subcomponents you want to display for selection. Use a null string ("") 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.

{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

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 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
```

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:

- ComponentSetupTypeEnum
- ComponentSetupTypeSet

Parameters

svMedia
When relevant to the error indicated in nvError, the name of the media involved is returned in svMedia.

svComponent
When relevant to the error indicated in nvError, the name of the component involved is returned in svComponent.

svFileGroup
When relevant to the error indicated in nvError, the name of the file group involved is returned in svFileGroup.

svFile
When relevant to the error indicated in nvError, the name of the file involved is returned in svFile.

nvError
When ComponentError is called after a component-related function returns a value less than zero, ComponentError always returns the error code in nvError.

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,	Specified component already exists.

	nvError	
-103	svMedia, svComponent, nvError	ComponentSelectItem tried to select a 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 ComponentMoveData is not valid.
-108	svMedia, svFile, nvError	Target disk does not have enough space. svFile will contain target path.
-109	svMedia, svFile, nvError	EnterDisk 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 file media library 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 ComponentFileInfo is not valid.
-117	svMedia, svComponent,	Cannot read the specified file. If svComponent and svFileGroup are empty,

	svFileGroup, svFile, nvError	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 ComponentFileEnum 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 ComponentGetData, which is not allowed!
-141	svMedia,	Specified password cannot be found. The

	svComponent, svFile, nvError	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

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;
        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
```

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
The name of the media.

szComponent
The name of the component to be enumerated. Do not use a null string (""); ComponentFileEnum requires a specific component name. For more information about specifying components and subcomponents in function calls, click here.

szQuery
Specify 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 quotes. 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
All files matching szQuery will be returned in this string list. Use ListCreate to create the string list.

nOption
Can be one of the following:

INCLUDE_SUBDIR
Include files in subfolders.

NO_SUBDIR
Do not include files in subfolders.

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

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
```

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

The name of the media.

szComponent

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

Specify 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 quotes.

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\\Is5.dll"	Shared Files is a file group and Is5.dll is a file name.
"Shared Files\\dev\\myapp\\dlls\\Is5.dll"	Shared Files is a filegroup, \\dev\\myapp\\dlls is a path, and Is5.dll is a file name.

nInfo

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
A number value is returned in nvResult 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
When nInfo specifies a date, time, or version, the resulting string is returned in svResult.

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

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
```

ComponentFilterLanguage

Syntax

ComponentFilterLanguage(szMedia, nLangID, bFiltered);

Description

The ComponentFilterLanguage function filters (excludes) files from file transfer based on language. By default, all languages included in the media build are unfiltered (included). The language ID options used for the nLangID parameter *cannot* be combined with the bitwise OR operator (|). You must call ComponentFilterLanguage for each language you wish to filter or unfilter.

The easiest way to filter language specific file groups during the setup is to do the following:

- Filter (exclude) all languages by calling ComponentFilterLanguage with ISLANG_ALL in the parameter nInfo and bFiltered set to TRUE.
- For each language that you want to install, call ComponentFilterLanguage with the appropriate language constant in nInfo and with the parameter bFiltered parameter set to FALSE. Each call will unfilter (include) file groups for the language specified in nInfo.

You *cannot* specify multiple language constants in the nLangID parameter by using the OR operator (|). Specifying multiple language constants will cause the function to perform incorrectly.

When using ComponentFilterLanguage, remember to add `#include "Sdlang.h"` before the program block. If you are using the Setup.rul generated by the Project Wizard, this line is already included for you.

Parameters

szMedia
The name of the media.

nLangID
The ID of the language you want to filter or unfilter. Note that only *one* language constant can be specified for each function call. Click here for a list of constants that can be passed in this parameter.

bFiltered
Use one of the following constants to specify whether or not you want the specified language(s) filtered (excluded).

TRUE
Filter the language specified in nLangID.

FALSE
Do not filter the language specified in nLangID.

Return values

0
ComponentFilterLanguage was successful.

< 0
ComponentFilterLanguage failed.

Comments

- When using this function in conjunction with the GetSystemInfo function, you must consider the following: The language constants that can be used to designate language specific file groups are a small subset of the language constants that can be returned by GetSystemInfo.

  If your setup includes language filtering based on these return values, you must use a switch statement to convert constants returned by this function into one of the constants supported for language filtering. For information about filtering language-dependent files based on the target system's language, click here.

- Remember that any edition of InstallShield will allow you to designate file groups for any language or sub-language that is supported by Windows; however, for a language specific file group to be built by the Media Build Wizard, the edition of InstallShield that you are using must support the language of that file group. Your setup must also support the language of the filegroup.

  If your setup includes language specific file groups that are designated as specific to a language not supported by the edition of InstallShield you are using or by your setup, the filegroup(s) will be filtered (not included) by the Media Build Wizard.

{button ,JI(`LANGREF.HLP>Examples',`ComponentFilterLanguage_example')} Example

{button ,AL(`Script created component set vs. file media library;Specifying components in function calls;Componentfunctions;Mark application files as language dependent;Determining the default language of the target system',0,`',`')} See also

ComponentFilterLanguage example

```
/*--------------------------------------------------------------------------*\
 * This example illustrates the use of ComponentFilterLanguage.
 *
 * First, ComponentFilterLanguage is called to exclude all languages.
 *
 * Next, GetSystemInfo is called to determine the target computer's
 * default language/locale.
 *
 * Then, ComponentFilterLanguage is called again to include the
 * language appropriate for the target computer. If language support
 * is not provided for the target computer, English is used.
 *
 * Finally, the ComponentMoveData is called to create the installation.
 *
\*--------------------------------------------------------------------------*/

// Sdlang.h contains ComponentLanguageFilter nLangID options and is required.
#include "sdlang.h"

STRING  szResult;
NUMBER  nResult, nDisk;

program

    // Filter out all language-specific file groups.
    ComponentFilterLanguage(MEDIA, ISLANG_ALL, TRUE);

    // Retrieve the target machine's default language/locale setting.
    GetSystemInfo(LANGUAGE, nResult, szResult);

    // Turn off filtering for file groups specific to the target
    // machine's default language/locale setting.
    switch (nResult)
        case ISLANG_FRENCH_CANADIAN:
            ComponentFilterLanguage(MEDIA, ISLANG_FRENCH_CANADIAN, FALSE);

        case ISLANG_FRENCH_STANDARD,
             ISLANG_FRENCH_BELGIAN,
             ISLANG_FRENCH_SWISS,
             ISLANG_FRENCH_LUXEMBOURG:
             ComponentFilterLanguage(MEDIA, ISLANG_FRENCH_STANDARD, FALSE );

        case ISLANG_GERMAN_STANDARD,
             ISLANG_GERMAN_SWISS ,
             ISLANG_GERMAN_AUSTRIAN,
             ISLANG_GERMAN_LUXEMBOURG,
             ISLANG_GERMAN_LIECHTENSTEIN:
             ComponentFilterLanguage(MEDIA, ISLANG_GERMAN, FALSE);

        // Use English as a default.
        default: ComponentFilterLanguage(MEDIA, ISLANG_ENGLISH, FALSE);

    endswitch;

    ComponentMoveData(MEDIA, nDisk, 0);

endprogram
```

```
// Source file: Is5fn626.rul
```

Language IDs

InstallShield provides language constants for all languages supported by Windows, however most of these constants are not supported for the designation of language specific file groups and language filtering.

InstallShield Language constants can be used in the following contexts:

- As the second parameter to the function ComponentFilterLanguage. In this context, the language constant specifies which files to filter or unfilter. Only supported language constants, which are listed in the table below, should be used for this purpose. Using an unsupported language constant for file group filtering has no effect since file groups designated for an unsupported language will be filtered (not included) during the media build and so cannot be installed.

  When using ComponentFilterLanguage, remember to add `#include "Sdlang.h"` before the program block. If you are using the Setup.rul generated by the Project Wizard, this line is already included for you. The file Sdlang.h defines these constants.

- As a return value from ComponentFileInfo when COMPONENT_INFO_LANGUAGE is passed as the nInfo parameter. As with ComponentFilterLanguage, only those constants listed in the table below should be used in this context.
- As the value returned in nvResult by function GetSystemInfo when it is called with the constant LANGUAGE in the nInfo parameter. In this context, any of the language constants listed in Sdlang.h can be returned because Windows supports all the language constants. Note however that if your installation includes language filtering based on these return values, you must use a switch statement to convert constants returned by this function into one of the constants supported for language filtering. For information about filtering language-dependent files based on the target system's language, click here.

For a complete list of language constants supported by InstallShield and their numeric equivalents refer to the file SdLang.h, which is located in the InstallShield Include folder.

When the language selection dialog is used by a multi-language installation to allow the user to select the installation language during setup initialization, the language dialog will display the Windows equivalent names listed in the table below. Because these names are generated by Windows, they will be localized to the version of Windows under which the setup is being run.

The table below shows the languages supported by the International West Version of InstallShield, their corresponding InstallShield language constants and equivalent language names used by English Windows to refer to those languages.

InstallShield Language	InstallShield Constant	English Windows 95 Equivalent	English Windows 3.x & NT 3.51/4.0 Equivalent
Basque	ISLANG_BASQUE	Basque	Not Supported
Catalan	ISLANG_CATALAN	Catalan	Catalan
Czech	ISLANG_CZECH	Czech	Czech
Danish	ISLANG_DANISH	Danish	Danish
Dutch	ISLANG_DUTCH	Dutch (Standard)	Dutch
English	ISLANG_ENGLISH	English (United States)	U.S. English
Finnish	ISLANG_FINNISH	Finnish	Finnish
French (Standard)	ISLANG_FRENCH_STANDARD	French (Standard)	French
French (Canadian)	ISLANG_FRENCH_CANADIAN	French (Canadian)	Canadian French

German	ISLANG_GERMAN	German (Standard)	German
Greek	ISLANG_GREEK	Greek	Greek
Hungarian	ISLANG_HUNGARIAN	Hungarian	Hungarian
Italian	ISLANG_ITALIAN	Itialian (Standard)	Italian
Norwegian	ISLANG_NORWEGIAN	Norwegian (Bokmal)	Norwgegian - Bokmal
Polish	ISLANG_POLISH	Polish	Polish
Portuguese (Brazilian)	ISLANG_PORTUGUESE_ BRAZILIAN	Portuguese (Brazilian)	Brazilian Portuguese
Portuguese (Standard)	ISLANG_PORTUGUESE_ STANDARD	Portuguese (Standard)	Portuguese
Slovak	ISLANG_SLOVAK	Slovak	Slovak
Slovenain	ISLANG_SLOVENIAN	Slovene	Not Supported
Spanish	ISLANG_SPANISH	Spanish (Traditional Sort)	Castilian Spanish
Swedish	ISLANG_SWEDISH	Swedish	Swedish

- InstallShield Language is the name used by the InstallShield IDE to refer to this language.
- InstallShield Constant is the language constant provided by InstallShield for filtering language specific file groups.
- English Windows 95 Equivalent is the name that English Windows 95 uses to refer to the language.
- English Windows NT Equivalent is the name that English Windows NT uses to refer to the language.
- Not Supported indicates that this platform does not support this language. If a setup includes a language not supported on the current platform, that language will be hidden in the setup language selection dialog to prevent the user from selecting an unsupported language.

ComponentFilterOS

Syntax

ComponentFilterOS (szMedia, nUpperOS, nLowerOS, bFiltered);

Description

The ComponentFilterOS function filters file groups that are flagged for specified operating systems. By default, no operating systems are filtered.

Parameters

szMedia

The name of the media.

nUpperOS

The upper 32 bits of a 64-bit operating system identifier field. Currently, nUpperOS is not used. Enter zero in this parameter. No other value is allowed.

nLowerOS

The lower 32 bits of a 64-bit operating system identifier field. nLowerOS specifies the operating system(s) you wish to filter. Choose from the following values. You can combine values using the bitwise OR operator (|).

ISOSL_ALL
ISOSL_WIN31
ISOSL_WIN95
ISOSL_NT351
ISOSL_NT351_ALPHA
ISOSL_NT351_MIPS
ISOSL_NT40
ISOSL_NT40_ALPHA
ISOSL_NT40_MIPS

bFiltered

Specifies whether or not you want to filter (exclude) the operating systems specified in nLowerOS. Use one of the following constants.

TRUE

Filter the specified operating system(s).

FALSE

Do not filter the specified operating system(s).

Return values

0

ComponentFilterOS was successful.

< 0

ComponentFilterOS failed. Call ComponentError for additional information.

{button ,JI(`LANGREF.HLP>Examples',`ComponentFilterOS_example')} Example

{button ,AL(`Script created component set vs. file media library;Specifying components in function calls;Componentfunctions',0,`',`')} See also

Operating system IDs

Use one of the following constants when calling ComponentFilterOS to specify operating system filtering. These are also the values returned when ComponentFileInfo is called with the COMPONENT_INFO_OS option.

ISOSL_ALL
ISOSL_WIN31
ISOSL_WIN95
ISOSL_NT351
ISOSL_NT351_ALPHA
ISOSL_NT351_MIPS
ISOSL_NT40
ISOSL_NT40_ALPHA
ISOSL_NT40_MIPS

ComponentFilterOS example

```
/*--------------------------------------------------------------------------*\
 * This example illustrates the use of ComponentFilterOS.
 *
 * To run this script, create a blank setup using the Project Wizard.
 * Specify Windows 95 and Windows NT 4.0 (Intel) for operating systems.
 * When you run the Media Build Wizard, you will be prompted to specify
 * which platforms to include in the build. Specify Windows 95 and
 * Windows NT 4.0 (Intel) for operating systems.
 *
\*--------------------------------------------------------------------------*/

// You can convert the following define statements into string table
// entries to localize your setup.
#define  COMPANY_NAME         "MultiLangOS Inc"
#define  PRODUCT_NAME         "MultiLangOS"
#define  PRODUCT_VERSION      "1.0"
#define  PROGRAMFOLDER        "MultiLangOS"
#define  PRODUCT_KEY          "Mlangos.exe"
#define  DEINST_KEY           "MultiLangOS"
#define  ASKDESTTITLE         "Destination Location"
#define  ASKDESTMSG           "Select a destination location."
#define  COMPERRTITLE         "Data Transfer Error Information"
#define  COMPERRMSG1          "ComponentError returned the following error."
#define  COMPERRMSG2          "Setup will now abort."
#define  COMPERRMSG3          "Media Name:"
#define  COMPERRMSG4          "Component:"
#define  COMPERRMSG5          "File Group:"
#define  COMPERRMSG6          "File:"
#define  COMPERRMSG7          "Error Number:"

prototype HandleComponentError(NUMBER);

STRING  svResult, svDir, svLogFile;
NUMBER  nvResult, nFilter, nvDisk;

program

    // Disable Back button, since it is not needed in this example.
    Disable(BACKBUTTON);
    // Set up uninstallation and get destination location.
    InstallationInfo(COMPANY_NAME, PRODUCT_NAME, PRODUCT_VERSION, PRODUCT_KEY);
    TARGETDIR = PROGRAMFILES ^ PROGRAMFOLDER;
    AskDestPath(ASKDESTTITLE, ASKDESTMSG, TARGETDIR , 0 );
    DeinstallStart(TARGETDIR, svLogFile, DEINST_KEY, 0);
    RegDBSetItem(REGDB_UNINSTALL_NAME, DEINST_KEY);

    // Get the operating system and set nFilter so ComponentFilterOS
    // can filter the operating system that is not present.
    GetSystemInfo(OS, nvResult, svResult);
    switch (nvResult)
        case IS_WINDOWSNT:
            GetSystemInfo(WINMAJOR, nvResult, svResult);
            if (nvResult = 4) then
                nFilter = ISOSL_WIN95;
            else
                MessageBox("Target system OS not supported.", SEVERE);
                abort;
            endif;
        case IS_WINDOWS95:
            nFilter = ISOSL_NT40;
        default:
            MessageBox("Target system OS not supported.", SEVERE);
            abort;
    endswitch;

    // Filter the operating system that is not present.
    ComponentFilterOS(MEDIA, 0, nFilter, TRUE);

    // Transfer files to target system.
    nvResult = ComponentMoveData (MEDIA, nvDisk, 0);
    if(nvResult < 0) then
        HandleComponentError(nvResult);
    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
    ComponentError( svMedia, svComponent, svFileGroup, svFile, nvError );
    SprintfBox( INFORMATION, COMPERRTITLE,
                COMPERRMSG1 + "\n" + COMPERRMSG2 + "\n\n" +
                COMPERRMSG3 + " %s\n" + COMPERRMSG4 + " %s\n" +
                COMPERRMSG5 + " %s\n" + COMPERRMSG6 + " %s\n" +
                COMPERRMSG7 + " %ld",
                svMedia, svComponent, svFileGroup, svFile, nvError );
            abort;
end;
```

```
// Source file: Is5fn627.rul
```

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
The name of the media.

szComponent
The name of the component whose information you want to retrieve. For more information about specifying components and subcomponents in function calls, click here.

nInfo
Type of information to retrieve.

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
When nInfo produces a number value, it is returned in nvResult.

svResult
When nInfo produces a string value, it is returned in svResult.

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

ComponentGetData 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
```

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
The name of the media.

szComponent
The name of the component whose size you want to retrieve. For more information about specifying components and subcomponents in function calls, click here.

nvSize
The size in bytes of the specified component is returned in this parameter. 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

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
The name of the media with the component whose selection setting you want to check.

szComponent
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

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;
        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
```

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 sub-component names is stored in listComponents. If szComponent has no children, listComponents will be an empty list.

Parameters

szMedia
The name of the media that contains the component whose subcomponents you want to list.

szComponent
The component whose subcomponents you want to list. Enter a null string ("") to list all top-level components. For more information about specifying components and subcomponents in function calls, click here**.**

listComponents
The name of a valid string list to contain the list of Components. Create the string list with ListCreate.

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

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%ld", nResult );
        nDone = AskYesNo( COMPTOTSIZEMSG1 + COMPTOTSIZEMSG2, YES );
    endwhile;

    ListDestroy( listCompList );

endprogram

#include "sddialog.rul"

// Source file: Is5fn021.rul
```

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

The media name of the file media library whose files you want to transfer with ComponentMoveData.

nvDisk

ComponentMoveData returns the number of the last disk it accessed in nvDisk. You do not need to initialize nvDisk before calling ComponentMoveData.

nReserved

Enter 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',0,`',`')}   See also

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;
        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
```

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

The media name of the file media library or script-created component set containing the component whose selection status you want to set.

szComponent

The component whose selection status you want to set. For more information about specifying components and subcomponents in function calls, click here.

bSelected

Specify whether the component is selected or deselected. The following constants are available:

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

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
```

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

The name of the media.

szComponent

The name of the component. For more information about specifying components and subcomponents in function calls, click here.

nInfo

The type of information to be set. Use one of the following constants:

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_SAMEORNEWERVERSION
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
When setting a numeric value, pass it in nData.

szData
When setting a string value, pass it in szData.

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

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
```

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

The name of the media.

szUserVar

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

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;Componentfunctions',0,`',`')} See also

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
```

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
The name of the media.

listSetupTypes
All setup types in the specified media will be returned in this list.

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

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
```

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

The name of the media.

szSetupType

The setup type name string exactly as it appears in the InstallShield IDE, for example, "Typical".

nInfo

Currently, the following data can be retrieved for the specified setup type:

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

Any return value of type NUMBER or LONG will be returned in nvResult.

svResult

Any return value of type STRING will be returned in svResult.

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

ComponentSetupTypeGetData 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
```

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
The name of the media.

szSetupType
The setup type you are setting.

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;Componentfunctions',0,`',`')}   See also

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
```

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

The name of the media.

szComponent

The name of the component whose size you wish to retrieve. Place a null string ("") here to retrieve the size of the entire media. For more information about specifying components and subcomponents in function calls, click here.

bIncludeSubcomp

Indicate whether or not to include selected subcomponents of szComponent. These constants are available:

TRUE

Include selected subcomponents in the size calculation.

FALSE

Do not include subcomponents in the size calculation.

bTargetSize

Indicate whether to retrieve the original/uncompressed size or the size in the media library.

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,`',`')} See also

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%ld", nResult );
        nDone = AskYesNo( COMPTOTSIZEMSG1 + COMPTOTSIZEMSG2, YES );
    endwhile;

    ListDestroy( listCompList );

endprogram

#include "sddialog.rul"

// Source file: Is5fn021.rul
```

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
The name of the media.

szComponent
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
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',0,`',`')} See also

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;
        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
```

SdSetupType

Syntax

SdSetupType (szTitle, szMsg, svDir, nReserved);

Description

The SdSetupType 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.

The dialog also displays a default destination path. A browse button launches a dialog box that allows the end user to change the destination path, either by entering a new folder name or by selecting an existing folder from a list. If the end user enters the name of a folder that does not exist, a message box appears asking whether to create the folder. If the end user clicks Yes, this function automatically creates the specified folder. The fully-qualified path of the specified folder is returned in svDir.

View sample dialog

When using SdSetup 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
Enter the text you want to appear in the title of the dialog box. To display the default title, "Setup Type," enter a null string ("").

szMsg
Enter the text you want to appear in the dialog box. This text is considered a static control. To display the default instructions for this dialog box, enter a null string ("").

svDir
Enter a default folder name. When SdSetupType returns, this parameter will contain the name of the selected folder.

nReserved
Reserved for future use. Enter 0 (zero) in nReserved.

Return values

TYPICAL
Indicates that the user selected Typical Setup.

COMPACT
Indicates that the user selected Compact Setup.

CUSTOM
Indicates that the user selected Custom Setup.

BACK
Indicates that the Back button was clicked.

{button ,JI(`LANGREF.HLP>Examples',`SdSetupType_example')} Example

{button ,AL(`SdAskOptions;SdProductName;SdSetupTypeEx',0,`',`')} See also

SdSetupType example

```
/*--------------------------------------------------------------------------*\
 *
 *  This example illustrates the use of SdSetupType.
 *
 *  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 SETUPTYPETITLE     "Choose Setup Type"
#define SETUPTYPEMSG       "Select a setup type."
#define COMPONENT          ""
#define SDCMPDLGTITLE      "Component Selection"
#define SDCMPDLGMSG        "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 setup type AND target location with SdSetupType.
    TARGETDIR = PROGRAMFILES ^ APPBASE_PATH;
    nResult = SdSetupType(SETUPTYPETITLE, SETUPTYPEMSG, TARGETDIR, 0);

    // If Custom setup type chosen, let user select components
    // and set change location if desired.
    if (nResult = CUSTOM) then
        SdComponentDialogAdv( SDCMPDLGTITLE, SDCMPDLGMSG,
                              TARGETDIR, COMPONENT);
    endif;

    // 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 );

    // 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: Is5fn148.rul
```

SdSetupTypeEx

Syntax

SdSetupTypeEx (szTitle, szMsg, szReserved, svSetupType, nReserved);

Description

The SdSetupTypeEx function displays a dialog box that enables the enables user to select the setup type when you specify setup types beyond Typical, Compact, and Custom.

View sample dialog

Parameters

szTitle
The title to display in the SdSetupTypeEx dialog box. To display the default title, "Setup Type," enter a null string ("").

szMsg
The message to be displayed in the SdSetupTypeEx dialog box. To display the default instructions for this dialog box, enter a null string ("").

szReserved
Enter a null string ("") in this parameter. No other value is allowed.

svSetupType
When calling this function, indicate a default setup type or enter a null string ("") to make the first setup in the list the default selection. On return, svSetupType contains the selected setup type. This string matches the name of the setup type specified in the IDE.

nReserved
Enter 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',0,`',`')} See also

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;
        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
```

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

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
The fully-qualified name of the driver you want 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
The name of the device driver relative to which you are adding szDriver.

nPosition
Use the following constants to indicate whether you want szDriver 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
```

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

The line of text you want to add to the system configuration file.

szRefKey

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

Use the following constants to indicate whether you want the line specified by szLine to be added before or after the line containing szRefKey.

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

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
```

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
The name of the parameter whose value you are retrieving.

nvValue
EzConfigGetValue returns the numeric value in nvValue.

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

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
```

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

The command whose value you want to change or add in the default system configuration file. If InstallShield cannot find the key, it is added for you.

nValue

The new value of szRefKey you want to set.

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

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
```

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.

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

The reference key (statement) you are adding to the system configuration file.

szValue

The value of the reference key you are adding to the system configuration file. A reference key is a word in the statement that uniquely defines the line. For example, in the statement DEVICE=C:\OS2\Kybrd.drv /1024 /C:345, the reference key is Kybrd.drv. In the following statement, the reference key is PATH: SET PATH=C:\OS2;C:\OS2\System\WinOS2

szRefKey

The reference key relative to which you are adding szKey in the system configuration file. If you enter a null string (""), the line is added as the first or last line in the file, depending on which nOptions constant you use.

nOptions

Specify whether you are adding the line before or after the line containing the reference key, or whether you are replacing an existing line. The following constants are available:

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

- When the ConfigAdd function replaces a line in a system configuration file, it compares the reference keys in the two lines.
- A reference key is a substring that uniquely defines the line. For example, in the statement DEVICE=C:\OS2\Kybrd.drv /1024 /C:345, the reference key is Kybrd.drv. In the statement SET PATH=C:\OS2;C:\OS2\System\WinOS2, the reference key is PATH.

{button ,JI(`LANGREF.HLP>Examples',`ConfigAdd_example')} Example

{button ,AL(`ConfigDelete;ConfigFileLoad;ConfigFileSave;ConfigFind;ConfigGetFileName;ConfigGetInt;ConfigMove;ConfigSetFileName;ConfigSetInt',0,`',`')} See also

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
```

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

The reference key that identifies the line or lines you want 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

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
```

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

The fully-qualified name of the system configuration file you are loading into memory. If you enter a null string (""), InstallShield loads the default system configuration file.

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

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
```

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

The backup filename to use to back up the original file you 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 enter a null string (""), 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

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
```

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 repeatedy 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

The reference key you are searching 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

ConfigFind returns the value of the reference key that was found in the system configuration file in svResult.

nOptions

Indicate whether to start the search from the beginning of the file, or to continue from where the previous search was terminated. This parameter can be set to these constants:

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

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
```

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

ConfigGetFileName returns the fully-qualified name of the default system configuration file in svFileName.

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

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
```

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
The reference key of the statement from which you want to retrieve the integer value.

nvValue
ConfigGetInt returns the integer value of the reference key in nvValue.

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

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
```

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
The line you are moving as a string.

szRefKey
The key that identifies the reference line used to position the line you are moving. The position of the line you are moving is determined by the value in the parameter nOptions.

nOptions
Specify whether you are moving the line specified by szMove before or after the line containing the reference key. The following constants are available:

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

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   = "DMDRVR.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
```

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 unnecssary 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
The fully-qualified name of the file you want to use 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

- The ConfigSetFileName function does not load a system configuration file into memory. You must use ConfigFileLoad to load a file into memory.
- 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

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
```

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
The reference keyword for the integer value you want to set.

nValue
The integer value you want to set.

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

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
```

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.

Many of the functions listed are for advanced developers who have significant Windows programming experience. Advanced functions are identified by the word "Advanced."

CmdGetHwndDlg
Retrieves the handle of a dialog. Advanced.

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.

CmdGetHwndDlg

Syntax

CmdGetHwndDlg (szDialogName);

Description

The CmdGetHwndDlg function retrieves the window handle of the dialog box identified by szDialogName. The dialog box already must have been defined with EzDefineDialog (or DefineDialog) and initialized by calling WaitOnDialog.

Parameters

szDialogName
The name of a valid dialog box as specified in the first parameter to EzDefineDialog (or DefineDialog).

Return values

> 0
The window handle of the dialog that was specified by szDialogName.

< 0
CmdGetHwndDlg was unable to retrieve the handle. Verify that szDialogName refers to a dialog box that has been properly defined and initialized.

Comments

- When a dialog box is initialized with the WaitOnDialog function, a window handle is assigned to it; that handle is associated with the dialog box only until it is closed by a call to EndDialog. If you call WaitOnDialog to open a dialog box that has been opened and closed previously in your script, you must call CmdGetHwndDlg again to get the new handle. The old handle is no longer valid.
- Normally, CmdGetHwndDlg is called in the DLG_INIT routine for a custom dialog box. The handle of the dialog box is assigned to a HWND variable to be used by other functions that need it.

{button ,JI(`LANGREF.HLP>Examples',`CmdGetHwndDlg_example')} Example

{button ,AL(`DefineDialog;EndDialog;EzDefineDialog;ReleaseDialog;WaitOnDialog',0,`',`')} See also

CmdGetHwndDlg example

```
/*--------------------------------------------------------------------------*\
 *
 * This example illustrates the usage of the CmdGetHwndDlg function.
 *
 * CmdGetHwndDlg is used in this example to retrieve the handle of the custom
 * dialog box.  This handle is then used by the SendMessage command to
 * maximize the dialog box when the Next push button is pressed.  The dialog
 * box is minimized when the Back button is pressed.
 *
 * NOTE: The 'custom' dialog used in this script is actually a dialog used
 *       by InstallShield, located in the _isres.dll file.  This .DLL file is
 *       already compressed in _sys1.cab.
 *
\*--------------------------------------------------------------------------*/

// Dialog box controls.
#define RES_DIALOG_ID        12027    // 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_PBUT_BACK           12    // ID of '< Back' push button

// Define Windows messages to InstallShield
#define  WM_SYSCOMMAND       0x0112
#define  SC_MINIMIZE         0xF020
#define  SC_MAXIMIZE         0xF030

   STRING szDialog, szDLL, svLine, svName, svCompName, svText;
   NUMBER nResult, nCmdValue, hwndDialog;
   BOOL bDone;

program

   szDialog = "ExDialog";
   szDLL = "";

   // szDLL is set to null, the _ISUSER.DLL and _ISRES.DLL files will be
   // searched.  This example 'custom' dialog is located in _ISRES.DLL.
   nResult = EzDefineDialog(szDialog, szDLL, "", RES_DIALOG_ID);

   if(nResult < 0) then

      MessageBox("Error in defining dialog", SEVERE);
      bDone = TRUE;
   else

      bDone = FALSE;
   endif;

   // Loop until bDone is true
   while (bDone = FALSE)

      nCmdValue = WaitOnDialog(szDialog);
      switch (nCmdValue)

         case DLG_ERR:

            bDone = TRUE;
         case DLG_INIT:
```

```
/*--------------------------------------------------------------------------*\
 *
 * CmdGetHwndDlg returns the handle of the custom dialog.
 *
\*--------------------------------------------------------------------------*/
            hwndDialog = CmdGetHwndDlg(szDialog);
         case DLG_CLOSE:

            bDone = TRUE;
         case RES_PBUT_NEXT:

            // Maximize the custom dialog box.
            SendMessage(hwndDialog, WM_SYSCOMMAND, SC_MAXIMIZE, 0);
         case RES_PBUT_CANCEL:

            bDone = TRUE;
         case RES_PBUT_BACK:

            // Minimize the custom dialog box.
            SendMessage(hwndDialog, WM_SYSCOMMAND, SC_MINIMIZE, 0);
      endswitch;
   endwhile;

   // Close the dialog box
   EndDialog(szDialog);

   // Free the dialog box from memory
   ReleaseDialog(szDialog);

endprogram

// Source file: Is5fn016.rul
```

CtrlClear

Syntax

CtrlClear (szDialogName, nControlID);

Description

The CtrlClear function clears the contents of various controls; it deletes the contents of a single- or multi-line edit field, static text field, single- or multi-selection list box, or the edit field of a combo box in a custom dialog box.

Parameters

szDialogName
The name of the dialog box that contains the control you want to delete.

nControlID
The control ID of a valid custom dialog box.

Return values

0
CtrlClear successfully deleted the contents of the specified control.

< 0
CtrlClear was unable to delete the contents of the dialog box.

{button ,JI(`LANGREF.HLP>Examples',`CtrlClear_example')}   Example

{button ,AL(`CtrlGetText;CtrlSetText',0,`',`')}   See also

CtrlClear example

```
/*--------------------------------------------------------------------------*\
 *
 * This example illustrates the usage of the CtrlClear, CtrlGetText, and
 * CtrlSetText functions.
 *
 * CtrlSetText is called when the dialog is initialized to set the edit boxes
 * in the custom dialog to the constants.  When the Back button is pressed,
 * the edit boxes are all cleared by calling CtrlClear three times.  Once the
 * Next button is pressed, the CtrlGetText is called three times to retrieve
 * the text in each edit box.  Once the while loop has completed, the text in
 * each edit box is then displayed individually.
 *
 * NOTE: The 'custom' dialog used in this script is actually a dialog used
 *       by InstallShield, located in the _isres.dll file.  This .DLL file is
 *       already compressed in _sys1.cab.
 *
\*--------------------------------------------------------------------------*/

#define USER_NAME    "Your Name"
#define COMPANY_NAME "Your Company"
#define SERIAL_NUM   "123"

   #define RES_DIALOG_ID     12002   // ID of Dialog Itself
   #define RES_EDIT_NAME       301    // ID of the User Name edit box
   #define RES_EDIT_COMPANY    302   // ID of the Company Name edit box.
   #define RES_EDIT_SERIAL     303
   #define RES_PBUT_NEXT         1   // ID of 'Next >' push button
   #define RES_PBUT_CANCEL       9   // ID of 'Cancel' push button
   #define RES_PBUT_BACK        12   // ID of '< Back' push button

   STRING szDialogName, szDLL, svLine, svName, svCompName, szText, svText1;
   STRING svText2, svText3;
   NUMBER nResult, nCmdValue, nControlID;
   BOOL bDone;

program

   // Define a dialog box.
   szDialogName   = "ExDialog";

   // szDLL is set to null, the _ISUSER.DLL and _ISRES.DLL files will be
   // searched.  This example 'custom' dialog is located in _ISRES.DLL.
   szDLL      = "";
   nResult    = EzDefineDialog (szDialogName, szDLL, "", RES_DIALOG_ID);

   if (nResult < 0) then

      MessageBox ("Error in defining dialog", WARNING);
      bDone = TRUE;
   else

      bDone = FALSE;
   endif;

   while (bDone = FALSE)

      // Display dialog and wait for user's input
      nCmdValue = WaitOnDialog (szDialogName);
      switch (nCmdValue)

         case DLG_ERR:

            MessageBox ("Dialog Failed", SEVERE);
            bDone = TRUE;
         case DLG_INIT:

/*--------------------------------------------------------------------------*\
 *
 * The following sets the edit fields in the custom dialog box to constants.
 *
\*--------------------------------------------------------------------------*/
            szText     = USER_NAME;
            nControlID = RES_EDIT_NAME;
            CtrlSetText (szDialogName, nControlID, szText);

            szText     = COMPANY_NAME;
            nControlID = RES_EDIT_COMPANY;
            CtrlSetText (szDialogName, nControlID, szText);

            szText     = SERIAL_NUM;
            nControlID = RES_EDIT_SERIAL;
            CtrlSetText (szDialogName, nControlID, szText);
         case RES_PBUT_BACK:
/*--------------------------------------------------------------------------*\
 *
 * When the Back button is pressed, all edit boxes are cleared.
 *
\*--------------------------------------------------------------------------*/
            nControlID = RES_EDIT_NAME;
            CtrlClear(szDialogName, nControlID);

            nControlID = RES_EDIT_COMPANY;
            CtrlClear(szDialogName, nControlID);

            nControlID = RES_EDIT_SERIAL;
            CtrlClear(szDialogName, nControlID);
         case RES_PBUT_CANCEL:

            bDone = TRUE;
         case RES_PBUT_NEXT:
/*--------------------------------------------------------------------------*\
 *
 * Retrieve the text from edit fields.
 *
\*--------------------------------------------------------------------------*/
            nControlID = RES_EDIT_NAME;
            CtrlGetText (szDialogName, nControlID, svText1);

            nControlID = RES_EDIT_COMPANY;
            CtrlGetText (szDialogName, nControlID, svText2);

            nControlID = RES_EDIT_SERIAL;
            CtrlGetText (szDialogName, nControlID, svText3);
            bDone = TRUE;
      endswitch;
   endwhile;

   // Display each the text in each field individually.
   SprintfBox(INFORMATION, "Name", "%s", svText1);
   SprintfBox(INFORMATION, "Company", "%s", svText2);

   SprintfBox(INFORMATION, "Serial", "%s", svText3);

   // Close the dialog box
   EndDialog (szDialogName);

   // Free the dialog box from memory
   ReleaseDialog (szDialogName);

endprogram

// Source file: Is5fn038.rul
```

CtrlDir

Syntax

CtrlDir (szDialogName, nControlID, szDir, nItems);

Description

The CtrlDir function fills a list box or a combo box control with a file listing that matches the specified path or filename in szDir. You can include names of files, subdirectories, and disk drives in the listing. The CtrlDir function works only with custom dialog boxes.

Parameters

szDialogName
The name of a valid dialog box.

nControlID
The resource ID of the list box or combo box control.

szDir
The fully-qualified path or filename, which may include wild card characters.

nItems
Specify the type of listing you want to display in the control. The following constants can be used individually or combined with the bitwise OR operator (|) to include more than one type of element:

DLG_DIR_FILE
Creates a list of files matching the file specification szDir.

DLG_DIR_DIRECTORY
Creates a list of subdirectories that exist in the path specification szDir.

DLG_DIR_DRIVE
Creates a list of drives.

Return values

0
CtrlDir successfully filled the specified control in a dialog box.

< 0
CtrlDir was unable to fill the specified control.

{button ,JI(`LANGREF.HLP>Examples',`CtrlDir_example')} Example

{button ,AL(`DefineDialog;EndDialog;ReleaseDialog;WaitOnDialog',0,`',`')} See also

CtrlDir example

```
/*--------------------------------------------------------------------------*\
 *
 * This example illustrates the usage of the CtrlDir function.
 *
 * When the dialog box is initialized, CtrlDir is called to return all files
 * of a path to a listbox in a custom dialog box.  When the Next button is
 * pressed, all directories of the same path are returned to the listbox.
 *
 * NOTE: The 'custom' dialog used in this script is actually a dialog used
 *       by InstallShield, located in the _isres.dll file.  This .DLL file is
 *       already compressed in _sys1.cab.
 *
\*--------------------------------------------------------------------------*/

   // Set constants for the dialog box controls.
   #define RES_DIALOG_ID      10204    // 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_PBUT_BROWSE       31    // ID of 'Browse' push button.
   #define RES_PBUT_BACK         12    // ID of 'Back' push button.
   #define RES_DIALOG_LISTBOX   501    // ID of List box.

   STRING  szDialogName, szDLL, szDir, szMsg, szDefPath, svResultPath;
   NUMBER  nResult, nCmdValue, nControlID, nItems;
   BOOL    bDone;

program

  start:

   Disable(BACKBUTTON);

   szDialogName = "ExDialog";
   szDLL        = "";

   // szDLL is set to null, the _ISUSER.DLL and _ISRES.DLL files will be
   // searched.  This dialog is located in _ISRES.DLL.
   nResult  = EzDefineDialog(szDialogName, szDLL, "", RES_DIALOG_ID);

   if(nResult < 0) then

      MessageBox("Error in defining dialog", SEVERE);
      abort;
   else

      bDone = FALSE;
   endif;

   szMsg = "Please enter a valid path:";
   AskPath(szMsg, szDefPath, svResultPath);
   szDir = svResultPath ^ "*.*";

   // Loop the following until bDone is set to TRUE.
   while (bDone = FALSE)

      // Display the dialog and return user's input.
      nCmdValue = WaitOnDialog(szDialogName);
      switch (nCmdValue)

         case DLG_CLOSE:

            bDone = TRUE;
         case DLG_ERR:

            MessageBox("Dialog failed", SEVERE);
            bDone = TRUE;
         case DLG_INIT:

            // Set CtrlDir variables.
            nControlID = RES_DIALOG_LISTBOX;
            nItems     = DLG_DIR_FILE;
/*--------------------------------------------------------------------------*\
 *
 * Return all filenames in the specified path to the listbox.
 *
\*--------------------------------------------------------------------------*/
            if (CtrlDir(szDialogName, nControlID, szDir, nItems) < 0) then

               MessageBox("CtrlDir failed.", SEVERE);
            else

               MessageBox("Filenames successfuly retrieved.", INFORMATION);
            endif;

         case RES_PBUT_CANCEL:

            bDone = TRUE;
         case RES_PBUT_NEXT:

            nControlID = RES_DIALOG_LISTBOX;
            nItems     = DLG_DIR_DIRECTORY;
/*--------------------------------------------------------------------------*\
 *
 * Return all directory names in the specified path to the listbox.
 *
\*--------------------------------------------------------------------------*/
            if (CtrlDir(szDialogName, nControlID, szDir, nItems) < 0) then

               MessageBox("CtrlDir failed.", SEVERE);
            else

               szMsg = "Directory names successfully retrieved.";
               MessageBox(szMsg, INFORMATION);
            endif;
         case RES_PBUT_BACK:

            if(AskYesNo("Exit example?", YES) = YES) then

               bDone = TRUE;
            endif;
         case RES_PBUT_BROWSE:

            szMsg = "Press the Next button to display directories, Back " +
                    "button to exit.";
            MessageBox(szMsg, INFORMATION);
      endswitch;
   endwhile;

   // Close the dialog box
   EndDialog(szDialogName);
   // Free the dialog box from memory
   ReleaseDialog(szDialogName);

   if (AskYesNo("Would you like to redo the example?", YES) = YES) then

      goto start;
   endif;

endprogram

// Source file: Is5fn039.rul
```

CtrlGetCurSel

Syntax

CtrlGetCurSel (szDialogName, nControlID, svText);

Description

The CtrlGetCurSel function retrieves the currently selected item from a single selection list box or combo box control in a custom dialog box. Call CtrlGetMultCurSel to retrieve items from multi-selection list boxes.

Parameters

szDialogName
The name of a valid custom dialog box that contains the item you want to retrieve.

nControlID
The resource ID of the single selection list box or combo box control.

svText
CtrlGetCurSel returns the currently selected item from the control in svText.

Return values

0
CtrlGetCurSel successfully retrieved the currently selected item from the dialog box.

< 0
CtrlGetCurSel was unable to retrieve the selected item.

{button ,JI(`LANGREF.HLP>Examples',`CtrlGetCurSel_example')} Example

{button ,AL(`CtrlSetCurSel;DefineDialog;EndDialog;ReleaseDialog;WaitOnDialog',0,`',`')} See also

CtrlGetCurSel example

```
/*--------------------------------------------------------------------------*\
 *
 * This example illustrates the usage of the CtrlGetCurSel and CtrlSetCurSel
 * functions
 *
 * After the dialog box is displayed, CtrlSetCurSel is called to set the
 * current selection to the ROOT_FILE file.
 *
 * CtrlGetCurSel is called every time a new current selection is selected from
 * within the list box.  The result of CtrlGetCurSel is displayed in a message
 * box every time the function is called.
 *
 * NOTE: The 'custom' dialog used in this script is actually a dialog used
 *       by InstallShield, located in the _isres.dll file.  This .DLL file is
 *       already compressed in _sys1.cab.
 *
 *       Also note, in order for this script to run properly, it is advised to
 *       set the VALID_DIR and SELECTED_FILE constants to a valid directory,
 *       and a valid file within that directory.
 *
\*--------------------------------------------------------------------------*/

#define VALID_DIR     "C:\\"
#define SELECTED_FILE "AUTOEXEC.BAT"

   // These are the dialog box controls.
   #define RES_DIALOG_ID       10204    // ID of Dialog Itself
   #define RES_PBUT_NEXT           1    // ID of OK push button
   #define RES_PBUT_CANCEL         9    // ID of 'Cancel' push button
   #define RES_PBUT_BACK          12    // ID of 'Help' push button
   #define RES_DIALOG_LISTBOX    501    // ID of List box

   STRING  szDialogName, szDLL, svText, szDir, szText, szDefPath;
   STRING  svResultPath;
   NUMBER  nResult, nCmdValue, nControlID, nItems;
   BOOL    bDone;

program

   // Set up 'custom' dialog.
   szDialogName = "ExDialog";
   szDLL        = "";
   // szDLL is set to null, the _ISUSER.DLL and _ISRES.DLL files will be
   // searched.  This dialog box is located in _ISRES.DLL.
   nResult      = EzDefineDialog(szDialogName, szDLL, "", RES_DIALOG_ID);

   if(nResult < 0) then

      MessageBox("Error in defining dialog", WARNING);
      bDone = TRUE;
   else

      bDone = FALSE;
   endif;

   szDefPath = VALID_DIR;
   AskPath("Please enter a valid path:", szDefPath, svResultPath);
   szDir = svResultPath ^ "*.*";
   // Loop WaitOnDialog until bDone is set to TRUE.
   while (bDone = FALSE)

      // Display this custom dialog box.
      nCmdValue = WaitOnDialog(szDialogName);
      switch (nCmdValue)

         case DLG_ERR:

            MessageBox("Dialog Failed", SEVERE);
            bDone = TRUE;
         case DLG_INIT:

            // Setup CtrlDir variables.
            nControlID = RES_DIALOG_LISTBOX;
            nItems     = DLG_DIR_FILE;

            // Retrieves and display files of the path in a list box.
            CtrlDir(szDialogName, nControlID, szDir, nItems);

/*--------------------------------------------------------------------------*\
 *
 * The following sets szText to be the highlighted default element.
 *
\*--------------------------------------------------------------------------*/
            szText = SELECTED_FILE;
            CtrlSetCurSel(szDialogName, nControlID, szText);
         case RES_PBUT_NEXT:

            bDone = TRUE;
         case RES_PBUT_CANCEL:

            bDone = TRUE;
         case RES_DIALOG_LISTBOX:

            nControlID = RES_DIALOG_LISTBOX;
/*--------------------------------------------------------------------------*\
 *
 * The following retrieves the current selection in the list box and sets it
 * to the svText variable.
 *
\*--------------------------------------------------------------------------*/
            CtrlGetCurSel(szDialogName, nControlID, svText);

            // Display the svText variable for every time it is retrieved..
            MessageBox(svText, INFORMATION);
         case DLG_CLOSE:

            bDone = TRUE;
      endswitch;
   endwhile;

   // Close the custom dialog box.
   EndDialog(szDialogName);

   // Remove the custom dialog box from memory.
   ReleaseDialog(szDialogName);

endprogram

// Source file: Is5fn040.rul
```

CtrlGetMLEText

Syntax

CtrlGetMLEText (szDialogName, nControlID, listID);

Description

The CtrlGetMLEText function retrieves the contents of a multi-line edit field control in a custom dialog box. InstallShield places each line of the multi-line edit field into a valid string list identified by listID. Call CtrlGetText to retrieve the contents of a single line edit field control.

Parameters

szDialogName
The name of a valid custom dialog box that contains the multi-line edit control whose contents you want to retrieve.

nControlID
The resource ID of the multi-line edit control.

listID
The name of a valid string list where you want to place the lines of the edit field.

Return values

0
CtrlGetMLEText successfully retrieved the contents of a multi-line edit field.

< 0
CtrlGetMLEText was unable to retrieve the contents of the control.

{button ,JI(`LANGREF.HLP>Examples',`CtrlGetMLEText_example')} Example

{button ,AL(`DefineDialog;EndDialog;ListCreate;ListDestroy;ListGetFirstString;ListGetNextString;ListReadFromFile;ListWriteToFile;ReleaseDialog;WaitOnDialog',0,`',`')} See also

CtrlGetMLEText example

```
/*--------------------------------------------------------------------------*\
 *
 * This example illustrates the usage of the CtrlSetMLEText and CtrlGetMLEText
 * functions.
 *
 * This script first retrieves the names of all program folders on the target
 * system and places them in a list.  When the dialog is displayed and
 * initialized, the CtrlSetMLEText function sets this list to be displayed in
 * the edit box.
 *
 * This list is then destroyed.  A new list is created when the Next button is
 * pressed.  CtrlGetMLEText then retrieves the elements in the edit box, and
 * assigns them to this new string list.  This list is then displayed in an
 * Sd dialog box.
 *
 * NOTE: The 'custom' dialog used in this script is actually a dialog used
 *       by InstallShield, located in the _isres.dll file.  This .DLL file is
 *       already compressed in _sys1.cab.
 *
 *       The GetGroupNameList function used in this example may return an
 *       error if the target system is running under a shell other than
 *       the Windows 95 shell or Program manager.
 *
\*--------------------------------------------------------------------------*/

   #include "Sddialog.h";

   // Dialog box controls.
   #define RES_DIALOG_ID        12007   // 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_PBUT_BACK           12   // ID of '< Back' push button
   #define RES_DIALOG_EDITBOX     301   // ID of edit box.

   STRING szDialogName, szDLL, szTitle, szMsg;
   NUMBER nCmdValue, nResult, nControlID;
   BOOL bDone;
   LIST listID, listFolders;

program

   Disable(BACKBUTTON);

   szDialogName = "CtrlSetMLEText";
   szDLL        = "";

   // szDLL is set to null, the _ISUSER.DLL and _ISRES.DLL files will be
   // searched.  This example 'custom' dialog is located in _ISRES.DLL.
   nResult = EzDefineDialog(szDialogName, szDLL, "", RES_DIALOG_ID);

   if (nResult < 0) then

      MessageBox("Error in defining dialog", SEVERE);
      bDone = TRUE;
   else

      bDone = FALSE;
   endif;
   // Create the listID string list.
   listID = ListCreate(STRINGLIST);
   MessageBox("listID created.", INFORMATION);

   if (listID = LIST_NULL) then

      MessageBox("Unable to create list.", SEVERE);
   endif;

   // Retrieve the program folder names into a list.
   GetGroupNameList(listID);

   while (bDone = FALSE)

      // Display the dialog and wait for a command.
      nCmdValue = WaitOnDialog(szDialogName);
      switch (nCmdValue)

         case DLG_ERR:

            MessageBox("Dialog failed", SEVERE);
            bDone = TRUE;
         case DLG_INIT:

/*--------------------------------------------------------------------------*\
 *
 * CtrlSetMLEText sets the RES_DIALOG_EDITBOX edit box to the list of the
 * program folders.
 *
\*--------------------------------------------------------------------------*/
            nControlID = RES_DIALOG_EDITBOX;
            if (CtrlSetMLEText(szDialogName, nControlID, listID) < 0) then

               MessageBox("CtrlSetMLEText failed.", SEVERE);
               bDone = TRUE;
            endif;

            // Destroy the listID string list.
            ListDestroy(listID);

            MessageBox("listID destroyed.", INFORMATION);
         case DLG_CLOSE:

            bDone = TRUE;
         case RES_PBUT_NEXT:

            // Create the listFolders string list.
            listFolders = ListCreate(STRINGLIST);

            if (listFolders = LIST_NULL) then

               MessageBox("Unable to create list.", SEVERE);
            endif;

            MessageBox("listFolders created.", INFORMATION);
/*--------------------------------------------------------------------------*\
 *
 * CtrlGetMLEText retrieves the elements in the RES_DIALOG_EDITBOX edit box,
 * and assigns these elements into the newly created listFolders string list.
 *
\*--------------------------------------------------------------------------*/
            nControlID = RES_DIALOG_EDITBOX;
            if (CtrlGetMLEText(szDialogName, nControlID,
                               listFolders) < 0) then

               MessageBox("CtrlGetMLEText failed.", SEVERE);
            else

               MessageBox("CtrlGetMLEText successful.", INFORMATION);
            endif;

            bDone = TRUE;
         case RES_PBUT_BACK:

            bDone = TRUE;
         case RES_PBUT_CANCEL:

            bDone = TRUE;
      endswitch;
   endwhile;

   szTitle = "CtrlGetMLEText & CtrlSetMLEText";
   szMsg   = "The following are the elements retrieved from the editbox:";

   // Display the list of elements retrieved.
   SdShowInfoList(szTitle, szMsg, listFolders);

   // Remove listFolders string list from memory.
   ListDestroy(listFolders);

   MessageBox("listFolders destroyed.", INFORMATION);

   // Close the dialog.
   EndDialog(szDialogName);

   // Remove dialog from memory.
   ReleaseDialog(szDialogName);

endprogram

   #define SD_SINGLE_DIALOGS 1
   #define SD_SHOWINFOLIST   1

   #include "Sddialog.rul"

// Source file: Is5fn041.rul
```

CtrlGetMultCurSel

Syntax

CtrlGetMultCurSel (szDialogName, nControlID, listID);

Description

The CtrlGetMultCurSel function retrieves the currently selected lines from a multi-selection list box control. Each selected line of the multi-selection list box is placed into a string list identified by listID. To retrieve selected text from a single selection list box control, call CtrlGetCurSel. CtrlGetMultCurSel is for use only with custom dialog boxes.

Parameters

szDialogName
The name of a valid custom dialog box that contains the list box control whose contents you want to retrieve.

nControlID
The resource ID of the multi-line edit control.

listID
The name of a valid string list where you want to place the lines of the list box.

Return values

0
CtrlGetMultCurSel successfully retrieved the currently selected items.

< 0
CtrlGetMultCurSel was unable to retrieve the items.

Comments

The style of a control determines its behavior. For example, an auto radio button control behaves differently than a manual radio button control. Verify that you are using the desired style for each control.

{button ,JI(`LANGREF.HLP>Examples',`CtrlGetMultCurSel_example')} Example

{button ,AL(`DefineDialog;EndDialog;ListCreate;ListDestroy;ListGetFirstString;ListGetNextString;ReleaseDialog;WaitOnDialog',0,`',`')} See also

CtrlGetMultCurSel example

```
/*--------------------------------------------------------------------------*\
 *
 * This example illustrates the usage of the CtrlSetMultCurSel and
 * CtrlGetMultCurSel functions.
 *
 * This script first retrieves the names of all program folders on the target
 * system and places them in a list.  When the dialog is initialized, the
 * CtrlSetList function sets this list to be displayed in the list box.  The
 * CtrlSetMultCurSel function is then called to highlight the user-selected
 * folder.
 *
 * This list is then destroyed.  A new list is created when the Next button is
 * pressed.  CtrlGetMultCurSel then retrieves the elements in the list box, and
 * assigns them to this new string list.  This list is then displayed in an
 * Sd dialog box.
 *
 * NOTE: The 'custom' dialog used in this script is actually a dialog used
 *       by InstallShield, located in the _isres.dll file.  This .DLL file is
 *       already compressed in _sys1.cab.
 *
 *       The GetGroupNameList function used in this example may return an
 *       error if the target system is running under a shell other than
 *       the Windows 95 shell or Program manager.
 *
\*--------------------------------------------------------------------------*/

   #include "Sddialog.h";

   // Dialog box controls.
   #define RES_DIALOG_ID        12013   // 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_PBUT_BACK           12   // ID of '< Back' push button
   #define RES_DIALOG_LISTBOX     401   // ID of list box.

   STRING szDialogName, szDLL, szTitle, szMsg;
   STRING szText, szDefFolder, svResultFolder;
   NUMBER nCmdValue, nResult, nControlID, nSelectFlag;
   BOOL bDone;
   LIST listID, listFolders;

program

   Disable(BACKBUTTON);

   szDialogName = "CtrlSetMultCurSel";
   szDLL        = "";

   // szDLL is set to null, the _ISUSER.DLL and _ISRES.DLL files will be
   // searched.  This example 'custom' dialog is located in _ISRES.DLL.
   nResult = EzDefineDialog(szDialogName, szDLL, "", RES_DIALOG_ID);

   if (nResult < 0) then

      MessageBox("Error in defining dialog", SEVERE);
      bDone = TRUE;
   else
```

```
      bDone = FALSE;
   endif;

   // Create the listID string list.
   listID = ListCreate(STRINGLIST);

   if (listID = LIST_NULL) then

      MessageBox("Unable to create list.", SEVERE);
   else

      MessageBox("listID created.", INFORMATION);
   endif;

   // Retrieve the program folder names into a list.
   GetGroupNameList(listID);

   // Retrieve a folder name from the user.
   szTitle = "CtrlGetMultCurSel & CtrlSetMultCurSel";
   SelectFolder(szTitle, szDefFolder, svResultFolder);

   while (bDone = FALSE)

      // Display the dialog and wait for a command.
      nCmdValue = WaitOnDialog(szDialogName);
      switch (nCmdValue)

         case DLG_ERR:

            MessageBox("Dialog failed", SEVERE);
            bDone = TRUE;
         case DLG_INIT:

            // The following sets the listbox to the list of program folders.
            nControlID = RES_DIALOG_LISTBOX;
            CtrlSetList(szDialogName, nControlID, listID);

            szText = svResultFolder;
            nSelectFlag = TRUE;
/*--------------------------------------------------------------------------*\
 *
 * CtrlSetMultCurSel is called to set the user-selected folder to be
 * highlighted.
 *
\*--------------------------------------------------------------------------*/
            if (CtrlSetMultCurSel(szDialogName, nControlID, szText,
                                  nSelectFlag) < 0) then

               MessageBox("CtrlSetMultCurSel failed.", SEVERE);
            endif;

            // Destroy the listID string list.
            ListDestroy(listID);
            MessageBox("listID destroyed.", INFORMATION);
         case DLG_CLOSE:

            bDone = TRUE;
         case RES_PBUT_NEXT:

            // Create the listFolders string list.
            listFolders = ListCreate(STRINGLIST);

            if (listFolders = LIST_NULL) then
```

```
              MessageBox("Unable to create list.", SEVERE);
           else

              MessageBox("listFolders created.", INFORMATION);
           endif;
/*--------------------------------------------------------------------------*\
 *
 * CtrlGetMultCurSel retrieves the highlighted elements in the listbox, and
 * asigns these elements into the newly created listFolders string list.
 *
\*--------------------------------------------------------------------------*/
           if (CtrlGetMultCurSel(szDialogName, nControlID,
                                 listFolders) < 0) then

              MessageBox("CtrlGetMultCurSel failed.", SEVERE);
           else

              MessageBox("CtrlGetMultCurSel successful.", INFORMATION);
           endif;

           bDone = TRUE;
        case RES_PBUT_BACK:

           bDone = TRUE;
        case RES_PBUT_CANCEL:

           bDone = TRUE;
     endswitch;
  endwhile;

  szMsg   = "The following are the elements highlighted in the listbox:";

  // Display the list of elements highlighted.
  SdShowInfoList(szTitle, szMsg, listFolders);

  // Remove listFolders string list from memory.
  ListDestroy(listFolders);
  MessageBox("listFolders destroyed.", INFORMATION);

  // Close the dialog.
  EndDialog(szDialogName);

  // Remove dialog from memory.
  ReleaseDialog(szDialogName);

endprogram

  #define SD_SINGLE_DIALOGS 1
  #define SD_SHOWINFOLIST   1

  #include "Sddialog.rul"

// Source file: Is5fn042.rul
```

CtrlGetState

Syntax

CtrlGetState (szDialogName, nControlID);

Description

The CtrlGetState function gets the current state of a check box or radio button control from a custom dialog box.

Parameters

szDialogName
The name of the dialog box that contains the control.

nControlID
The resource ID of the check box or radio button control whose state you want to retrieve.

Return values

BUTTON_CHECKED
The check box or radio button is selected.

BUTTON_UNCHECKED
The check box or radio button is not selected.

DLG_ERR
CtrlGetState was unable to determine the state of the control.

{button ,JI(`LANGREF.HLP>Examples',`CtrlGetState_example')} Example

{button ,AL(`CtrlSetState;DefineDialog;EndDialog;ReleaseDialog;WaitOnDialog',0,`',`')} See also

CtrlGetState example

```
/*--------------------------------------------------------------------------*\
 *
 * This example illustrates the usage of the CtrlGetState and CtrlSetState
 * functions.
 *
 * CtrlSetState is called to initialize the radio and check buttons on the
 * custom dialog box.
 *
 * After the dialog box has been closed, CtrlGetState is called in succession
 * to display messages indicating the choices the user had made.
 *
 * NOTE: The 'custom' dialog used in this script is actually a dialog used
 *       by InstallShield, located in the _isres.dll file.  This .DLL file is
 *       already compressed in _sys1.cab.
 *
\*--------------------------------------------------------------------------*/

// Set the dialog box controls.
#define  RES_DIALOG_ID       12020    // 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_PBUT_BACK          12    // ID of '< Back' push button.
#define  ID_OP1_CHECK          501    // ID of Option 1 check button.
#define  ID_OP2_CHECK          502    // ID of Option 2 check button.
#define  ID_OP3_CHECK          503    // ID of Option 3 check button.
#define  ID_OP4_CHECK          504    // ID of Option 4 check button.

   STRING  szDialogName, szDLL, szProgram, szTitle, szMsg;
   NUMBER  nResult, nCmdValue, nControlID, nState;
   BOOL    bDone;

program

   szDialogName = "ExDialog";
   szDLL        = "";

   // szDLL is set to null, EzDefineDialog searches the _ISUSER.DLL and
   // _ISRES.DLL files.  This example 'custom' dialog is located in _ISRES.DLL.
   nResult  = EzDefineDialog(szDialogName, szDLL, "", RES_DIALOG_ID);

   if(nResult < 0) then

      MessageBox("Error in defining dialog", WARNING);
      bDone = TRUE;
   else

      bDone = FALSE;
   endif;

   while (bDone = FALSE)

      // Display the dialog box and wait for a command.
      nCmdValue = WaitOnDialog(szDialogName);
      switch (nCmdValue)

         case DLG_ERR:

            MessageBox("Dialog failed", SEVERE);
            bDone = TRUE;
         case DLG_INIT:

/*--------------------------------------------------------------------------*\
 *
 * Initialize the dialog box by setting the Option 1 and Option 2 radio
 * buttons to checked.
 *
\*--------------------------------------------------------------------------*/
            nControlID = ID_OP1_CHECK;
            nState     = BUTTON_CHECKED;
            if (CtrlSetState(szDialogName, nControlID, nState) < 0) then

               MessageBox("First call to CtrlSetState failed.", SEVERE);
            endif;

            nControlID = ID_OP2_CHECK;
            if (CtrlSetState(szDialogName, nControlID, nState) < 0) then

               MessageBox("Second call to CtrlSetState failed.", SEVERE);
            endif;
         case RES_PBUT_NEXT:

            bDone = TRUE;
         case RES_PBUT_CANCEL:

            bDone = TRUE;
         case RES_PBUT_BACK:

            bDone = TRUE;
      endswitch;
   endwhile;

   szTitle = "CtrlGetState & CtrlSetState Example";
   szMsg   = "You selected the following:";
   SprintfBox(INFORMATION, szTitle, szMsg);

/*--------------------------------------------------------------------------*\
 *
 * Call CtrlGetState to retrieve the state of a button.  If the button is
 * checked, display the operating system the selected button represents.
 *
\*--------------------------------------------------------------------------*/
   nControlID = ID_OP1_CHECK;
   if (CtrlGetState(szDialogName, nControlID) = BUTTON_CHECKED) then

      MessageBox("Option 1", INFORMATION);
   endif;

   nControlID = ID_OP2_CHECK;
   if (CtrlGetState(szDialogName, nControlID) = BUTTON_CHECKED) then

      MessageBox("Option 2", INFORMATION);
   endif;

   nControlID = ID_OP3_CHECK;
   if (CtrlGetState(szDialogName, nControlID) = BUTTON_CHECKED) then

      MessageBox("Option 3", INFORMATION);
   endif;

   nControlID = ID_OP4_CHECK;
   if (CtrlGetState(szDialogName, nControlID) = BUTTON_CHECKED) then
```

```
      MessageBox("Option 4", INFORMATION);
   endif;

   EndDialog(szDialogName);
   ReleaseDialog(szDialogName);

endprogram

// Source file: Is5fn043.rul
```

CtrlGetSubCommand

Syntax

CtrlGetSubCommand (szDialogName);

Description

The CtrlGetSubCommand function retrieves the action performed on a control in a custom dialog box. For example, CtrlGetSubCommand can tell you if the user single-clicked or double-clicked a list box or combo box control. It can also tell you when the contents of an edit field have changed.

Advanced developers can call CmdGetHwndDlg to handle additional information.

Parameters

szDialogName
The name of a valid custom dialog box.

Return values

LISTBOX_ENTER
The user double-clicked a list box item.

LISTBOX_SELECT
The user single-clicked a list box item.

EDITBOX_CHANGE
The contents of the edit box have changed.

{button ,JI(`LANGREF.HLP>Examples',`CtrlGetSubCommand_example')} Example

{button ,AL(`CmdGetHwndDlg;DefineDialog;EndDialog;EzDefineDialog;ReleaseDialog;WaitOnDialog',0,`',`')} See also

CtrlGetSubCommand example

```
/*--------------------------------------------------------------------------*\
 *
 * This example illustrates the usage of the CtrlGetSubCommand function.
 *
 * The following script calls CtrlGetSubCommand to return all of the
 * subcommands for a control edit or list box.
 *
 * NOTE: The 'custom' dialog used in this script is actually a dialog used
 *       by InstallShield, located in the _isres.dll file.  This .DLL file is
 *       already compressed in _sys1.cab.
 *
 *       Also note the VALID_DRIVE constant must be set to a valid path on the
 *       target system.
 *
\*--------------------------------------------------------------------------*/

#define VALID_DRIVE "C:\\*.*"

   // Dialog box controls.
   #define RES_DIALOG_ID      12008    // ID of Dialog Itself
   #define RES_PBUT_NEXT          1    // ID of OK push button
   #define RES_PBUT_CANCEL        9    // ID of 'Cancel' push button
   #define RES_PBUT_BACK         12    // ID of '< Back' push button
   #define RES_DIALOG_LISTBOX   401    // ID of List box
   #define RES_DIALOG_EDITBOX   301    // ID of Edit box

   STRING szDialog, szDLL, svText;
   NUMBER nResult, nCmdValue, nSubCommand;
   BOOL bDone;

program

   szDialog = "ExDialog";
   szDLL    = "";
   // szDLL is set to null, the _ISUSER.DLL and _ISRES.DLL files will be
   // searched.  This dialog is located in _ISRES.DLL.
   nResult  = EzDefineDialog(szDialog, szDLL, "", RES_DIALOG_ID);

   if(nResult < 0) then

      MessageBox("Error in defining dialog", SEVERE);
      bDone = TRUE;
   else

      bDone = FALSE;
   endif;

   // Loop the following until bDone is set to TRUE.
   while (bDone = FALSE)

      nCmdValue = WaitOnDialog(szDialog);
      switch (nCmdValue)

         case DLG_ERR:

            MessageBox("Dialog Failed", SEVERE);
            bDone = TRUE;
         case DLG_INIT:
            // Retrieve the files in the VALID_DRIVE path and set it to the
            // list box.
            CtrlDir(szDialog, RES_DIALOG_LISTBOX, VALID_DRIVE, DLG_DIR_FILE);
         case RES_PBUT_NEXT:

            bDone = TRUE;
         case RES_PBUT_CANCEL:

            bDone = TRUE;
         case RES_DIALOG_LISTBOX:

            nSubCommand = CtrlGetSubCommand(szDialog);

            if (nSubCommand = LISTBOX_SELECT) then

               MessageBox("single click.", INFORMATION);
            elseif (nSubCommand = LISTBOX_ENTER) then

               MessageBox("double click.", INFORMATION);
            endif;
         case RES_DIALOG_EDITBOX:

            nSubCommand = CtrlGetSubCommand(szDialog);

            if (nSubCommand = EDITBOX_CHANGE) then

               MessageBox("Editbox changed", SEVERE);
            endif;
      endswitch;
   endwhile;

   EndDialog(szDialog);
   ReleaseDialog(szDialog);

endprogram

// Source file: Is5fn044.rul
```

CtrlGetText

Syntax

CtrlGetText (szDialogName, nControlID, svText);

Description

The CtrlGetText function retrieves the text from an edit field, static text field, or button control of a custom dialog box. To retrieve the text from multi-line edit field controls, call CtrlGetMLEText.

Parameters

szDialogName
The name of a valid dialog box that contains the field or control whose text you want to retrieve.

nControlID
The resource ID of the edit field, static text field, or push button control.

svText
CtrlGetText returns the text from the control or field in svText.

Return values

0
CtrlGetText successfully retrieved the contents of the control.

< 0
CtrlGetText was unable to retrieve the contents.

{button ,JI(`LANGREF.HLP>Examples',`CtrlGetText_example')}Example

{button ,AL(`CtrlClear;CtrlSetText;DefineDialog;EndDialog;ReleaseDialog;WaitOnDialog',0,`',`')} See also

CtrlGetText example

```
/*--------------------------------------------------------------------------*\
 *
 * This example illustrates the usage of the CtrlSetText and CtrlGetText
 * functions.
 *
 * When the dialog box is initialized, CtrlSetText is used to set text in the
 * dialog box.  When the Next button is pressed, CtrlGetText is called to
 * retrieve the name and company strings from the dialog box.  These strings
 * are then displayed in message boxes.
 *
 * NOTE: The 'custom' dialog used in this script is actually a dialog used
 *       by InstallShield, located in the _isres.dll file.  This .DLL file is
 *       already compressed in _sys1.cab.
 *
\*--------------------------------------------------------------------------*/

#define NAME    "Your Name"
#define COMPANY "Your Company"

   #define RES_DIALOG_ID       12001 // ID of Dialog Itself
   #define RES_PBUT_NEXT           1 // ID of 'Next >' push button
   #define RES_PBUT_BACK          12 // ID of '< Back' push button
   #define RES_PBUT_CANCEL         9 // ID of 'Cancel' push button
   #define RES_EDITBOX_NAME      301 // ID of the User Name edit box
   #define RES_EDITBOX_COMPANY   302 // ID of the Company Name edit box.

   STRING szDialogName, szDLL, szText, svText;
   NUMBER nResult, nCmdValue, nControlID;
   BOOL   bDone;

program

   // Define a dialog box.
   szDialogName = "ExDialog";
   szDLL = "";

   nResult  = EzDefineDialog(szDialogName, szDLL, "", RES_DIALOG_ID);

   if(nResult < 0) then

      MessageBox("Error in defining dialog", SEVERE);
      bDone = TRUE;
   else

      bDone = FALSE;
   endif;

Register:

   while (bDone = FALSE)

      // Display dialog and wait for user's input
      nCmdValue = WaitOnDialog(szDialogName);

      switch (nCmdValue)

         case DLG_ERR:
            MessageBox("Dialog Failed", SEVERE);
            bDone = TRUE;
         case DLG_INIT:

/*--------------------------------------------------------------------------*\
 *
 * CtrlSetText sets the text of the edit box to the NAME and COMPANY
 * preprocessor string constants.
 *
\*--------------------------------------------------------------------------*/
            nControlID = RES_EDITBOX_NAME;
            szText     = NAME;
            if (CtrlSetText(szDialogName, nControlID, szText) < 0) then

               MessageBox("First call to CtrlSetText failed.", SEVERE);
               bDone = TRUE;
            endif;

            nControlID = RES_EDITBOX_COMPANY;
            szText     = COMPANY;
            if (CtrlSetText(szDialogName, nControlID, szText) < 0) then

               MessageBox("Second call to CtrlSetText failed.", SEVERE);
               bDone = TRUE;
            endif;
         case RES_PBUT_BACK:

            bDone = TRUE;
         case RES_PBUT_CANCEL:

            bDone = TRUE;
         case RES_PBUT_NEXT:

            bDone = TRUE;
      endswitch;
   endwhile;

/*--------------------------------------------------------------------------*\
 *
 * CtrlGetText is called to retrieve the values in the edit boxes.
 *
\*--------------------------------------------------------------------------*/
   nControlID = RES_EDITBOX_NAME;
   if (CtrlGetText(szDialogName, nControlID, svText) < 0) then

      MessageBox("First call to CtrlGetText failed.", SEVERE);
      bDone = TRUE;
   endif;

   // Display text retrieved.
   MessageBox(svText, INFORMATION);

   nControlID = RES_EDITBOX_COMPANY;
   if (CtrlGetText(szDialogName, nControlID, svText) < 0) then

      MessageBox("Second call to CtrlGetText failed.", SEVERE);
      bDone = TRUE;
   endif;

   MessageBox(svText, INFORMATION);

   // Close the dialog box
   EndDialog(szDialogName);
   // Free the dialog box from memory
   ReleaseDialog(szDialogName);

endprogram

// Source file: Is5fn045.rul
```

CtrlPGroups

Syntax

CtrlPGroups (szDialogName, nControlID);

Description

The CtrlPGroups function places a list of existing program folders in a list box or combo box control. This function is for use only with custom dialog boxes.

Parameters

szDialogName
The name of a valid custom dialog box that contains the control you want to use.

nControlID
The resource ID of a valid list box or combo box control.

Return values

0
CtrlPGroups successfully placed the specified list of program folders in the control.

< 0
CtrlPGroups was unable to place the specified list of program folders in the control.

{button ,JI(`LANGREF.HLP>Examples',`CtrlPGroups_example')} Example

{button ,AL(`DefineDialog;EndDialog;EzDefineDialog;ReleaseDialog;WaitOnDialog',0,`',`')} See also

CtrlPGroups example

```
/*--------------------------------------------------------------------------*\
 *
 * This example illustrates the usage of the CtrlPGroups function.
 *
 * This function displays a list of program groups in the listbox.
 *
 * NOTE: The 'custom' dialog used in this script is actually a dialog used
 *       by InstallShield, located in the _isres.dll file.  This .DLL file is
 *       already compressed in _sys1.cab.
 *
\*--------------------------------------------------------------------------*/

   // These are the dialog box controls.
   #define RES_DIALOG_ID     10204    // 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_PBUT_BACK        12    // ID of '< Back' push button
   #define RES_DIALOG_LISTBOX  501    // ID of List box

   STRING szDialogName, szDLL;
   NUMBER nResult, nCmdValue, nControlID;
   BOOL   bDone;

program

   // Define a dialog box.
   szDialogName = "ExDialog";
   szDLL        = "";

   // szDLL is set to null, the _ISUSER.DLL and _ISRES.DLL files will be
   // searched.  This example 'custom' dialog is located in _ISRES.DLL.
   nResult  = EzDefineDialog(szDialogName, szDLL, "", RES_DIALOG_ID);

   if(nResult < 0) then

      MessageBox("Error in defining dialog", WARNING);
      bDone = TRUE;
   else

      bDone = FALSE;
   endif;

   while (bDone = FALSE)

      // Display dialog and wait for user's input
      nCmdValue = WaitOnDialog(szDialogName);

      switch (nCmdValue)

         case DLG_ERR:

            MessageBox("Dialog Failed", SEVERE);
            bDone = TRUE;
         case DLG_INIT:

/*--------------------------------------------------------------------------*\
 *
 * CtrlPGroups will place a list of folders into the dialog list box.
 *
\*-----------------------------------------------------------------------------*/
            nControlID = RES_DIALOG_LISTBOX;
            if (CtrlPGroups(szDialogName, nControlID) < 0) then

               MessageBox("CtrlPGroups failed.", SEVERE);
            endif;
         case DLG_CLOSE:

            bDone = TRUE;
         case RES_PBUT_CANCEL:

            bDone = TRUE;
         case RES_PBUT_NEXT:

            bDone = TRUE;
         case RES_PBUT_BACK:

            bDone = TRUE;
      endswitch;
   endwhile;

   // Close the dialog box
   EndDialog(szDialogName);

   // Free the dialog box from memory
   ReleaseDialog(szDialogName);

endprogram

// Source file: Is5fn046.rul
```

CtrlSelectText

Syntax

CtrlSelectText (szDialogName, nControlID);

Description

The CtrlSelectText function selects all the text in an edit field or the edit field of a combo box. If the control is a multi-line edit field, this function selects all the text on all lines. This function is for use only with custom dialog boxes.

Parameters

szDialogName
The name of a valid dialog box that contains the edit field you want to select.

nControlID
The resource ID of the edit field or combo box control you want to select.

Return values

0
CtrlSelectText successfully selected all the text in the field.

< 0
CtrlSelectText was unable to select the text.

{button ,JI(`LANGREF.HLP>Examples',`CtrlSelectText_example')} Example

{button ,AL(`CtrlGetText;CtrlSetText;DefineDialog;EndDialog;ReleaseDialog;WaitOnDialog',0,`',`')} See also

CtrlSelectText example

```
/*--------------------------------------------------------------------------*\
 *
 * This example illustrates the usage of the CtrlSelectText.
 *
 * When the dialog box is initialized, CtrlSetText is called to set the edit
 * box to the string the user entered.  CtrlSelectText is then called to
 * highlight this string in the edit box.
 *
 * NOTE: The 'custom' dialog used in this script is actually a dialog used
 *       by InstallShield, located in the _isres.dll file.  This .DLL file is
 *       already compressed in _sys1.cab.
 *
\*--------------------------------------------------------------------------*/

   // Dialog box controls.
   #define RES_DIALOG_ID      12017  // 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_PBUT_BACK         12  // ID of '< Back' push button.
   #define RES_EDITBOX          301  // ID of edit box.

   STRING szDialogName, szDLL, szDir, szText, szDefault;
   NUMBER nResult, nCmdValue, nControlID;
   BOOL   bDone;

program

   szDialogName = "ExDialog";
   szDLL        = "";

   // szDLL is set to null, the _ISUSER.DLL and _ISRES.DLL files will be
   // searched.  This example 'custom' dialog is located in _ISRES.DLL.
   nResult      = EzDefineDialog(szDialogName, szDLL, "", RES_DIALOG_ID);

   if (nResult < 0) then

      MessageBox("Error in defining dialog", WARNING);
      bDone = TRUE;
   else

      bDone = FALSE;
   endif;

   // Ask user to enter any string of text.
   szDefault = "any line of text:";
   AskText("Please enter any line of text:", szDefault, szText);

   while (bDone = FALSE)

      // Display the dialog and wait for user commands.
      nCmdValue = WaitOnDialog(szDialogName);
      switch (nCmdValue)

         case DLG_ERR:

            MessageBox("Dialog Failed", SEVERE);
            bDone = TRUE;
         case DLG_INIT:
            nControlID = RES_EDITBOX;
            // Set the edit box to the string the user entered.
            CtrlSetText(szDialogName, nControlID, szText);

/*--------------------------------------------------------------------------*\
 *
 * CtrlSelectText is called to highlight the edit box when the dialog box is
 * initialized.
 *
\*--------------------------------------------------------------------------*/
            CtrlSelectText(szDialogName, nControlID);
         case DLG_CLOSE:

            bDone = TRUE;
         case RES_PBUT_NEXT:

            bDone = TRUE;
         case RES_PBUT_CANCEL:

            bDone = TRUE;
         case RES_PBUT_BACK:

            bDone = TRUE;
      endswitch;
   endwhile;

   // Close the dialog box.
   EndDialog(szDialogName);

   // Remove the dialog box from memory.
   ReleaseDialog(szDialogName);

endprogram

// Source file: Is5fn047.rul
```

CtrlSetCurSel

Syntax

CtrlSetCurSel (szDialogName, nControlID, szText);

Description

The CtrlSetCurSel function searches the specified list or combo box control for a string. If found, CtrlSetCurSel selects (highlights) the item. Call CtrlSetMultCurSel for multi-selection list box and combo box controls. The CtrlSetCurSel function is for use only with custom dialog boxes.

Parameters

szDialogName
The name of a valid custom dialog box that contains the control you want to find.

nControlID
The resource ID of the control that contains the string you are searching for.

szText
The string you are searching for. If CtrlSetCurSel finds the item, it is selected (highlighted).

Return values

0
CtrlSetCurSel successfully found and selected the specified string.

< 0
CtrlSetCurSel was unable to find and select the specified string.

{button ,JI(`LANGREF.HLP>Examples',`CtrlSetCurSel_example')} Example

{button ,AL(`CtrlGetCurSel;DefineDialog;EndDialog;ReleaseDialog;WaitOnDialog',0,`',`')} See also

CtrlSetCurSel example

```
/*--------------------------------------------------------------------------*\
 *
 * This example illustrates the usage of the CtrlGetCurSel and CtrlSetCurSel
 * functions
 *
 * After the dialog box is displayed, CtrlSetCurSel is called to set the
 * current selection to the ROOT_FILE file.
 *
 * CtrlGetCurSel is called every time a new current selection is selected from
 * within the list box.  The result of CtrlGetCurSel is displayed in a message
 * box every time the function is called.
 *
 * NOTE: The 'custom' dialog used in this script is actually a dialog used
 *       by InstallShield, located in the _isres.dll file.  This .DLL file is
 *       already compressed in _sys1.cab.
 *
 *       Also note, in order for this script to run properly, it is advised to
 *       set the VALID_DIR and SELECTED_FILE constants to a valid directory,
 *       and a valid file within that directory.
 *
\*--------------------------------------------------------------------------*/

#define VALID_DIR     "C:\\"
#define SELECTED_FILE "AUTOEXEC.BAT"

   // These are the dialog box controls.
   #define RES_DIALOG_ID       10204    // ID of Dialog Itself
   #define RES_PBUT_NEXT           1    // ID of OK push button
   #define RES_PBUT_CANCEL         9    // ID of 'Cancel' push button
   #define RES_PBUT_BACK          12    // ID of 'Help' push button
   #define RES_DIALOG_LISTBOX    501    // ID of List box

   STRING  szDialogName, szDLL, svText, szDir, szText, szDefPath;
   STRING  svResultPath;
   NUMBER  nResult, nCmdValue, nControlID, nItems;
   BOOL    bDone;

program

   // Set up 'custom' dialog.
   szDialogName = "ExDialog";
   szDLL        = "";
   // szDLL is set to null, the _ISUSER.DLL and _ISRES.DLL files will be
   // searched.  This dialog box is located in _ISRES.DLL.
   nResult      = EzDefineDialog(szDialogName, szDLL, "", RES_DIALOG_ID);

   if(nResult < 0) then

      MessageBox("Error in defining dialog", WARNING);
      bDone = TRUE;
   else

      bDone = FALSE;
   endif;

   szDefPath = VALID_DIR;
   AskPath("Please enter a valid path:", szDefPath, svResultPath);
   szDir = svResultPath ^ "*.*";
   // Loop WaitOnDialog until bDone is set to TRUE.
   while (bDone = FALSE)

      // Display this custom dialog box.
      nCmdValue = WaitOnDialog(szDialogName);
      switch (nCmdValue)

         case DLG_ERR:

            MessageBox("Dialog Failed", SEVERE);
            bDone = TRUE;
         case DLG_INIT:

            // Setup CtrlDir variables.
            nControlID = RES_DIALOG_LISTBOX;
            nItems     = DLG_DIR_FILE;

            // Retrieves and display files of the path in a list box.
            CtrlDir(szDialogName, nControlID, szDir, nItems);

/*--------------------------------------------------------------------------*\
 *
 * The following sets szText to be the highlighted default element.
 *
\*--------------------------------------------------------------------------*/
            szText = SELECTED_FILE;
            CtrlSetCurSel(szDialogName, nControlID, szText);
         case RES_PBUT_NEXT:

            bDone = TRUE;
         case RES_PBUT_CANCEL:

            bDone = TRUE;
         case RES_DIALOG_LISTBOX:

            nControlID = RES_DIALOG_LISTBOX;
/*--------------------------------------------------------------------------*\
 *
 * The following retrieves the current selection in the list box and sets it
 * to the svText variable.
 *
\*--------------------------------------------------------------------------*/
            CtrlGetCurSel(szDialogName, nControlID, svText);

            // Display the svText variable for every time it is retrieved..
            MessageBox(svText, INFORMATION);
         case DLG_CLOSE:

            bDone = TRUE;
      endswitch;
   endwhile;

   // Close the custom dialog box.
   EndDialog(szDialogName);

   // Remove the custom dialog box from memory.
   ReleaseDialog(szDialogName);

endprogram

// Source file: Is5fn040.rul
```

CtrlSetFont

Syntax

CtrlSetFont (szDialogName, hFont, nControlID);

Description

The CtrlSetFont function specifies a font for a control in a custom dialog box. Call this function from within the DLG_INIT routine of the dialog box message processing loop.

Parameters

szDialogName
The name of a valid dialog box.

hFont
A valid handle to a font created by GetFont.

nControlID
The resource ID of the control in the dialog box whose font you want to set. To set the font for all the controls in the dialog box, enter ALLCONTROLS in this parameter.

Return values

0
CtrlSetFont successfully set the requested font in a dialog box.

< 0
CtrlSetFont was unable to set the font in the requested dialog box.

{button ,JI(`LANGREF.HLP>Examples',`CtrlSetFont_example')}Example

{button ,AL(`GetFont',0,`',`')} See also

CtrlSetFont 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
```

CtrlSetList

Syntax

CtrlSetList (szDialogName, nControlID, listID);

Description

The CtrlSetList function places the contents of a string list into the specified single- or multi-selection list box or combo box control. Any pre-existing contents are replaced with the items contained in listID. InstallShield places each element of the string list into each element of the list box or combo box control.

Parameters

szDialogName
The name of a valid dialog box that contains the list box or combo box.

nControlID
The resource ID of the list box or combo box.

listID
The name of a valid string list that contains the elements you want to copy into the list box or combo box control.

Return values

0
CtrlSetList successfully placed the contents of the string list into the control.

< 0
CtrlSetList was unable to place the contents of the string list into the control.

{button ,JI(`LANGREF.HLP>Examples',`CtrlSetList_example')} Example

{button ,AL(`DefineDialog;EndDialog;ListAddItem;ListAddString;ListCreate;ListDeleteItem;ListDeleteString;ListDestroy;ReleaseDialog;WaitOnDialog',0,`',`')} See also

CtrlSetList example

```
/*--------------------------------------------------------------------------*\
 *
 * This example illustrates the usage of the CtrlSetList function.
 *
 * The following script calls CtrlSetList to show how to display a list of
 * items in a multi-selection list box.
 *
 * NOTE: The 'custom' dialog used in this script is actually a dialog used
 *       by InstallShield, located in the _isres.dll file.  This .DLL file is
 *       already compressed in _sys1.cab.
 *
\*--------------------------------------------------------------------------*/

   // Set the dialog box controls.
   #define RES_DIALOG_ID      10207 // ID of Dialog Itself
   #define RES_PBUT_NEXT          1 // ID of 'OK' push button
   #define RES_PBUT_CANCEL        9 // ID of 'Cancel' push button
   #define RES_PBUT_BACK         12 // ID of 'Help'   push button
   #define RES_DIALOG_LISTBOX   502 // ID of List box

   STRING szDialogName, szDLL;
   NUMBER nResult, nCmdValue, nControlID;
   BOOL   bDone;
   LIST   listID;

program

   // Create a list for use in CtrlGetMultCurSel.
   listID   = ListCreate(STRINGLIST);

   if (listID = NULL) then

      MessageBox("Unable to create list.", SEVERE);
     abort;
   endif;

   // Fill the list with some strings.
   ListAddString(listID, "First string", AFTER);
   ListAddString(listID, "Second string", AFTER);

   // szDLL is set to null, the _ISUSER.DLL and _ISRES.DLL files will be
   // searched.  This dialog is located in _ISRES.DLL.
   szDialogName = "CtrlSetListEx";
   szDLL        = "";

   // Define the custom dialog.
   nResult  = EzDefineDialog(szDialogName, szDLL, "", RES_DIALOG_ID);

   if(nResult < 0) then

      MessageBox("Error in defining dialog", WARNING);
      bDone = TRUE;
   else

      bDone = FALSE;
   endif;

   while (bDone = FALSE)
```

```
      nCmdValue = WaitOnDialog(szDialogName);
      switch (nCmdValue)

         case DLG_ERR:

            MessageBox("Dialog failed", SEVERE);
            bDone = TRUE;
         case DLG_INIT:

            nControlID = RES_DIALOG_LISTBOX;
/*--------------------------------------------------------------------------*\
 *
 * The following returns the list of strings into the list box when the dialog
 * initializes.
 *
\*--------------------------------------------------------------------------*/
            if (CtrlSetList(szDialogName, nControlID, listID) < 0) then

               MessageBox("CtrlSetList failed.", SEVERE);
               bDone = TRUE;
            endif;

         case DLG_CLOSE:

            bDone = TRUE;
         case RES_PBUT_NEXT:

            bDone = TRUE;
         case RES_PBUT_CANCEL:

            bDone = TRUE;
         case RES_PBUT_BACK:

            bDone = TRUE;
      endswitch;
   endwhile;

   // Remove list from memory.
   ListDestroy(listID);

   // Close Dialog.
   EndDialog(szDialogName);

   // Remove dialog from memory.
   ReleaseDialog(szDialogName);

endprogram

// Source file: Is5fn050.rul
```

CtrlSetMLEText

Syntax

CtrlSetMLEText (szDialogName, nControlID, listID);

Description

The CtrlSetMLEText function sets the text of a multi-line edit box control. InstallShield separately places each string in listID into the multi-line edit box control. This function is for use only with custom dialog boxes.

Parameters

szDialogName
The name of a valid dialog box.

nControlID
The resource ID of the multi-line edit box control in the dialog box.

listID
The name of a valid string list that contains the elements you want to copy into the multi-line edit control.

Return values

0
CtrlSetMLEText set the text into the control.

< 0
CtrlSetMLEText was unable to set the text in the control.

{button ,JI(`LANGREF.HLP>Examples',`CtrlSetMLEText_example')} Example

{button ,AL(`CtrlGetMLEText;DefineDialog;EndDialog;ReleaseDialog',0,`',`')} See also

CtrlSetMLEText example

```
/*--------------------------------------------------------------------------*\
 *
 * This example illustrates the usage of the CtrlSetMLEText and CtrlGetMLEText
 * functions.
 *
 * This script first retrieves the names of all program folders on the target
 * system and places them in a list.  When the dialog is displayed and
 * initialized, the CtrlSetMLEText function sets this list to be displayed in
 * the edit box.
 *
 * This list is then destroyed.  A new list is created when the Next button is
 * pressed.  CtrlGetMLEText then retrieves the elements in the edit box, and
 * assigns them to this new string list.  This list is then displayed in an
 * Sd dialog box.
 *
 * NOTE: The 'custom' dialog used in this script is actually a dialog used
 *       by InstallShield, located in the _isres.dll file.  This .DLL file is
 *       already compressed in _sys1.cab.
 *
 *       The GetGroupNameList function used in this example may return an
 *       error if the target system is running under a shell other than
 *       the Windows 95 shell or Program manager.
 *
\*--------------------------------------------------------------------------*/

   #include "Sddialog.h";

   // Dialog box controls.
   #define RES_DIALOG_ID        12007   // 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_PBUT_BACK           12   // ID of '< Back' push button
   #define RES_DIALOG_EDITBOX     301   // ID of edit box.

   STRING szDialogName, szDLL, szTitle, szMsg;
   NUMBER nCmdValue, nResult, nControlID;
   BOOL bDone;
   LIST listID, listFolders;

program

   Disable(BACKBUTTON);

   szDialogName = "CtrlSetMLEText";
   szDLL        = "";

   // szDLL is set to null, the _ISUSER.DLL and _ISRES.DLL files will be
   // searched.  This example 'custom' dialog is located in _ISRES.DLL.
   nResult = EzDefineDialog(szDialogName, szDLL, "", RES_DIALOG_ID);

   if (nResult < 0) then

      MessageBox("Error in defining dialog", SEVERE);
      bDone = TRUE;
   else

      bDone = FALSE;
   endif;
   // Create the listID string list.
   listID = ListCreate(STRINGLIST);
   MessageBox("listID created.", INFORMATION);

   if (listID = LIST_NULL) then

      MessageBox("Unable to create list.", SEVERE);
   endif;

   // Retrieve the program folder names into a list.
   GetGroupNameList(listID);

   while (bDone = FALSE)

      // Display the dialog and wait for a command.
      nCmdValue = WaitOnDialog(szDialogName);
      switch (nCmdValue)

         case DLG_ERR:

            MessageBox("Dialog failed", SEVERE);
            bDone = TRUE;
         case DLG_INIT:

/*--------------------------------------------------------------------------*\
 *
 * CtrlSetMLEText sets the RES_DIALOG_EDITBOX edit box to the list of the
 * program folders.
 *
\*--------------------------------------------------------------------------*/
            nControlID = RES_DIALOG_EDITBOX;
            if (CtrlSetMLEText(szDialogName, nControlID, listID) < 0) then

               MessageBox("CtrlSetMLEText failed.", SEVERE);
               bDone = TRUE;
            endif;

            // Destroy the listID string list.
            ListDestroy(listID);

            MessageBox("listID destroyed.", INFORMATION);
         case DLG_CLOSE:

            bDone = TRUE;
         case RES_PBUT_NEXT:

            // Create the listFolders string list.
            listFolders = ListCreate(STRINGLIST);

            if (listFolders = LIST_NULL) then

               MessageBox("Unable to create list.", SEVERE);
            endif;

            MessageBox("listFolders created.", INFORMATION);
/*--------------------------------------------------------------------------*\
 *
 * CtrlGetMLEText retrieves the elements in the RES_DIALOG_EDITBOX edit box,
 * and assigns these elements into the newly created listFolders string list.
 *
\*--------------------------------------------------------------------------*/
            nControlID = RES_DIALOG_EDITBOX;
            if (CtrlGetMLEText(szDialogName, nControlID,
                               listFolders) < 0) then

               MessageBox("CtrlGetMLEText failed.", SEVERE);
            else

               MessageBox("CtrlGetMLEText successful.", INFORMATION);
            endif;

            bDone = TRUE;
         case RES_PBUT_BACK:

            bDone = TRUE;
         case RES_PBUT_CANCEL:

            bDone = TRUE;
      endswitch;
   endwhile;

   szTitle = "CtrlGetMLEText & CtrlSetMLEText";
   szMsg   = "The following are the elements retrieved from the editbox:";

   // Display the list of elements retrieved.
   SdShowInfoList(szTitle, szMsg, listFolders);

   // Remove listFolders string list from memory.
   ListDestroy(listFolders);

   MessageBox("listFolders destroyed.", INFORMATION);

   // Close the dialog.
   EndDialog(szDialogName);

   // Remove dialog from memory.
   ReleaseDialog(szDialogName);

endprogram

   #define SD_SINGLE_DIALOGS 1
   #define SD_SHOWINFOLIST   1

   #include "Sddialog.rul"

// Source file: Is5fn041.rul
```

CtrlSetMultCurSel

Syntax

CtrlSetMultCurSel (szDialogName, nControlID, szText, nSelectFlag);

Description

The CtrlSetMultCurSel function searches the specified multi-selection list or combo box control. If nSelectFlag is set to TRUE, CtrlSetMultCurSel selects (highlights) the item when it is found. This function is for use only with custom dialog boxes.

Parameters

szDialogName
The name of a valid custom dialog box.

nControlID
The resource ID of the multi-selection list box control in the dialog box.

szText
The string you are searching for.

nSelectFlag
Either TRUE or FALSE, indicating whether you want to highlight an item when CtrlSetMultCurSel finds it. TRUE indicates you want the item highlighted. FALSE indicates you do not want the item highlighted.

Return values

0
CtrlSetMultCurSel successfully found the text in the control and highlighted it or not, as indicated in nSelectFlag.

< 0
CtrlSetMultCurSel was unable to find the text in the control.

{button ,JI(`LANGREF.HLP>Examples',`CtrlSetMultCurSel_example')} Example

{button ,AL(`DefineDialog;EndDialog;ListAddItem;ListAddString;ListCreate;ListDeleteItem;ListDeleteString;ListDestroy;WaitOnDialog',0,`',`')} See also

CtrlSetMultCurSel example

```
/*--------------------------------------------------------------------------*\
 *
 * This example illustrates the usage of the CtrlSetMultCurSel and
 * CtrlGetMultCurSel functions.
 *
 * This script first retrieves the names of all program folders on the target
 * system and places them in a list.  When the dialog is initialized, the
 * CtrlSetList function sets this list to be displayed in the list box.  The
 * CtrlSetMultCurSel function is then called to highlight the user-selected
 * folder.
 *
 * This list is then destroyed.  A new list is created when the Next button is
 * pressed.  CtrlGetMultCurSel then retrieves the elements in the list box, and
 * assigns them to this new string list.  This list is then displayed in an
 * Sd dialog box.
 *
 * NOTE: The 'custom' dialog used in this script is actually a dialog used
 *       by InstallShield, located in the _isres.dll file.  This .DLL file is
 *       already compressed in _sys1.cab.
 *
 *       The GetGroupNameList function used in this example may return an
 *       error if the target system is running under a shell other than
 *       the Windows 95 shell or Program manager.
 *
\*--------------------------------------------------------------------------*/

   #include "Sddialog.h";

   // Dialog box controls.
   #define RES_DIALOG_ID        12013   // 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_PBUT_BACK           12   // ID of '< Back' push button
   #define RES_DIALOG_LISTBOX     401   // ID of list box.

   STRING szDialogName, szDLL, szTitle, szMsg;
   STRING szText, szDefFolder, svResultFolder;
   NUMBER nCmdValue, nResult, nControlID, nSelectFlag;
   BOOL bDone;
   LIST listID, listFolders;

program

   Disable(BACKBUTTON);

   szDialogName = "CtrlSetMultCurSel";
   szDLL        = "";

   // szDLL is set to null, the _ISUSER.DLL and _ISRES.DLL files will be
   // searched.  This example 'custom' dialog is located in _ISRES.DLL.
   nResult = EzDefineDialog(szDialogName, szDLL, "", RES_DIALOG_ID);

   if (nResult < 0) then

      MessageBox("Error in defining dialog", SEVERE);
      bDone = TRUE;
   else
```

```
      bDone = FALSE;
   endif;

   // Create the listID string list.
   listID = ListCreate(STRINGLIST);

   if (listID = LIST_NULL) then

      MessageBox("Unable to create list.", SEVERE);
   else

      MessageBox("listID created.", INFORMATION);
   endif;

   // Retrieve the program folder names into a list.
   GetGroupNameList(listID);

   // Retrieve a folder name from the user.
   szTitle = "CtrlGetMultCurSel & CtrlSetMultCurSel";
   SelectFolder(szTitle, szDefFolder, svResultFolder);

   while (bDone = FALSE)

      // Display the dialog and wait for a command.
      nCmdValue = WaitOnDialog(szDialogName);
      switch (nCmdValue)

         case DLG_ERR:

            MessageBox("Dialog failed", SEVERE);
            bDone = TRUE;
         case DLG_INIT:

            // The following sets the listbox to the list of program folders.
            nControlID = RES_DIALOG_LISTBOX;
            CtrlSetList(szDialogName, nControlID, listID);

            szText = svResultFolder;
            nSelectFlag = TRUE;
/*--------------------------------------------------------------------------*\
 *
 * CtrlSetMultCurSel is called to set the user-selected folder to be
 * highlighted.
 *
\*--------------------------------------------------------------------------*/
            if (CtrlSetMultCurSel(szDialogName, nControlID, szText,
                                  nSelectFlag) < 0) then

               MessageBox("CtrlSetMultCurSel failed.", SEVERE);
            endif;

            // Destroy the listID string list.
            ListDestroy(listID);
            MessageBox("listID destroyed.", INFORMATION);
         case DLG_CLOSE:

            bDone = TRUE;
         case RES_PBUT_NEXT:

            // Create the listFolders string list.
            listFolders = ListCreate(STRINGLIST);

            if (listFolders = LIST_NULL) then
```

```
              MessageBox("Unable to create list.", SEVERE);
           else

              MessageBox("listFolders created.", INFORMATION);
           endif;
/*--------------------------------------------------------------------------*\
 *
 * CtrlGetMultCurSel retrieves the highlighted elements in the listbox, and
 * asigns these elements into the newly created listFolders string list.
 *
\*--------------------------------------------------------------------------*/
           if (CtrlGetMultCurSel(szDialogName, nControlID,
                                 listFolders) < 0) then

              MessageBox("CtrlGetMultCurSel failed.", SEVERE);
           else

              MessageBox("CtrlGetMultCurSel successful.", INFORMATION);
           endif;

           bDone = TRUE;
        case RES_PBUT_BACK:

           bDone = TRUE;
        case RES_PBUT_CANCEL:

           bDone = TRUE;
     endswitch;
  endwhile;

  szMsg   = "The following are the elements highlighted in the listbox:";

  // Display the list of elements highlighted.
  SdShowInfoList(szTitle, szMsg, listFolders);

  // Remove listFolders string list from memory.
  ListDestroy(listFolders);
  MessageBox("listFolders destroyed.", INFORMATION);

  // Close the dialog.
  EndDialog(szDialogName);

  // Remove dialog from memory.
  ReleaseDialog(szDialogName);

endprogram

  #define SD_SINGLE_DIALOGS 1
  #define SD_SHOWINFOLIST   1

  #include "Sddialog.rul"

// Source file: Is5fn042.rul
```

CtrlSetState

Syntax

CtrlSetState (szDialogName, nControlID, nState);

Description

The CtrlSetState function sets the current state of a check box or radio button control in a custom dialog box. You can set certain characteristics of radio buttons and check boxes when you create them using a resource or dialog editor. If you experience difficulties with the behavior of a button control, check the characteristics of the control in the editor.

Parameters

szDialogName
The name of a valid dialog box that contains the check box or radio button control.

nControlID
The resource ID of the check box or radio button control.

nState
Specify the new state of the button control. The following constants are available:

BUTTON_CHECKED
Sets the button's state to CHECKED.

BUTTON_UNCHECKED
Sets the button's state to UNCHECKED.

Return values

0
CtrlSetState successfully set the state of the check box or radio button control.

< 0
CtrlSetState was unable to set the state of the control.

{button ,JI(`LANGREF.HLP>Examples',`CtrlSetState_example')} Example

{button ,AL(`CtrlGetState;DefineDialog;EndDialog;ReleaseDialog;WaitOnDialog',0,`',`')} See also

CtrlSetState example

```
/*--------------------------------------------------------------------------*\
 *
 * This example illustrates the usage of the CtrlGetState and CtrlSetState
 * functions.
 *
 * CtrlSetState is called to initialize the radio and check buttons on the
 * custom dialog box.
 *
 * After the dialog box has been closed, CtrlGetState is called in succession
 * to display messages indicating the choices the user had made.
 *
 * NOTE: The 'custom' dialog used in this script is actually a dialog used
 *       by InstallShield, located in the _isres.dll file.  This .DLL file is
 *       already compressed in _sys1.cab.
 *
\*--------------------------------------------------------------------------*/

// Set the dialog box controls.
#define  RES_DIALOG_ID       12020    // 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_PBUT_BACK          12    // ID of '< Back' push button.
#define  ID_OP1_CHECK          501    // ID of Option 1 check button.
#define  ID_OP2_CHECK          502    // ID of Option 2 check button.
#define  ID_OP3_CHECK          503    // ID of Option 3 check button.
#define  ID_OP4_CHECK          504    // ID of Option 4 check button.

   STRING  szDialogName, szDLL, szProgram, szTitle, szMsg;
   NUMBER  nResult, nCmdValue, nControlID, nState;
   BOOL    bDone;

program

   szDialogName = "ExDialog";
   szDLL        = "";

   // szDLL is set to null, EzDefineDialog searches the _ISUSER.DLL and
   // _ISRES.DLL files.  This example 'custom' dialog is located in _ISRES.DLL.
   nResult  = EzDefineDialog(szDialogName, szDLL, "", RES_DIALOG_ID);

   if(nResult < 0) then

      MessageBox("Error in defining dialog", WARNING);
      bDone = TRUE;
   else

      bDone = FALSE;
   endif;

   while (bDone = FALSE)

      // Display the dialog box and wait for a command.
      nCmdValue = WaitOnDialog(szDialogName);
      switch (nCmdValue)

         case DLG_ERR:

            MessageBox("Dialog failed", SEVERE);
            bDone = TRUE;
         case DLG_INIT:

/*--------------------------------------------------------------------------*\
 *
 * Initialize the dialog box by setting the Option 1 and Option 2 radio
 * buttons to checked.
 *
\*--------------------------------------------------------------------------*/
            nControlID = ID_OP1_CHECK;
            nState     = BUTTON_CHECKED;
            if (CtrlSetState(szDialogName, nControlID, nState) < 0) then

               MessageBox("First call to CtrlSetState failed.", SEVERE);
            endif;

            nControlID = ID_OP2_CHECK;
            if (CtrlSetState(szDialogName, nControlID, nState) < 0) then

               MessageBox("Second call to CtrlSetState failed.", SEVERE);
            endif;
         case RES_PBUT_NEXT:

            bDone = TRUE;
         case RES_PBUT_CANCEL:

            bDone = TRUE;
         case RES_PBUT_BACK:

            bDone = TRUE;
      endswitch;
   endwhile;

   szTitle = "CtrlGetState & CtrlSetState Example";
   szMsg   = "You selected the following:";
   SprintfBox(INFORMATION, szTitle, szMsg);

/*--------------------------------------------------------------------------*\
 *
 * Call CtrlGetState to retrieve the state of a button.  If the button is
 * checked, display the operating system the selected button represents.
 *
\*--------------------------------------------------------------------------*/
   nControlID = ID_OP1_CHECK;
   if (CtrlGetState(szDialogName, nControlID) = BUTTON_CHECKED) then

      MessageBox("Option 1", INFORMATION);
   endif;

   nControlID = ID_OP2_CHECK;
   if (CtrlGetState(szDialogName, nControlID) = BUTTON_CHECKED) then

      MessageBox("Option 2", INFORMATION);
   endif;

   nControlID = ID_OP3_CHECK;
   if (CtrlGetState(szDialogName, nControlID) = BUTTON_CHECKED) then

      MessageBox("Option 3", INFORMATION);
   endif;

   nControlID = ID_OP4_CHECK;
   if (CtrlGetState(szDialogName, nControlID) = BUTTON_CHECKED) then
```

```
      MessageBox("Option 4", INFORMATION);
   endif;

   EndDialog(szDialogName);
   ReleaseDialog(szDialogName);

endprogram

// Source file: Is5fn043.rul
```

CtrlSetText

Syntax

CtrlSetText (szDialogName, nControlID, szText);

Description

The CtrlSetText function sets the text of a single-line edit field, static text field, or button control in a custom dialog box. To set the text in multi-line edit fields, call CtrlSetMLEText.

Parameters

szDialogName
The name of a valid dialog box.

nControlID
The resource ID of the single line edit field, static text field, or button control in a dialog box you are setting the text for.

szText
The text you want to place in the control.

Return values

0
CtrlSetText successfully set the text in the control.

< 0
CtrlSetText was unable to set the text in the control.

{button ,JI(`LANGREF.HLP>Examples',`CtrlSetText_example')} Example

{button ,AL(`CtrlClear;CtrlGetText;DefineDialog;EndDialog;ReleaseDialog;WaitOnDialog',0,`',`')} See also

CtrlSetText example

```
/*--------------------------------------------------------------------------*\
 *
 * This example illustrates the usage of the CtrlSetText and CtrlGetText
 * functions.
 *
 * When the dialog box is initialized, CtrlSetText is used to set text in the
 * dialog box.  When the Next button is pressed, CtrlGetText is called to
 * retrieve the name and company strings from the dialog box.  These strings
 * are then displayed in message boxes.
 *
 * NOTE: The 'custom' dialog used in this script is actually a dialog used
 *       by InstallShield, located in the _isres.dll file.  This .DLL file is
 *       already compressed in _sys1.cab.
 *
\*--------------------------------------------------------------------------*/

#define NAME    "Your Name"
#define COMPANY "Your Company"

   #define RES_DIALOG_ID       12001 // ID of Dialog Itself
   #define RES_PBUT_NEXT           1 // ID of 'Next >' push button
   #define RES_PBUT_BACK          12 // ID of '< Back' push button
   #define RES_PBUT_CANCEL         9 // ID of 'Cancel' push button
   #define RES_EDITBOX_NAME      301 // ID of the User Name edit box
   #define RES_EDITBOX_COMPANY   302 // ID of the Company Name edit box.

   STRING szDialogName, szDLL, szText, svText;
   NUMBER nResult, nCmdValue, nControlID;
   BOOL   bDone;

program

   // Define a dialog box.
   szDialogName = "ExDialog";
   szDLL = "";

   nResult  = EzDefineDialog(szDialogName, szDLL, "", RES_DIALOG_ID);

   if(nResult < 0) then

      MessageBox("Error in defining dialog", SEVERE);
      bDone = TRUE;
   else

      bDone = FALSE;
   endif;

Register:

   while (bDone = FALSE)

      // Display dialog and wait for user's input
      nCmdValue = WaitOnDialog(szDialogName);

      switch (nCmdValue)

         case DLG_ERR:
            MessageBox("Dialog Failed", SEVERE);
            bDone = TRUE;
         case DLG_INIT:

/*--------------------------------------------------------------------------*\
 *
 * CtrlSetText sets the text of the edit box to the NAME and COMPANY
 * preprocessor string constants.
 *
\*--------------------------------------------------------------------------*/
            nControlID = RES_EDITBOX_NAME;
            szText     = NAME;
            if (CtrlSetText(szDialogName, nControlID, szText) < 0) then

               MessageBox("First call to CtrlSetText failed.", SEVERE);
               bDone = TRUE;
            endif;

            nControlID = RES_EDITBOX_COMPANY;
            szText     = COMPANY;
            if (CtrlSetText(szDialogName, nControlID, szText) < 0) then

               MessageBox("Second call to CtrlSetText failed.", SEVERE);
               bDone = TRUE;
            endif;
         case RES_PBUT_BACK:

            bDone = TRUE;
         case RES_PBUT_CANCEL:

            bDone = TRUE;
         case RES_PBUT_NEXT:

            bDone = TRUE;
      endswitch;
   endwhile;

/*--------------------------------------------------------------------------*\
 *
 * CtrlGetText is called to retrieve the values in the edit boxes.
 *
\*--------------------------------------------------------------------------*/
   nControlID = RES_EDITBOX_NAME;
   if (CtrlGetText(szDialogName, nControlID, svText) < 0) then

      MessageBox("First call to CtrlGetText failed.", SEVERE);
      bDone = TRUE;
   endif;

   // Display text retrieved.
   MessageBox(svText, INFORMATION);

   nControlID = RES_EDITBOX_COMPANY;
   if (CtrlGetText(szDialogName, nControlID, svText) < 0) then

      MessageBox("Second call to CtrlGetText failed.", SEVERE);
      bDone = TRUE;
   endif;

   MessageBox(svText, INFORMATION);

   // Close the dialog box
   EndDialog(szDialogName);
   // Free the dialog box from memory
   ReleaseDialog(szDialogName);

endprogram

// Source file: Is5fn045.rul
```

DefineDialog

Syntax

DefineDialog (szDialogName, hInstance, szDLLName, nDialog, szDialog, nReserved, hwndOwner, lStyle);

Description

The DefineDialog function defines a custom dialog box. Call this function instead of EzDefineDialog when you need to specify a dialog attribute that cannot be specified with EzDefineDialog. Note that DefineDialog does *not* display the custom dialog box. To display a custom dialog box, you must call WaitOnDialog.

Parameters

szDialogName
The name of the custom dialog you want to define. This name identifies this dialog and is used in all subsequent calls to custom dialog box functions. The dialog's name is case-sensitive—you must use it exactly as you specify in this parameter.

hInstance
The instance handle of the DLL in which the dialog box resides. If you specify the fully-qualified name of the DLL in szDLLName, you can enter 0 in this parameter. To obtain the instance handle of a DLL, call the Microsoft Windows API LoadLibrary.

szDLLName
The fully-qualified name of the DLL file that contains the dialog resource. If you do not specify a path, InstallShield searches for the DLL in the Windows folder. If it is not found there, InstallShield searches the folders specified in the search path. If you specified the instance handle of the DLL in the parameter hInstance, you can enter a null string ("") here. When the dialog box is located in _isuser.dll you can specify a null string ("") in this parameter. InstallShield will automatically check _isuser.dll if this parameter is specified as a null string ("").

nDialog
The dialog's resource ID if you use a number (rather than a string) to identify the resource. This parameter is used only when szDialog is a null string (""). Using this parameter rather than szDialog to identify the dialog resource is highly recommended.

szDialog
The dialog's resource ID if you use a string (rather than a number) to identify the resource. If this parameter is a null string (""), InstallShield uses nDialog to identify the dialog resources. Using nDialog rather than szDialog to identify the dialog resource is highly recommended.

nReserved
Enter 0 in this parameter.

hwndOwner
The window handle of the owner window. Specify HWND_INSTALL in this parameter to set the owner of the dialog box to the InstallShield main installation window.

lStyle
Specify the constant DLG_MSG_STANDARD ORed with the constant DLG_CENTERED. No other values are allowed. This parameter is provided for compatibility with previous versions of InstallShield.

Return values

0
DefineDialog successfully defined the dialog.

DLG_ERR_ALREADY_EXISTS

You are trying to define a dialog already defined by DefineDialog. You cannot define two dialogs with the same name.

DLG_ERR

Indicates another unspecified error condition.

{button ,JI(`LANGREF.HLP>Examples',`DefineDialog_example')} Example

{button ,AL(`EndDialog;ReleaseDialog;WaitOnDialog',0,`',`')} See also

DefineDialog example

```
/*--------------------------------------------------------------------------*\
 *
 * This example illustrates the uses of the DefineDialog function.
 *
 * The script below first defines "ExDialog" using the DefineDialog
 * function, then displays the dialog box by calling the WaitOnDialog
 * function.
 *
 * NOTE: The 'custom' dialog used in this script is actually a dialog used
 *       by InstallShield, located in the _isres.dll file.  This .DLL file is
 *       already compressed in _sys1.cab.
 *
\*--------------------------------------------------------------------------*/

   // Set constants for the dialog box controls.
   #define RES_DIALOG_ID      12027 // 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_PBUT_BACK         12 // ID of '< Back' push button

   STRING  szDialogName, szDLLName, szDialog;
   NUMBER  nDialog, nResult, nCmdValue;
   BOOL    bDone;
   HWND    hInstance, hwndParent, hwndOwner;
   INT     iStyle;

program

start:

   // Define the name of a dialog box.
   szDialogName = "DefineDialogEx";
   hInstance    = 0;

   // The .DLL which contains the dialog box.
   szDLLName    = "";
   nDialog      = RES_DIALOG_ID;
   szDialog     = "";
   hwndParent   = 0;
   hwndOwner    = HWND_INSTALL;
   iStyle       = DLG_MSG_STANDARD|DLG_CENTERED;

/*--------------------------------------------------------------------------*\
 *
 * The following defines a dialog to be used in this script.  Note the szDLL
 * variable is set to null.  The _ISUSER.DLL and _ISRES.DLL will be searched
 * by InstallShield.  This dialog box is located in _ISRES.DLL.
 *
\*--------------------------------------------------------------------------*/
   nResult  = DefineDialog(szDialogName, hInstance, szDLLName, nDialog,
                           szDialog, hwndParent, hwndOwner, iStyle);

   // Error check DefineDialog
   if(nResult < 0) then

      MessageBox("Error in defining dialog", SEVERE);
      bDone = TRUE;
   else
```

```
      bDone = FALSE;
   endif;

   while (bDone = FALSE)

      // Display dialog box and set nCmdValue to the return command.
      nCmdValue = WaitOnDialog(szDialogName);

      switch (nCmdValue)

         case DLG_CLOSE:

            MessageBox("Dialog closed.", INFORMATION);
            bDone = TRUE;
         case DLG_ERR:

            MessageBox("Dialog failed", SEVERE);
            bDone = TRUE;
         case DLG_INIT:

         case RES_PBUT_CANCEL:

            MessageBox("Dialog canceled.", INFORMATION);
            bDone = TRUE;
         case RES_PBUT_NEXT:

            MessageBox("Next button pressed.", INFORMATION);
            bDone = TRUE;
         case RES_PBUT_BACK:

            MessageBox("Back button pressed.", INFORMATION);
            bDone = TRUE;
      endswitch;
   endwhile;

   // Close the dialog box
   EndDialog(szDialogName);

   // Free the dialog box from memory
   ReleaseDialog(szDialogName);

   if (AskYesNo("Would you like to redo this example?", YES) = YES) then

      goto start;
   endif;

endprogram

// Source file: Is5fn055.rul
```

EndDialog

Syntax

EndDialog (szDialogName);

Description

The EndDialog function closes a custom dialog box. It removes the dialog box and initiates the dialog closing process. Use EndDialog when any of the following conditions exist:

- The Next button or its equivalent has been processed.
- The Cancel button or its equivalent has been processed.
- The Close system menu option has been selected (this action sends the DLG_CLOSE message).
- Any other situation in which the user ends the dialog operation.

Parameters

szDialogName
The name of a valid dialog box you want to close.

Return values

0
EndDialog successfully closed the dialog box.

< 0
EndDialog was unable to close the dialog box.

Comments

- After calling EndDialog to end a custom dialog box, call the ReleaseDialog function to free the memory associated with the custom dialog box.
- You can call WaitOnDialog to redisplay a custom dialog box that was closed by a call to EndDialog provided that you have not called ReleaseDialog to remove the dialog box from memory. Note, however, that if you call WaitOnDialog to open a dialog box that has been opened and closed previously in your script, you must call CmdGetHwndDlg again to get the new handle. The old handle is no longer valid.

{button ,JI(`LANGREF.HLP>Examples',`EndDialog_example')} Example

{button ,AL(`DefineDialog;ReleaseDialog;WaitOnDialog',0,`',`')} See also

EndDialog example

```
/*---------------------------------------------------------------------------*\
 *
 * This example illustrates the uses of the EndDialog and ReleaseDialog
 * functions.
 *
 * WaitOnDialog is called at first to load a custom dialog box into memory,
 * and to display it on the screen.  Once the dialog has been closed or
 * cancelled, EndDialog is called.  This will close the custom dialog box.
 * Right after EndDialog has executed, ReleaseDialog will be called to remove
 * the box completely from memory.
 *
 * NOTE: The 'custom' dialog used in this script is actually a dialog used
 *       by InstallShield, located in the _isres.dll file.  This .DLL file is
 *       already compressed in _sys1.cab.
 *
\*---------------------------------------------------------------------------*/

   // Define the dialog box controls.
   #define RES_DIALOG_ID      12027 // 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_PBUT_BACK         12 // ID of '< Back' push button

   STRING szDialogName, szDLL;
   NUMBER nResult, nCmdValue;
   BOOL   bDone;

program

   // Define a dialog box.
   szDialogName = "EndDialogEx";
   szDLL    = "";

   // szDLL is set to null, the _ISUSER.DLL and _ISRES.DLL files will be
   // searched.  This dialog is located in _ISRES.DLL.
   nResult  = EzDefineDialog(szDialogName, szDLL, "", RES_DIALOG_ID);

   if(nResult < 0) then

      MessageBox("Error in defining dialog", WARNING);
     abort;
   else

      bDone = FALSE;
   endif;

   while (bDone = FALSE)

      // Display dialog and wait for user's input
      nCmdValue = WaitOnDialog(szDialogName);

      switch (nCmdValue)
         case DLG_CLOSE:
            bDone = TRUE;

         case DLG_ERR:

            MessageBox("Dialog Failed", SEVERE);
            bDone = TRUE;
         case DLG_INIT:

         case RES_PBUT_CANCEL:

            bDone = TRUE;
         case RES_PBUT_NEXT:

            bDone = TRUE;
         case RES_PBUT_BACK:

            bDone = TRUE;
      endswitch;
   endwhile;

/*--------------------------------------------------------------------------*\
 *
 * EndDialog is called to close the custom dialog box.
 *
\*--------------------------------------------------------------------------*/
   if (EndDialog(szDialogName) < 0) then

      MessageBox("EndDialog failed.", SEVERE);
   else

      MessageBox("EndDialog removed the dialog box from display.",
                 INFORMATION);
   endif;

/*--------------------------------------------------------------------------*\
 *
 * ReleaseDialog is called to remove the custom dialog box from memory.
 *
\*--------------------------------------------------------------------------*/
   if (ReleaseDialog(szDialogName) < 0) then

      MessageBox("ReleaseDialog failed.", SEVERE);
   else

      MessageBox("ReleaseDialog released the custom dialog box from memory.",
                 INFORMATION);
   endif;

endprogram

// Source file: Is5fn059.rul
```

EzDefineDialog

Syntax

EzDefineDialog (szDialogName, szDLL, szID, nID);

Description

Call the EzDefineDialog function to register a custom dialog with InstallShield. EzDefineDialog centers the dialog box on the target screen. If you need to position the dialog box in a different location, register or define the dialog box to InstallShield by calling DefineDialog.

Parameters

szDialogName
A unique name for the dialog. InstallShield uses this name to identify and register the dialog in the script.

szDLL
The fully-qualified name of the DLL that contains the dialog box you are registering. If you enter a null string (""), InstallShield will look in _isuser.dll and _isres.dll for the dialog.

szID
The name (identifier) of the dialog box. Enter the string identifier only. If the dialog box has a numeric identifier, enter a null string ("") in this parameter.

nID
The numeric identifier of the dialog box. Only enter the numeric identifier. If the dialog box has a string identifier, enter 0 in this parameter.

Return values

0
EzDefineDialog successfully registered the dialog box.

DLG_ERR_ALREADY_EXISTS
Indicates that you are trying to define a dialog already defined in the setup script. You cannot define two dialogs with the same name.

DLG_ERR
Indicates an unspecified error condition.

{button ,JI(`LANGREF.HLP>Examples',`EzDefineDialog_example')} Example

{button ,AL(`DefineDialog;EndDialog;ReleaseDialog;WaitOnDialog',0,`',`')} See also

EzDefineDialog example

```
/*--------------------------------------------------------------------------*\
 *
 * This example illustrates the uses of the EzDefineDialog and WaitOnDialog.
 *
 * The script below first defines "ExDialog" using the EzDefineDialog
 * function, then displays the dialog box by calling the WaitOnDialog
 * function.  If the dialog box is displayed correctly, the listbox in the
 * dialog box will contain all of the files in the TARGET_DIR directory.  If
 * the OK button is pressed, all of the subdirectories in the TARGET_DIR
 * directory will then be displayed within the listbox.
 *
 * NOTE: The 'custom' dialog used in this script is actually a dialog used
 *       by InstallShield, located in the _isres.dll file.  This .DLL file is
 *       already compressed in _sys1.cab.
 *
\*--------------------------------------------------------------------------*/

   // Set constants for the dialog box controls.
   #define RES_DIALOG_ID      12027 // 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_PBUT_BACK         12 // ID of '< Back' push button

   STRING  szDialogName, szDLL, szID;
   NUMBER  nResult, nCmdValue, nID;
   BOOL    bDone;

program

   // Define a dialog box.
   szDialogName = "ExDialog";
   szDLL        = "";
   szID         = "";
   nID          = RES_DIALOG_ID;
/*--------------------------------------------------------------------------*\
 *
 * The following defines a dialog to be used in this script.  Note the szDLL
 * variable is set to null.  The _ISUSER.DLL and _ISRES.DLL files will be
 * searched by InstallShield.  This dialog is located in _ISRES.DLL.
 *
\*--------------------------------------------------------------------------*/
   nResult  = EzDefineDialog(szDialogName, szDLL, szID, nID);

   // Error check EzDefineDialog
   if(nResult < 0) then

      MessageBox("Error in defining dialog", SEVERE);
      abort;
   else

      MessageBox("EzDefineDialog successfully defined custom dialog",
         INFORMATION);
      bDone = FALSE;
   endif;

   while (bDone = FALSE)

/*--------------------------------------------------------------------------*\
 *
 * WaitOnDialog displays the dialog and returns user's input from within a
 * while loop.  The while loop is exited when bDone is set to TRUE.
 *
\*-----------------------------------------------------------------------------*/
      nCmdValue = WaitOnDialog(szDialogName);

      switch (nCmdValue)

         // WaitOnDialog returned the dialog box was closed.
         case DLG_CLOSE:

            MessageBox("Dialog closed.", INFORMATION);

            // End while loop.
            bDone = TRUE;

         // WaitOnDialog returned an error occurred.
         case DLG_ERR:

            MessageBox("Dialog failed", SEVERE);
            bDone = TRUE;

         // WaitOnDialog returned the dialog box is to be displayed.
         case DLG_INIT:
         Delay(1);

         // WaitOnDialog returned the cancel pushbutton was pressed.
         case RES_PBUT_CANCEL:

            MessageBox("Dialog canceled.", INFORMATION);
            bDone = TRUE;

         // WaitOnDialog returned the Next pushbutton was pressed.
         case RES_PBUT_NEXT:

            MessageBox("Next button pressed.", INFORMATION);
            bDone = TRUE;

         // WaitOnDialog returned the help button was pressed.
         case RES_PBUT_BACK:

            MessageBox("Back button pressed.", INFORMATION);
            bDone = TRUE;
      endswitch;
   endwhile;

   // Close the dialog box
   EndDialog(szDialogName);

   // Free the dialog box from memory
   ReleaseDialog(szDialogName);


endprogram

// Source file: Is5fn069.rul
```

GetFont

Syntax

GetFont (szFontName, nPointSize, nAttributes);

Description

The GetFont function builds a font and retrieve its handle. You can use the font handle to specify the font used by the controls in a custom dialog box.

Parameters

szFontName

The font face name you want to build.

nPointSize

The point size of the font you want to build.

nAttributes

Specify the style of the font you are building. The constants listed below are available. You can combine the constants using 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

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
```

HIWORD

Syntax

HIWORD (lValue);

Description

The HIWORD function retrieves the high-order word from the 32-bit integer value you specify in lValue.

Parameters

lValue
The value you want to convert.

Return values

0
HIWORD successfully retrieved the high word.

< 0
HIWORD was unable to retrieve the high word.

{button ,JI(`LANGREF.HLP>Examples',`HIWORD_example')}   Example

{button ,AL(`LOWORD',0,`',`')} See also

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
```

LOWORD

Syntax

LOWORD (lValue);

Description

The LOWORD function extracts the low-order word (two bytes) from the 32-bit integer value specified by lValue. Advanced Microsoft Windows developers will recognize this as the LOWORD macro.

Parameters

lValue

Enter the 32-bit integer from which you want to extract the lower two bytes.

Return values

This function returns the lower two bytes of the integer.

{button ,JI(`LANGREF.HLP>Examples',`LOWORD_example')} Example

{button ,AL(`HIWORD',0,`',`')}   See also

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
```

ReleaseDialog

Syntax

ReleaseDialog (szDialogName);

Description

The ReleaseDialog function frees all memory associated with the custom dialog identified in szDialogName. Call this function after calling EndDialog. Call this function outside the message handling case statement.

Parameters

szDialogName
Enter the name of the dialog you want to destroy.

Return values

0
Indicates that the function successfully freed all memory associated with the custom dialog.

DLG_ERR
Function failed. The dialog name may be invalid.

DLG_ERR_ENDDLG
ReleaseDialog was called before EndDialog. You must call EndDialog first to remove the dialog.

{button ,JI(`LANGREF.HLP>Examples',`ReleaseDialog_example')} Example

{button ,AL(`DefineDialog;EndDialog;WaitOnDialog',0,`',`')} See also

ReleaseDialog example

```
/*--------------------------------------------------------------------------*\
 *
 * This example illustrates the uses of the EndDialog and ReleaseDialog
 * functions.
 *
 * WaitOnDialog is called at first to load a custom dialog box into memory,
 * and to display it on the screen.  Once the dialog has been closed or
 * cancelled, EndDialog is called.  This will close the custom dialog box.
 * Right after EndDialog has executed, ReleaseDialog will be called to remove
 * the box completely from memory.
 *
 * NOTE: The 'custom' dialog used in this script is actually a dialog used
 *       by InstallShield, located in the _isres.dll file.  This .DLL file is
 *       already compressed in _sys1.cab.
 *
\*--------------------------------------------------------------------------*/

   // Define the dialog box controls.
   #define RES_DIALOG_ID      12027 // 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_PBUT_BACK         12 // ID of '< Back' push button

   STRING szDialogName, szDLL;
   NUMBER nResult, nCmdValue;
   BOOL   bDone;

program

   // Define a dialog box.
   szDialogName = "EndDialogEx";
   szDLL    = "";

   // szDLL is set to null, the _ISUSER.DLL and _ISRES.DLL files will be
   // searched.  This dialog is located in _ISRES.DLL.
   nResult  = EzDefineDialog(szDialogName, szDLL, "", RES_DIALOG_ID);

   if(nResult < 0) then

      MessageBox("Error in defining dialog", WARNING);
     abort;
   else

      bDone = FALSE;
   endif;

   while (bDone = FALSE)

      // Display dialog and wait for user's input
      nCmdValue = WaitOnDialog(szDialogName);

      switch (nCmdValue)
         case DLG_CLOSE:
            bDone = TRUE;

         case DLG_ERR:

            MessageBox("Dialog Failed", SEVERE);
            bDone = TRUE;
         case DLG_INIT:

         case RES_PBUT_CANCEL:

            bDone = TRUE;
         case RES_PBUT_NEXT:

            bDone = TRUE;
         case RES_PBUT_BACK:

            bDone = TRUE;
      endswitch;
   endwhile;

/*-------------------------------------------------------------------------*\
 *
 * EndDialog is called to close the custom dialog box.
 *
\*-------------------------------------------------------------------------*/
   if (EndDialog(szDialogName) < 0) then

      MessageBox("EndDialog failed.", SEVERE);
   else

      MessageBox("EndDialog removed the dialog box from display.",
                 INFORMATION);
   endif;

/*-------------------------------------------------------------------------*\
 *
 * ReleaseDialog is called to remove the custom dialog box from memory.
 *
\*-------------------------------------------------------------------------*/
   if (ReleaseDialog(szDialogName) < 0) then

      MessageBox("ReleaseDialog failed.", SEVERE);
   else

      MessageBox("ReleaseDialog released the custom dialog box from memory.",
                 INFORMATION);
   endif;

endprogram

// Source file: Is5fn059.rul
```

SdMakeName

Syntax

SdMakeName (svSection, szDlg, szUnused, nvDlgName);

Description

The SdMakeName function 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.

When using SdMakeName 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

svSection
The section name (for example, “MyDlg-0”). InstallShield places a value into this variable using the variables szDlg and nvDlgName. This value is used by SilentReadData and SilentWriteData.

szDlg
The name of the custom dialog (for example, “MyDlg”) for which you are creating a section name.

szUnused
This parameter is not used; you can enter a null string ("") or an empty string variable here.

nvDlgName
The counter that records the number of times SdMakeName is called for the dialog named in szDlg. InstallShield automatically increments this counter. Use a unique variable name for each custom dialog (see Comments, below).

Return values

None.

Comments

For sections to be properly named, you must use a unique variable name in the fourth parameter for each different custom dialog. A simple way to do this is to use the dialog name in szDlg to name the variable. For example, when szDlg is “MyDlgOne”, name the variable in the fourth parameter nvMyDlgOne, and when szDlg is “MyDlgTwo”, name the variable nvMyDlgTwo.

{button ,JI(`LANGREF.HLP>Examples',`SdMakeName_example')} Example

{button ,AL(`SilentReadData;SilentWriteData',0,`',`')} See also

SdMakeName example

```
/*---------------------------------------------------------------------------*\
 *
 * The following example shows how to use SdMakeName, SilentReadData and
 * SilentWriteData to handle a custom dialog box in a silent installation.
 * The resource DLL for the example custom dialog box shown below should be
 * stored in a compressed form under the SetupFiles tab of the IDE.
 *
 * The example dialog was built from the custom dialog template provided
 * with InstallShield5.
 *
 * Dialog control IDs and other information are included in the RESOURCE.H
 * file (not shown), which is included in the first line of the example must
 * be inserted in the Scripts tab of the IDE.
 * The example creates a text file called COMINIT.TXT.  If the installation
 * runs in silent mode, SilentReadData is called and the custom dialog box
 * control selections are read from the .ISS file.  The selections are then
 * saved in the COMINIT.TXT file as a means of demonstrating that they were
 * successfully read from the .ISS file.  If the installation runs in normal
 * mode, the custom dialog box is displayed, and the selections are recorded
 * in the .ISS file and indicated in message boxes.  The initial .ISS file
 * text is shown after the example script.
 *
\*---------------------------------------------------------------------------*/

   #include "Resource.h"

   #include "SDDIALOG.H"

   BOOL bDone;

   STRING svSection, svComPort, svPulse, svTone, svDial9, svVal;
   NUMBER nvCommDialog, nCmdValue, nPulseState, nToneState;
   NUMBER nDial9State, nResult, nvHandle;
   LIST listID;

program

   // Open the text file COMINIT.TXT so custom dialog box selections
   // from the .ISS file can be stored in it.
   OpenFileMode(FILE_MODE_APPEND);
   OpenFile(nvHandle, "c:\\rul", "cominit.txt");

   // If operating in silent mode, then read from the .ISS file.

   if (MODE=SILENTMODE) then

      SdMakeName(svSection, "COMM_DIALOG", "", nvCommDialog);
      SilentReadData(svSection, "Result", DATA_NUMBER, svVal, nResult);

         if(nResult = 1) then

            SRCDIR = "c:\\rul";
            // Read the data from the .ISS file. For purposes of
            // writing the results to a text file, read the
            // data as strings.
            SilentReadData(svSection, "nPulseState", DATA_STRING,
                           svPulse, nResult);
            SilentReadData(svSection, "nToneState", DATA_STRING,
                           svTone, nResult);

            SilentReadData(svSection, "nDial9State", DATA_STRING,
                           svDial9, nResult);

            // Store the custom dialog box selections in
            // the text file COMINIT.TXT.
            svVal = "Pulse box is: " ^ svPulse;
            WriteLine(nvHandle, svVal);

            svVal = "Tone  box is: " ^ svTone;
            WriteLine(nvHandle, svVal);


            svVal = "Dial9 box is: " ^ svDial9;
            WriteLine(nvHandle, svVal);

         endif;

   // If not in silent mode, then call and handle the custom dialog box
   // as you normally would.
   else

      listID = ListCreate(STRINGLIST);
      ListAddString(listID,  "COMM1:", AFTER);
      ListAddString(listID,  "COMM2:", AFTER);
      ListAddString(listID,  "COMM3:", AFTER);
      ListAddString(listID,  "COMM4:", AFTER);

      EzDefineDialog("MYCOMDIALOG", SUPPORTDIR^"RESOURCE.DLL",

                     "COMM_DIALOG",0);

      bDone = FALSE;

      while(bDone=FALSE)

         nCmdValue = WaitOnDialog("MYCOMDIALOG");

         switch(nCmdValue)
            case DLG_INIT:

               CtrlSetState("MYCOMDIALOG", ID_TONE, BUTTON_CHECKED);
               CtrlSetList("MYCOMDIALOG", ID_COMPORT, listID);
               CtrlSetState("MYCOMDIALOG", ID_DIAL9, BUTTON_CHECKED);
            case OK:

               CtrlGetCurSel("MYCOMDIALOG", ID_COMPORT, svComPort);

               nPulseState = CtrlGetState("MYCOMDIALOG", ID_PULSE);
               nToneState = CtrlGetState("MYCOMDIALOG", ID_TONE);
               nDial9State = CtrlGetState("MYCOMDIALOG", ID_DIAL9);
          bdone = TRUE;

            case RES_PBUT_CANCEL:

              bDone = TRUE;
            case DLG_CLOSE:

               bDone = TRUE;
            case ID_PULSE:

               nPulseState = CtrlGetState("MYCOMDIALOG", ID_PULSE);
               if (nPulseState = BUTTON_CHECKED) then


                  CtrlSetState("MYCOMDIALOG", ID_TONE, BUTTON_UNCHECKED);
                  CtrlSetState("MYCOMDIALOG", ID_PULSE, BUTTON_CHECKED);
               else

                  CtrlSetState("MYCOMDIALOG", ID_TONE, BUTTON_CHECKED);
                  CtrlSetState("MYCOMDIALOG", ID_PULSE, BUTTON_UNCHECKED);
               endif;
            case ID_TONE:

               nToneState = CtrlGetState("MYCOMDIALOG", ID_TONE);

               if (nPulseState = BUTTON_CHECKED) then


                  CtrlSetState("MYCOMDIALOG", ID_TONE, BUTTON_CHECKED);

                  CtrlSetState("MYCOMDIALOG", ID_PULSE, BUTTON_UNCHECKED);
               else

                  CtrlSetState("MYCOMDIALOG", ID_TONE, BUTTON_UNCHECKED);

                  CtrlSetState("MYCOMDIALOG", ID_PULSE, BUTTON_CHECKED);
               endif;

            case DLG_ERR:

               MessageBox("Internal Dialog Box Error", SEVERE);
               bDone = TRUE;
         endswitch;

      endwhile;

      EndDialog("MYCOMDIALOG");
      ReleaseDialog("MYCOMDIALOG");

      SdMakeName(svSection, "COMM_DIALOG", "", nvCommDialog);
      SilentWriteData(svSection, "nPulseState", DATA_NUMBER,
                           svPulse, nPulseState);
      SilentWriteData(svSection, "nToneState", DATA_NUMBER,
                           svTone, nToneState);
      SilentWriteData(svSection, "nDial9State", DATA_NUMBER,
                           svDial9, nDial9State);



      if (nPulseState = BUTTON_CHECKED) then

         MessageBox("The Pulse button was checked.", INFORMATION);
      else

         MessageBox("The Pulse button was unchecked.", INFORMATION);
      endif;

      if (nToneState = BUTTON_CHECKED) then
         MessageBox("The Tone button was checked.", INFORMATION);
      else
         MessageBox("The Tone button was unchecked.", INFORMATION);
```

```
      endif;

      if (nDial9State = BUTTON_CHECKED) then
         MessageBox("The Dial9 button was checked.", INFORMATION);

      else
         MessageBox("The Dial9 button was unchecked.", INFORMATION);
      endif;
   endif;

   // Close the text file COMINIT.TXT
   CloseFile(nvHandle);

endprogram

#include "SDDIALOG.RUL"

//Source File :- IS5FN168.RUL
```

The following is the initial .iss file text for the above example:

```
[InstallShield Silent]
Version=v2.10.000
File=Response File
[Application]
Name=MyDialog
Version=4.0
Company=InstallShield
[DlgOrder]
Dlg0=COMM_DIALOG-0
Count=1
[COMM_DIALOG-0]
nPulseState=CHECKED
nToneState=UNCHECKED
nDial9State=CHECKED
Result=1
```

SilentReadData

Syntax

SilentReadData (szSection, szValName, nValType, svVal, nvVal);

Description

The SilentReadData function instructs InstallShield Silent on how to read the .iss file dialog data for a custom dialog box when a setup runs in silent mode (when using the -s switch with Setup.exe).

To use SilentReadData in your script, construct the logic so that it first checks to make sure that the setup is running in silent mode. Place the SilentReadData function call inside an if-else statement, based on a test of the system variable MODE, as shown below:

```
if (MODE=SILENTMODE) then
   // Call SilentReadData here.
else
   // Make a normal, non-silent function call here.
endif;
```

Custom dialog boxes can be resources that you call and handle in your setup script using functions like EzDefineDialog and WaitOnDialog, or they can be completely external, executed as calls to functions in DLLs. In either case, you must use SilentReadData to handle the dialog's button return value (Next, Back, Cancel, etc.) and any values set or returned in variables.

Parameters

szSection

The name of the dialog data section in the .iss file. Do not include the square brackets ([ ]). The parameter szSection takes the form <functionname>-<number>, where <functionname> is the name of the dialog box function as it is used in the script, and <number> is the number of the occurrence of that dialog box in the script, beginning with 0 (zero). For example, the first occurrence of the MyDialog function dialog box would have an szSection value of "MyDialog-0", the second occurrence "MyDialog-1", the third "MyDialog-2", and so on.

szValName

The value name appearing in the dialog data section of the .iss file. Every dialog has at least one value for szValName— "Result"—which identifies the value returned by the dialog box button controls (BACK, NEXT or OK, or CANCEL). Other value names are used to identify values and data associated with the other dialog box controls.

nValType

Constant identifying the data type of the value assigned to the value name in szValName. The value itself is stored in either svVal or nvVal, depending upon the value of nValType. The following constants are available:

DATA_STRING

The value assigned to the value name in szValName is of type STRING. Its value will be stored in svVal.

DATA_NUMBER

The value assigned to the value name in szValName is of type NUMBER. Its value will be stored in nvVal.

DATA_COMPONENT

The value assigned to the value name in szValName is the name of a component It will be stored in svVal.

DATA_LIST

The value assigned to the value name in szValName is the list ID for an InstallShield list. It will be stored in nvVal.

svVal

The value assigned to the value name in szValName when nValType is DATA_STRING or

DATA_COMPONENT.

nvVal

The value assigned to the value name in szValName when nValType is DATA_NUMBER or DATA_LIST.

Return values

0

SilentReadData successfully instructed InstallShield Silent on how to read the dialog data for the custom dialog box.

< 0

SilentReadData was unable to instruct InstallShield Silent on how to read the dialog data for the custom dialog box.

Comments

SilentReadData will fail (return less than zero) under the following conditions:

- The .iss file is not found.
- The dialog sequence in the script does not match exactly the dialog sequence specified in the .iss file.
- The specified dialog data section is not found in the .iss file.
- The specified keyname is not found in the specified dialog data section.
- The data type of the value assigned to the specified keyname does not match that specified in the call to SilentReadData.

{button ,JI(`LANGREF.HLP>Examples',`SilentReadData_example')} Example

{button ,AL(`SilentWriteData;EzDefineDialog;WaitOnDialog',0,`',`')} See also

SilentReadData example

```
/*--------------------------------------------------------------------------*\
 *
 * The following example shows how to use SdMakeName, SilentReadData and
 * SilentWriteData to handle a custom dialog box in a silent installation.
 * The resource DLL for the example custom dialog box shown below should be
 * stored in a compressed form under the SetupFiles tab of the IDE.
 *
 * The example dialog was built from the custom dialog template provided
 * with InstallShield5.
 *
 * Dialog control IDs and other information are included in the RESOURCE.H
 * file (not shown), which is included in the first line of the example must
 * be inserted in the Scripts tab of the IDE.
 * The example creates a text file called COMINIT.TXT.  If the installation
 * runs in silent mode, SilentReadData is called and the custom dialog box
 * control selections are read from the .ISS file.  The selections are then
 * saved in the COMINIT.TXT file as a means of demonstrating that they were
 * successfully read from the .ISS file.  If the installation runs in normal
 * mode, the custom dialog box is displayed, and the selections are recorded
 * in the .ISS file and indicated in message boxes.  The initial .ISS file
 * text is shown after the example script.
 *
\*--------------------------------------------------------------------------*/

   #include "Resource.h"

   #include "SDDIALOG.H"

   BOOL bDone;

   STRING svSection, svComPort, svPulse, svTone, svDial9, svVal;
   NUMBER nvCommDialog, nCmdValue, nPulseState, nToneState;
   NUMBER nDial9State, nResult, nvHandle;
   LIST listID;

program

   // Open the text file COMINIT.TXT so custom dialog box selections
   // from the .ISS file can be stored in it.
   OpenFileMode(FILE_MODE_APPEND);
   OpenFile(nvHandle, "c:\\rul", "cominit.txt");

   // If operating in silent mode, then read from the .ISS file.

   if (MODE=SILENTMODE) then

      SdMakeName(svSection, "COMM_DIALOG", "", nvCommDialog);
      SilentReadData(svSection, "Result", DATA_NUMBER, svVal, nResult);

         if(nResult = 1) then

            SRCDIR = "c:\\rul";
            // Read the data from the .ISS file. For purposes of
            // writing the results to a text file, read the
            // data as strings.
            SilentReadData(svSection, "nPulseState", DATA_STRING,
                           svPulse, nResult);
            SilentReadData(svSection, "nToneState", DATA_STRING,
                           svTone, nResult);

            SilentReadData(svSection, "nDial9State", DATA_STRING,
                           svDial9, nResult);

            // Store the custom dialog box selections in
            // the text file COMINIT.TXT.
            svVal = "Pulse box is: " ^ svPulse;
            WriteLine(nvHandle, svVal);

            svVal = "Tone  box is: " ^ svTone;
            WriteLine(nvHandle, svVal);


            svVal = "Dial9 box is: " ^ svDial9;
            WriteLine(nvHandle, svVal);

         endif;

   // If not in silent mode, then call and handle the custom dialog box
   // as you normally would.
   else

      listID = ListCreate(STRINGLIST);
      ListAddString(listID,  "COMM1:", AFTER);
      ListAddString(listID,  "COMM2:", AFTER);
      ListAddString(listID,  "COMM3:", AFTER);
      ListAddString(listID,  "COMM4:", AFTER);

      EzDefineDialog("MYCOMDIALOG", SUPPORTDIR^"RESOURCE.DLL",

                     "COMM_DIALOG",0);

      bDone = FALSE;

      while(bDone=FALSE)

         nCmdValue = WaitOnDialog("MYCOMDIALOG");

         switch(nCmdValue)
            case DLG_INIT:

               CtrlSetState("MYCOMDIALOG", ID_TONE, BUTTON_CHECKED);
               CtrlSetList("MYCOMDIALOG", ID_COMPORT, listID);
               CtrlSetState("MYCOMDIALOG", ID_DIAL9, BUTTON_CHECKED);
            case OK:

               CtrlGetCurSel("MYCOMDIALOG", ID_COMPORT, svComPort);

               nPulseState = CtrlGetState("MYCOMDIALOG", ID_PULSE);
               nToneState = CtrlGetState("MYCOMDIALOG", ID_TONE);
               nDial9State = CtrlGetState("MYCOMDIALOG", ID_DIAL9);
          bdone = TRUE;

            case RES_PBUT_CANCEL:

              bDone = TRUE;
            case DLG_CLOSE:

               bDone = TRUE;
            case ID_PULSE:

               nPulseState = CtrlGetState("MYCOMDIALOG", ID_PULSE);
               if (nPulseState = BUTTON_CHECKED) then


                  CtrlSetState("MYCOMDIALOG", ID_TONE, BUTTON_UNCHECKED);
                  CtrlSetState("MYCOMDIALOG", ID_PULSE, BUTTON_CHECKED);
               else

                  CtrlSetState("MYCOMDIALOG", ID_TONE, BUTTON_CHECKED);
                  CtrlSetState("MYCOMDIALOG", ID_PULSE, BUTTON_UNCHECKED);
               endif;
            case ID_TONE:

               nToneState = CtrlGetState("MYCOMDIALOG", ID_TONE);

               if (nPulseState = BUTTON_CHECKED) then


                  CtrlSetState("MYCOMDIALOG", ID_TONE, BUTTON_CHECKED);

                  CtrlSetState("MYCOMDIALOG", ID_PULSE, BUTTON_UNCHECKED);
               else

                  CtrlSetState("MYCOMDIALOG", ID_TONE, BUTTON_UNCHECKED);

                  CtrlSetState("MYCOMDIALOG", ID_PULSE, BUTTON_CHECKED);
               endif;

            case DLG_ERR:

               MessageBox("Internal Dialog Box Error", SEVERE);
               bDone = TRUE;
         endswitch;

      endwhile;

      EndDialog("MYCOMDIALOG");
      ReleaseDialog("MYCOMDIALOG");

      SdMakeName(svSection, "COMM_DIALOG", "", nvCommDialog);
      SilentWriteData(svSection, "nPulseState", DATA_NUMBER,
                           svPulse, nPulseState);
      SilentWriteData(svSection, "nToneState", DATA_NUMBER,
                           svTone, nToneState);
      SilentWriteData(svSection, "nDial9State", DATA_NUMBER,
                           svDial9, nDial9State);



      if (nPulseState = BUTTON_CHECKED) then

         MessageBox("The Pulse button was checked.", INFORMATION);
      else

         MessageBox("The Pulse button was unchecked.", INFORMATION);
      endif;

      if (nToneState = BUTTON_CHECKED) then
         MessageBox("The Tone button was checked.", INFORMATION);
      else
         MessageBox("The Tone button was unchecked.", INFORMATION);
```

```
      endif;

      if (nDial9State = BUTTON_CHECKED) then
         MessageBox("The Dial9 button was checked.", INFORMATION);

      else
         MessageBox("The Dial9 button was unchecked.", INFORMATION);
      endif;
   endif;

   // Close the text file COMINIT.TXT
   CloseFile(nvHandle);

endprogram

#include "SDDIALOG.RUL"

//Source File :- IS5FN168.RUL
```

The following is the initial .iss file text for the above example:

```
[InstallShield Silent]
Version=v2.10.000
File=Response File
[Application]
Name=MyDialog
Version=4.0
Company=InstallShield
[DlgOrder]
Dlg0=COMM_DIALOG-0
Count=1
[COMM_DIALOG-0]
nPulseState=CHECKED
nToneState=UNCHECKED
nDial9State=CHECKED
Result=1
```

SilentWriteData

Syntax

SilentWriteData (szSection, szValName, nValType, szVal, nVal);

Description

The SilentWriteData function records selections made in custom dialogs during the setup. This selection data is written to an .iss file for use by InstallShield Silent. To write to an .iss file during a setup, run the setup using the -r switch with Setup.exe.

Custom dialog boxes can be resources that you call and handle in your setup script using functions like EzDefineDialog and WaitOnDialog, or they can be completely external, executed as calls to functions in .dlls. In either case, you must use SilentWriteData to record the dialog's button return value (Next, Back, Cancel, etc.) and any values set or returned in variables.

Parameters

szSection

The name of the dialog data section in the .iss file. Do not include the square brackets ([ ]). The parameter szSection takes the form <functionname>-<number>, where <functionname> is the name of the dialog box function as it is used in the script, and <number> is the number of the occurrence of that dialog box in the script, beginning with 0 (zero). For example, the first occurrence of the MyDialog function dialog box would have an szSection value of "MyDialog-0", the second occurrence "MyDialog-1", the third "MyDialog-2", and so on.

szValName

The value name appearing in the dialog data section of the .iss file. Every dialog has at least one value for szValName— "Result"—which identifies the value returned by the dialog box button controls (BACK, NEXT or OK, or CANCEL). Other value names are used to identify values and data associated with the other dialog box controls.

nValType

Constant identifying the data type of the value assigned to the value name in szValName. The value itself is stored in either szVal or nVal, depending upon the value of nValType. The following constants are available:

DATA_STRING

The value assigned to the value name in szValName is of type STRING. Its value will be stored in szVal.

DATA_NUMBER

The value assigned to the value name in szValName is of type NUMBER. Its value will be stored in nVal.

DATA_COMPONENT

The value assigned to the value name in szValName is the name of a component. It will be stored in szVal.

DATA_LIST

The value assigned to the value name in szValName is the list ID for an InstallShield list. It will be stored in nVal.

szVal

The value assigned to the value name in szValName when nValType is DATA_STRING or DATA_COMPONENT.

nVal

The value assigned to the value name in szValName when nValType is DATA_NUMBER or DATA_LIST.

Return values

0

SilentReadData successfully wrote the dialog data for the custom dialog box to Setup.iss.

< 0

SilentReadData was unable to write the dialog data for the custom dialog box to Setup.iss.

{button ,JI(`LANGREF.HLP>Examples',`SilentWriteData_example')} Example

{button ,AL(`SilentReadData;EzDefineDialog;WaitOnDialog',0,`',`')} See also

SilentWriteData example

```
/*--------------------------------------------------------------------------*\
 *
 * The following example shows how to use SdMakeName, SilentReadData and
 * SilentWriteData to handle a custom dialog box in a silent installation.
 * The resource DLL for the example custom dialog box shown below should be
 * stored in a compressed form under the SetupFiles tab of the IDE.
 *
 * The example dialog was built from the custom dialog template provided
 * with InstallShield5.
 *
 * Dialog control IDs and other information are included in the RESOURCE.H
 * file (not shown), which is included in the first line of the example must
 * be inserted in the Scripts tab of the IDE.
 * The example creates a text file called COMINIT.TXT.  If the installation
 * runs in silent mode, SilentReadData is called and the custom dialog box
 * control selections are read from the .ISS file.  The selections are then
 * saved in the COMINIT.TXT file as a means of demonstrating that they were
 * successfully read from the .ISS file.  If the installation runs in normal
 * mode, the custom dialog box is displayed, and the selections are recorded
 * in the .ISS file and indicated in message boxes.  The initial .ISS file
 * text is shown after the example script.
 *
\*--------------------------------------------------------------------------*/

   #include "Resource.h"

   #include "SDDIALOG.H"

   BOOL bDone;

   STRING svSection, svComPort, svPulse, svTone, svDial9, svVal;
   NUMBER nvCommDialog, nCmdValue, nPulseState, nToneState;
   NUMBER nDial9State, nResult, nvHandle;
   LIST listID;

program

   // Open the text file COMINIT.TXT so custom dialog box selections
   // from the .ISS file can be stored in it.
   OpenFileMode(FILE_MODE_APPEND);
   OpenFile(nvHandle, "c:\\rul", "cominit.txt");

   // If operating in silent mode, then read from the .ISS file.

   if (MODE=SILENTMODE) then

      SdMakeName(svSection, "COMM_DIALOG", "", nvCommDialog);
      SilentReadData(svSection, "Result", DATA_NUMBER, svVal, nResult);

         if(nResult = 1) then

            SRCDIR = "c:\\rul";
            // Read the data from the .ISS file. For purposes of
            // writing the results to a text file, read the
            // data as strings.
            SilentReadData(svSection, "nPulseState", DATA_STRING,
                           svPulse, nResult);
            SilentReadData(svSection, "nToneState", DATA_STRING,
                           svTone, nResult);

            SilentReadData(svSection, "nDial9State", DATA_STRING,
                           svDial9, nResult);

            // Store the custom dialog box selections in
            // the text file COMINIT.TXT.
            svVal = "Pulse box is: " ^ svPulse;
            WriteLine(nvHandle, svVal);

            svVal = "Tone  box is: " ^ svTone;
            WriteLine(nvHandle, svVal);


            svVal = "Dial9 box is: " ^ svDial9;
            WriteLine(nvHandle, svVal);

         endif;

   // If not in silent mode, then call and handle the custom dialog box
   // as you normally would.
   else

      listID = ListCreate(STRINGLIST);
      ListAddString(listID,  "COMM1:", AFTER);
      ListAddString(listID,  "COMM2:", AFTER);
      ListAddString(listID,  "COMM3:", AFTER);
      ListAddString(listID,  "COMM4:", AFTER);

      EzDefineDialog("MYCOMDIALOG", SUPPORTDIR^"RESOURCE.DLL",

                     "COMM_DIALOG",0);

      bDone = FALSE;

      while(bDone=FALSE)

         nCmdValue = WaitOnDialog("MYCOMDIALOG");

         switch(nCmdValue)
            case DLG_INIT:

               CtrlSetState("MYCOMDIALOG", ID_TONE, BUTTON_CHECKED);
               CtrlSetList("MYCOMDIALOG", ID_COMPORT, listID);
               CtrlSetState("MYCOMDIALOG", ID_DIAL9, BUTTON_CHECKED);
            case OK:

               CtrlGetCurSel("MYCOMDIALOG", ID_COMPORT, svComPort);

               nPulseState = CtrlGetState("MYCOMDIALOG", ID_PULSE);
               nToneState = CtrlGetState("MYCOMDIALOG", ID_TONE);
               nDial9State = CtrlGetState("MYCOMDIALOG", ID_DIAL9);
          bdone = TRUE;

            case RES_PBUT_CANCEL:

              bDone = TRUE;
            case DLG_CLOSE:

               bDone = TRUE;
            case ID_PULSE:

               nPulseState = CtrlGetState("MYCOMDIALOG", ID_PULSE);
               if (nPulseState = BUTTON_CHECKED) then


                  CtrlSetState("MYCOMDIALOG", ID_TONE, BUTTON_UNCHECKED);
                  CtrlSetState("MYCOMDIALOG", ID_PULSE, BUTTON_CHECKED);
               else

                  CtrlSetState("MYCOMDIALOG", ID_TONE, BUTTON_CHECKED);
                  CtrlSetState("MYCOMDIALOG", ID_PULSE, BUTTON_UNCHECKED);
               endif;
            case ID_TONE:

               nToneState = CtrlGetState("MYCOMDIALOG", ID_TONE);

               if (nPulseState = BUTTON_CHECKED) then


                  CtrlSetState("MYCOMDIALOG", ID_TONE, BUTTON_CHECKED);

                  CtrlSetState("MYCOMDIALOG", ID_PULSE, BUTTON_UNCHECKED);
               else

                  CtrlSetState("MYCOMDIALOG", ID_TONE, BUTTON_UNCHECKED);

                  CtrlSetState("MYCOMDIALOG", ID_PULSE, BUTTON_CHECKED);
               endif;

            case DLG_ERR:

               MessageBox("Internal Dialog Box Error", SEVERE);
               bDone = TRUE;
         endswitch;

      endwhile;

      EndDialog("MYCOMDIALOG");
      ReleaseDialog("MYCOMDIALOG");

      SdMakeName(svSection, "COMM_DIALOG", "", nvCommDialog);
      SilentWriteData(svSection, "nPulseState", DATA_NUMBER,
                           svPulse, nPulseState);
      SilentWriteData(svSection, "nToneState", DATA_NUMBER,
                           svTone, nToneState);
      SilentWriteData(svSection, "nDial9State", DATA_NUMBER,
                           svDial9, nDial9State);



      if (nPulseState = BUTTON_CHECKED) then

         MessageBox("The Pulse button was checked.", INFORMATION);
      else

         MessageBox("The Pulse button was unchecked.", INFORMATION);
      endif;

      if (nToneState = BUTTON_CHECKED) then
         MessageBox("The Tone button was checked.", INFORMATION);
      else
         MessageBox("The Tone button was unchecked.", INFORMATION);
```

```
      endif;

      if (nDial9State = BUTTON_CHECKED) then
         MessageBox("The Dial9 button was checked.", INFORMATION);

      else
         MessageBox("The Dial9 button was unchecked.", INFORMATION);
      endif;
   endif;

   // Close the text file COMINIT.TXT
   CloseFile(nvHandle);

endprogram

#include "SDDIALOG.RUL"

//Source File :- IS5FN168.RUL
```

The following is the initial .iss file text for the above example:

```
[InstallShield Silent]
Version=v2.10.000
File=Response File
[Application]
Name=MyDialog
Version=4.0
Company=InstallShield
[DlgOrder]
Dlg0=COMM_DIALOG-0
Count=1
[COMM_DIALOG-0]
nPulseState=CHECKED
nToneState=UNCHECKED
nDial9State=CHECKED
Result=1
```

WaitOnDialog

Syntax

WaitOnDialog (szDlgName);

Description

The WaitOnDialog function displays a custom dialog box. You can write your script to handle different responses from the user based on the return value from this function.

Parameters

szDlgName
Enter the ID of the dialog box you want to display.

Return values

dialog control ID
The ID of the dialog control that received the WM_COMMAND message.

DLG_CLOSE
This message is received as a signal that the dialog box is about to close.

DLG_ERR
This message is received if any errors occurred.

DLG_INIT
This message is received immediately before the dialog box is displayed.

{button ,JI(`LANGREF.HLP>Examples',`WaitOnDialog_example')} Example

{button ,AL(`DefineDialog;EndDialog;ReleaseDialog;EzDefineDialog',0,`',`')} See also

WaitOnDialog example

```
/*--------------------------------------------------------------------------*\
 *
 * This example illustrates the uses of the EzDefineDialog and WaitOnDialog.
 *
 * The script below first defines "ExDialog" using the EzDefineDialog
 * function, then displays the dialog box by calling the WaitOnDialog
 * function.  If the dialog box is displayed correctly, the listbox in the
 * dialog box will contain all of the files in the TARGET_DIR directory.  If
 * the OK button is pressed, all of the subdirectories in the TARGET_DIR
 * directory will then be displayed within the listbox.
 *
 * NOTE: The 'custom' dialog used in this script is actually a dialog used
 *       by InstallShield, located in the _isres.dll file.  This .DLL file is
 *       already compressed in _sys1.cab.
 *
\*--------------------------------------------------------------------------*/

   // Set constants for the dialog box controls.
   #define RES_DIALOG_ID      12027 // 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_PBUT_BACK         12 // ID of '< Back' push button

   STRING  szDialogName, szDLL, szID;
   NUMBER  nResult, nCmdValue, nID;
   BOOL    bDone;

program

   // Define a dialog box.
   szDialogName = "ExDialog";
   szDLL        = "";
   szID         = "";
   nID          = RES_DIALOG_ID;
/*--------------------------------------------------------------------------*\
 *
 * The following defines a dialog to be used in this script.  Note the szDLL
 * variable is set to null.  The _ISUSER.DLL and _ISRES.DLL files will be
 * searched by InstallShield.  This dialog is located in _ISRES.DLL.
 *
\*--------------------------------------------------------------------------*/
   nResult  = EzDefineDialog(szDialogName, szDLL, szID, nID);

   // Error check EzDefineDialog
   if(nResult < 0) then

      MessageBox("Error in defining dialog", SEVERE);
      abort;
   else

      MessageBox("EzDefineDialog successfully defined custom dialog",
         INFORMATION);
      bDone = FALSE;
   endif;

   while (bDone = FALSE)

/*--------------------------------------------------------------------------*\
 *
 * WaitOnDialog displays the dialog and returns user's input from within a
 * while loop.  The while loop is exited when bDone is set to TRUE.
 *
\*-----------------------------------------------------------------------*/
      nCmdValue = WaitOnDialog(szDialogName);

      switch (nCmdValue)

         // WaitOnDialog returned the dialog box was closed.
         case DLG_CLOSE:

            MessageBox("Dialog closed.", INFORMATION);

            // End while loop.
            bDone = TRUE;

         // WaitOnDialog returned an error occurred.
         case DLG_ERR:

            MessageBox("Dialog failed", SEVERE);
            bDone = TRUE;

         // WaitOnDialog returned the dialog box is to be displayed.
         case DLG_INIT:
         Delay(1);

         // WaitOnDialog returned the cancel pushbutton was pressed.
         case RES_PBUT_CANCEL:

            MessageBox("Dialog canceled.", INFORMATION);
            bDone = TRUE;

         // WaitOnDialog returned the Next pushbutton was pressed.
         case RES_PBUT_NEXT:

            MessageBox("Next button pressed.", INFORMATION);
            bDone = TRUE;

         // WaitOnDialog returned the help button was pressed.
         case RES_PBUT_BACK:

            MessageBox("Back button pressed.", INFORMATION);
            bDone = TRUE;
      endswitch;
   endwhile;

   // Close the dialog box
   EndDialog(szDialogName);

   // Free the dialog box from memory
   ReleaseDialog(szDialogName);


endprogram

// Source file: Is5fn069.rul
```

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.

AppCommand

Syntax

AppCommand (nObject, nCommand);

Description

The AppCommand function controls the Program Manager. This function should not be used on systems that use the Explorer shell, such as Windows 95 or Windows NT 4.0. The AppCommand function uses the standard Windows mechanism for controlling the Program Manager. This function may not work correctly with third party shells that are not 100% Program Manager compatible.

Parameters

nObject

The name of the Program Manager. Use the constant PROGMAN for this parameter.

nCommand

Specify the command (message) you want to send to the Program Manager. These constants are available:

CMD_CLOSE

Sends a close message to the Program Manager. Your application reacts based on how it processes the CLOSE message.

CMD_MAXIMIZE

Maximizes the Program Manager.

CMD_MINIMIZE

Minimizes the Program Manager.

CMD_PUSHDOWN

Pushes the Program Manager down through the stack of windows.

CMD_RESTORE

Restores the Program Manager to its original size.

Return values

0

AppCommand successfully gained control of Program Manager.

< 0

AppCommand was unable to gain control of Program Manager.

Comments

- Call the AppCommand function to control the Program Manager when you are creating program folders and items. You can use AppCommand to minimize or push the Program Manager down, causing its windows to be invisible during program folder creation.
- Two closely related functions are SendMessage and FindWindow. Use these functions to find and send messages to any other application. They are similar to the Windows API functions.

{button ,JI(`LANGREF.HLP>Examples',`AppCommand_example')} Example

{button ,AL(`FindWindow;SendMessage',0,`',`')} See also

AppCommand example

```
/*--------------------------------------------------------------------------*\
 *
 * This example illustrates the usage of the AppCommand function.
 *
 * AppCommand is used to control the Program Manager.  It is best to launch
 * this script when the Program Manager is minimized.
 *
 * NOTE: AppCommand and therefore this script have no effect under the
 *       Explorer Shell.
 *
\*--------------------------------------------------------------------------*/

program

   // The background is disabled because it covers over the Program manager.
   Disable(BACKGROUND);

/*--------------------------------------------------------------------------*\
 *
 * The folllowing pushes the Program Manager to the top of the windows stack.
 *
\*--------------------------------------------------------------------------*/
   AppCommand(PROGMAN, CMD_RESTORE);
   Delay(3);

/*--------------------------------------------------------------------------*\
 *
 * The following pushes the Program Manager to the bottom of the stack.
 *
\*--------------------------------------------------------------------------*/
   AppCommand(PROGMAN, CMD_PUSHDOWN);
   Delay(3);

/*--------------------------------------------------------------------------*\
 *
 * The following minimizes the Program Manager.
 *
\*--------------------------------------------------------------------------*/
   AppCommand(PROGMAN, CMD_MINIMIZE);

endprogram

// Source file: Is5fn005.rul
```

CallDLLFx

Syntax

CallDLLFx (szDLL, szFunction, lvValue, 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  APIENTRY  YourFunction (HWND hwnd, LPLONG lpIValue, LPSTR lpszValue);
```

Parameters

szDLL

Enter the fully-qualified filename of the DLL that contains the function you wish to execute.

szFunction

Enter the name of the function you are calling from the DLL specified in szDLL.

lvValue

This parameter is of type LONG. It is passed by reference to the DLL function. You can define the value of this parameter.

svValue

Enter the string you want 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',0,`',`')} See also

CallDLLFx example

```
/*--------------------------------------------------------------------------*\
 *
 *   IIIIIII SSSSSS
 *      II    SS                          InstallShield (R)
 *      II    SSSSSS          (c) 1990-1995, Stirling Technologies, Inc.
 *      II        SS                     All Rights Reserved.
 *   IIIIIII SSSSSS
 *
 *
 *   This source code is intended as a suplement to the InstallShield product
 *   documentation, and is provided AS IS.
 *
 *
 *       Filename:  SETUP.RUL
 *
 *    Description:  This example shows how to call a DLL function from
 *                  an InstallShield SDK Edition Script using the CallDLLFx
 *                  function.
 *
 *                  The complete Visual C++ 5.0 based DLL Project is also
 *                  provided with this example.
 *
 *
 *       External
 *          Files: None
 *
 *       Comments: Make sure that MYDLL.DLL is
 *                 compressed in the compressed lib.
 *                 (Add it on the Setup files pane/Windows 95 & NT 4.0)
 *
\*--------------------------------------------------------------------------*/


#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 ( "InstallShield SDK Edition\nCalling a DLL Example #1", 18,
WHITE );
        szDLL = SUPPORTDIR ^ "MYDLL.DLL";

        // Initialize inputs
        nValue   = 3000;
        szString = "Test String";

   EXAMPLE_1:
        // Following example calls function 'Test' in the dll MYDLL.DLL with
        // two input parameters - one integer parameter and another string
        // parameter.  The parameters are updated and returned back to the user
        // in the same variables.

        // Show inputs to users
        NumToStr( szValue, nValue );
        szResult = "Before - nValue: " + szValue + ", szString: " + szString;
        MessageBox( szResult, INFORMATION );

        // Call to DLL function, pass by value.
        nResult = CallDLLFx( szDLL, "Test", nValue, szString );

        NumToStr( szValue, nValue );
        NumToStr( szReturn, nResult );
        szResult = "Returned: " + szReturn + "\nnValue: " + szValue + "\nszString: "
+ szString;
        MessageBox( szResult, INFORMATION );

   EXAMPLE_2:
        // The following example calls function "DrawMFCDialog" in MYDLL.DLL
        // with two parameters.  The dialog is displayed, which has look and
        // feel of 'Windows 95'.  The modified values are returned in the
        // same variables.

        SetTitle ( "InstallShield SDK Edition\nCalling a DLL Example #2", 18,
WHITE );
        nValue = 20;
        szString = "Stirling";

        // Call to DLL function, pass by value.
        nResult = CallDLLFx( szDLL, "DrawMFCDialog", nValue, szString );

        if( nResult = ID_NEXT ) then
           szReturn = "Next >";
        elseif( nResult = ID_CANCEL ) then
                szReturn = "Cancel";
            else
                szReturn = "< Back";
        endif;

        NumToStr( szValue, nValue );
        szResult = "User Pressed: '" + szReturn + "'\nnValue: " + szValue + "\
nszString: " + szString;
        MessageBox( szResult, INFORMATION );

endprogram

// Source File : IS5FN614.RUL
```

Delay

Syntax

Delay (nSeconds);

Description

The Delay function delays the execution of n setup script by a specified number of seconds. Other tasks running simultaneously with InstallShield proceed normally while InstallShield is delayed.

Parameters

nSeconds

Enter the number of seconds you want 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

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
```

LaunchApp

Syntax

LaunchApp (szCommand, szCmdLine);

Description

The LaunchApp function allows you to launch another application program from within the script. The launched application and the script run simultaneously. InstallShield has no control over the launched application and cannot determine if the launched application runs successfully. Launched applications run like any other application you may start while running InstallShield. You can launch any application from InstallShield that you can normally launch from the operating system.

Parameters

szCommand
Enter the fully-qualified name of the application you want to launch. If you specify a filename without a path, InstallShield searches for the application in the current directory, the Windows directory, the Windows\System directory, and the directories listed in the PATH environment variable.

szCmdLine
Enter the command line parameters (if there are any) you want to pass to the launched application. Otherwise, enter a null string ("").

Return values

0
LaunchApp successfully launched the application.

< 0
LaunchApp was unable to find the application or could not launch the application.

Comments

n The setup process continues after the application is launched. The application may be running even after the setup script terminates.

n You can also use the FindWindow and SendMessage functions to control or send messages to the launched application. If you want to run a DOS application in a Window, you can provide a PIF (Program Information File) with the same name as the DOS application. In the PIF file you specify a windowed mode for the application to run under.

n When running a DOS program you cannot determine the return result DOS_ERRORLEVEL. However, you can place a DOS application in a batch file and have the batch file recognize the error and then create another file that contains the returned error code. You can then read the file and determine the error code returned from the DOS application.

n LaunchApp uses the Windows API WinExec to launch the application.

{button ,JI(`LANGREF.HLP>Examples',`LaunchApp_example')}Example

{button ,AL(`LaunchAppAndWait',0,`',`')} See also

LaunchApp example

```
/*--------------------------------------------------------------------------*\
 *
 * This example illustrates the use of the LaunchApp Command.
 *
 * LaunchApp is called to execute an application.
 *
 * 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 APPLICATION "EXAMPLE\\NOTEPAD.EXE"
#define CMD_LINE    "EXAMPLE\\README.TXT"

   STRING szCommand, szCmdLine;

program

/*--------------------------------------------------------------------------*\
 *
 * The following launches the NOTEPAD.EXE application with the file README.TXT
 * as the command line.
 *
\*--------------------------------------------------------------------------*/
   szCommand = APPLICATION;
   szCmdLine = CMD_LINE;
   if (LaunchApp(szCommand, szCmdLine) < 0) then

      MessageBox("LaunchApp failed.", SEVERE);
   endif;

endprogram

// Source file: Is5fn092.rul
```

LaunchAppAndWait

Syntax

LaunchAppAndWait (szProgram, szCmdLine, lWait);

Description

The LaunchAppAndWait function launches an application, szProgram, with the command line szCmdLine. Unlike LaunchApp, this function allows the script to wait at this statement until the launched application terminates.

If you launch an application (application A) that then launches another application (application B), InstallShield cannot wait for application B to finish. InstallShield cannot monitor application B in any way. InstallShield can only wait for application A to terminate. If application A hangs, the setup script will wait forever if you specify the constant WAIT.

Parameters

szProgram

Enter the fully-qualified filename (including the extension) of the application you want to launch. If you specify a filename without the path and the file extension, InstallShield may not be able to launch the application.

szCmdLine

Enter the command line parameters (if there are any) you want to pass to the launched application. Otherwise, enter a null string (""). Do not enter a string literal in this parameter. Use a variable to specify the command line and pass the variable in this parameter.

lWait

Use this parameter to specify whether you want InstallShield to wait for the launched application to terminate before it continues with the next line in the setup script. These constants are available:

NOWAIT

Specifies that InstallShield continues processing the script after launching the application. The application runs simultaneously with the setup script.

WAIT

Specifies that InstallShield waits until the application you launched has terminated.

Return values

1

Indicates that the launched application terminated and the setup script can resume.

< 0

Indicates that the application was not launched. Either the application was not found or some other error prevented the application from being launched.

Comments

- LaunchAppAndWait launches applications and optionally waits for them to terminate before continuing with the script. The LaunchApp function launches another application and immediately resumes executing the script.

  Owing to operating system limitations, 16-bit InstallShield running on Windows NT will not wait for a 32-bit launched application to terminate.

- LaunchAppAndWait uses the WinExec function in Windows 3.1 and the CreateProcess function in Windows 95 and Windows NT. After InstallShield launches the requested application, it looks for the window handle of the launched application. If it finds the window handle, it waits in a loop until the application window disappears.

- If you are using LaunchAppAndWait with the constant WAIT and the function does not wait until the launched application terminates before the script resumes processing, check to see if the launched application does not launch or if starts other processes to perform work. The LaunchAppAndWait function monitors the window handle of the primary launched application. If it passes control to a secondary application or process and then terminates, the script appears not to wait.
- Another problem may be that the launched application does not have a window handle. Some applications do not create a window. Since the LaunchAppAndWait function monitors window handle of the launched application, the function cannot monitor an application that does not have a window. The launched application's window does not need to be visible, but it must exist.
- It is also possible that the script cannot load a DLL needed by the application. Verify that you are loading the DLL correctly in the application. InstallShield always expects the DLL to be in the current directory. If the DLL is not located in the current directory or in the search path specified by the PATH environment variable, use the ChangeDirectory function to set the current directory to the directory containing the DLL, then use the LaunchAppAndWait function to launch the application.
- If LaunchAppAndWait is waiting for the application to terminate successfully and the application does not terminate successfully (aborts due to an error), the setup script may cease.
- You can launch DOS programs with LaunchAppAndWait. The launched DOS programs will be full-screen DOS windows unless you indicate otherwise. If you want to run a DOS application in a Window, provide a PIF (Program Information File) that has the same name as the DOS application. You can specify a windowed mode in the PIF file for the application to run under. The user can then press ALT + Enter to toggle a DOS window between the two modes.

{button ,JI(`LANGREF.HLP>Examples',`LaunchAppAndWait_example')} Example

{button ,AL(`LaunchApp',0,`',`')}See also

LaunchAppAndWait example

```
/*---------------------------------------------------------------------------*\
 *
 * This example illustrates the uses of the LaunchAppAndWait function.
 *
 * The first call to LaunchAppAndWait executes PROGRAM.  Once this executable
 * has been closed, the installation will continue.  Otherwise, the
 * installation waits forever.
 *
 * The second call to LaunchAppAndWait does not wait for PROGRAM to close.
 * Instead it displays a message immediately after execution.
 *
 * 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 PROGRAM "EXAMPLE\\NOTEPAD.EXE"
#define EXAMPLE "EXAMPLE\\EXAMPLE.TXT"

   STRING szProgram, szCmdLine, szMsg;

program

   szMsg = "Setup will now display an example text.\nThe installation will " +
            "continue once " + PROGRAM + " has been closed.";
   MessageBox(szMsg, INFORMATION);

   szProgram = PROGRAM;
   szCmdLine = EXAMPLE;

/*---------------------------------------------------------------------------*\
 *
 * Launch PROGRAM and wait for it to close.
 *
\*---------------------------------------------------------------------------*/
   if (LaunchAppAndWait(szProgram, szCmdLine, WAIT) < 0) then

      MessageBox("LaunchAppAndWait failed.", SEVERE);
   endif;

   szMsg = PROGRAM + " has been closed.\nThe installation will now " +
           "continue.";
   MessageBox(szMsg, INFORMATION);

/*---------------------------------------------------------------------------*\
 *
 * Launch PROGRAM and continue with the installation.
 *
\*---------------------------------------------------------------------------*/
   if (LaunchAppAndWait(szProgram, szCmdLine, NOWAIT) < 0) then

      MessageBox("LaunchAppAndWait failed.", SEVERE);
   endif;

   szMsg = PROGRAM + " did not have to be closed for this message to be " +
           "displayed.";
   MessageBox(szMsg, INFORMATION);
```

```
endprogram

// Source file: Is5fn093.rul
```

UnUseDLL

Syntax

UnUseDLL (szDLLName);

Description

The UnUseDLL function unloads a DLL from memory. UnUseDLL decrements the lock count of the DLL by one. When the lock count equals zero, InstallShield unloads the DLL. Every call to UseDLL should have a matching call to UnUseDLL so that DLLs are not left in memory after they are no longer needed, wasting system resources. Once you unload a DLL, you cannot call the functions in that DLL.

Microsoft Windows system DLLs, such as User.exe, User32.exe, Gdi.exe, Gdi32.exe, Krnl386.exe, and Krnl286.exe, are loaded and unloaded automatically by Windows. Do not call UseDLL and UnUseDLL to load and unload these DLLs.

Parameters

szDLLName
Enter the filename of the DLL. Do not include the path in this parameter.

Return values

0
Indicates that the function successfully unlocked and possibly unloaded the DLL from memory.

< 0
Indicates that the function was unable to unlock or unload the DLL.

Comments

If the script exits or terminates before properly unloading the DLL with UnUseDLL, the DLL will be locked in memory. If you attempt to access the DLL again, your script may fail. You must remove the DLL from memory by restarting Windows.

{button ,JI(`LANGREF.HLP>Examples',`UnUseDLL_example')} Example

{button ,AL(`CallDLLFx;UseDLL',0,`',`')} See also

UnUseDLL example

```
/*--------------------------------------------------------------------------*\
 *
 * This example illustrates the uses of the UseDLL and the UnUseDLL
 * functions.
 *
 * UseDLL is called to load an example DLL file to memory, a function of this
 * DLL file is then called to demonstrate how DLL functions can be used in
 * InstallScript.
 *
 * UnUseDLL is then called to unload the example DLL file from memory.
 *
 * 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
 * MydllReturn whose format matches the prototype declaration below.
 *
\*--------------------------------------------------------------------------*/

#define DLL_FILE  "MYDLL.DLL"

   // Prototype MydllReturn in MYDLL.DLL.
   prototype  MYDLL.MydllReturn( INT, POINTER );

   STRING  szDLL, svString;
   INT     nValue;
   POINTER psvString;
   NUMBER  nResult;
   BOOL    bDone;

program

   szDLL = DLL_FILE;

/*--------------------------------------------------------------------------*\
 *
 * Load MYDLL.DLL into memory.
 *
\*--------------------------------------------------------------------------*/
   nResult = UseDLL (szDLL);

   if (nResult = 0) then
      MessageBox ("UseDLL successful \n\n.DLL file loaded.", INFORMATION);
   else
      MessageBox ("UseDLL failed.\n\nCouldn't load .DLL file.", INFORMATION);
      abort;
   endif;

   // bDone controls the following while loop.
   bDone = FALSE;

   // As long as bDone is FALSE, continue the loop.
   while (bDone != TRUE)

      // Disable the Back button on the AskText dialog.
      Disable(BACKBUTTON);

      AskText("Enter an example string.", "Example string.", svString);

      // Get a pointer to the string, since MydllReturn takes a string
      // pointer.
      psvString = &svString;

      // Get the string length to pass to MydllReturn.
      nValue = StrLength(svString);

      // Call MydllReturn. It will display a message box demonstrating that
      // it received the data and used it.
      MydllReturn(nValue, psvString);

      // Display the new value MydllReturn altered.
      SprintfBox(INFORMATION, "UseDLL", "MydllReturn() changed the string " +
                 "to: %s", svString);

      // Give the user a chance to do another example.
      if (AskYesNo("Do another example?", YES) = NO) then
         bDone = TRUE;
      endif;
   endwhile;

/*--------------------------------------------------------------------------*\
 *
 * The following removes MYDLL.DLL from memory.
 *
\*--------------------------------------------------------------------------*/

   if (UnUseDLL (szDLL) < 0) then
      MessageBox("UnUseDLL failed.\n\nDLL still in memory.", SEVERE);
   else
      MessageBox("UnUseDLL successful.\n\n.DLL file removed from memory.",
                 INFORMATION);
   endif;

endprogram

// Source file Is5fn179.rul
```

UseDLL

Syntax

UseDLL (szDLLName);

Description

The UseDLL function loads a DLL into memory. You must load the DLL into memory before you can call a function in the DLL from a setup script. If the loaded DLL accesses other DLLs, you must ensure that the other DLLs are located in the folder specified by the system variable SUPPORTDIR.

Each time you load a DLL into memory, the DLL's lock count is incremented. The lock count counts the number of applications that are using a DLL. You should call UnUseDLL to unload a DLL as soon as you are done using it. If you do not unload a DLL when you are done with it, the DLL will remain in memory when no applications need it, thereby wasting system resources. Every call to UseDLL should have a matching call to UnUseDLL in the script.

Microsoft Windows system DLLs, such as User.exe, User32.exe, Gdi.exe, Gdi32.exe, Krnl386.exe, and Krnl286.exe, are loaded and unloaded automatically by Windows. Do not use call UseDLL and UnUseDLL to load and unload these DLLs.

Parameters

szDLLName

Enter the fully-qualified name of the DLL you want to load into memory. To include the DLL in your setup, add it to the appropriate subfolder of the Language Independent folder in the Setup Files pane. InstallShield will compress it into _user1.cab.

When Setup.exe executes, it automatically decompresses and copies the contents of _user1.cab into the temporary directory specified by SUPPORTDIR. You can then append the DLL filename to SUPPORTDIR as follows in order to reference the DLL:

```
szDLLName = SUPPORTDIR^"MYDLL.DLL";
UseDLL (szDLLName);
```

If you do not place your DLL into _user1.cab (by inserting it into the appropriate folder in the Setup Files pane), you can compress it into another library file. However, if you do so you must specify the location to which you decompress the DLL in order to reference it. You must also make sure that you do not attempt to load the DLL before you transfer it to the target system.

If you give only a filename in this parameter, InstallShield looks for the DLL in the following locations, in the order listed:

- The Windows directory.
- The Windows\System directory.
- The directories listed in the PATH environment variable.

If you do not specify an extension, InstallShield assumes the file has the extension .dll or an .exe.

Return values

0

Indicates that the function successfully loaded the DLL into memory.

< 0

Indicates that the function was unable to load the DLL into memory.

Comments

- If UseDLL fails, the most likely cause is that the DLL was not found. If this happens, make sure the correct path is specified in the parameter szDLLName.

- Another common cause of failure associated with using DLLs is related to DLL dependencies: DLLs accessed by the DLL that you load. If the DLLs that your DLL accesses are not loaded or found, your DLL call may fail. If this occurs, make sure that the other DLLs are on the system and that they are accessible.
- If the script exits or terminates before properly unloading the DLL with UnUseDLL, the DLL will be locked in memory. If you attempt to access the DLL again, your script may fail. You must remove the DLL from memory by restarting Windows.

{button ,JI(`LANGREF.HLP>Examples',`UseDLL_example')} Example

{button ,AL(`CallDLLFx;UnUseDLL',0,`',`')} See also

UseDLL example

```
/*--------------------------------------------------------------------------*\
 *
 * This example illustrates the uses of the UseDLL and the UnUseDLL
 * functions.
 *
 * UseDLL is called to load an example DLL file to memory, a function of this
 * DLL file is then called to demonstrate how DLL functions can be used in
 * InstallScript.
 *
 * UnUseDLL is then called to unload the example DLL file from memory.
 *
 * 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
 * MydllReturn whose format matches the prototype declaration below.
 *
\*--------------------------------------------------------------------------*/

#define DLL_FILE  "MYDLL.DLL"

   // Prototype MydllReturn in MYDLL.DLL.
   prototype  MYDLL.MydllReturn( INT, POINTER );

   STRING  szDLL, svString;
   INT     nValue;
   POINTER psvString;
   NUMBER  nResult;
   BOOL    bDone;

program

   szDLL = DLL_FILE;

/*--------------------------------------------------------------------------*\
 *
 * Load MYDLL.DLL into memory.
 *
\*--------------------------------------------------------------------------*/
   nResult = UseDLL (szDLL);

   if (nResult = 0) then
      MessageBox ("UseDLL successful \n\n.DLL file loaded.", INFORMATION);
   else
      MessageBox ("UseDLL failed.\n\nCouldn't load .DLL file.", INFORMATION);
      abort;
   endif;

   // bDone controls the following while loop.
   bDone = FALSE;

   // As long as bDone is FALSE, continue the loop.
   while (bDone != TRUE)

      // Disable the Back button on the AskText dialog.
      Disable(BACKBUTTON);

      AskText("Enter an example string.", "Example string.", svString);

      // Get a pointer to the string, since MydllReturn takes a string
      // pointer.
      psvString = &svString;

      // Get the string length to pass to MydllReturn.
      nValue = StrLength(svString);

      // Call MydllReturn. It will display a message box demonstrating that
      // it received the data and used it.
      MydllReturn(nValue, psvString);

      // Display the new value MydllReturn altered.
      SprintfBox(INFORMATION, "UseDLL", "MydllReturn() changed the string " +
                 "to: %s", svString);

      // Give the user a chance to do another example.
      if (AskYesNo("Do another example?", YES) = NO) then
         bDone = TRUE;
      endif;
   endwhile;

/*--------------------------------------------------------------------------*\
 *
 * The following removes MYDLL.DLL from memory.
 *
\*--------------------------------------------------------------------------*/

   if (UnUseDLL (szDLL) < 0) then
      MessageBox("UnUseDLL failed.\n\nDLL still in memory.", SEVERE);
   else
      MessageBox("UnUseDLL successful.\n\n.DLL file removed from memory.",
                 INFORMATION);
   endif;

endprogram

// Source file Is5fn179.rul
```

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 desired.

Related Function

SelectDir

Allows you to select a directory. The function creates the directory if it does not exist.

ChangeDirectory

Syntax

ChangeDirectory (szPath);

Description

The ChangeDirectory function sets the current directory.

Parameters

szPath

The fully-qualified name of the directory you are setting as the new current directory. That value 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

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
```

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
Enter the file handle that specifies 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

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
```

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

Enter the name of the file you are copying from the source directory (as specified by the current value of SRCDIR). You cannot specify a path in this parameter, only the filename. You can use wild card characters in this parameter to copy groups of files. If you use a wildcard character in szSrcFile, the parameter szTargetFile is essentially ignored.

szTargetFile

Enter the name you are giving to the file you are copying to the TARGETDIR. If you specify a wild card in szSrcFile, szTargetFile is essentially ignored. Each source file that meets the wild card specification will be copied to TARGETDIR with the same name. You cannot specify a path in this parameter, only the filename. You cannot rename groups of files using 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

- 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.
- 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.
- 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).
- 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

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
```

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

Enter 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:

- The path is illegal.
- The drive or any subdirectory in the path is write-protected.
- The drive name is invalid.
- You do not have network privileges to create subdirectories.

{button ,JI(`LANGREF.HLP>Examples',`CreateDir_example')} Example

{button ,AL(`DeleteDir;GetDir',0,`',`')} See also

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
```

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
After CreateFile creates the file, it returns the file handle for the new file in this parameter.

szPath
Enter the fully-qualified path of the subdirectory where you want to create the new file.

szFileName
Enter the name of the file you are creating.

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

- 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.
- The CreateFile function's actions are not logged for uninstallation when logging is enabled. If you want a file created by CreateFile to be logged for uninstallation, transfer a starter file with the desired filename 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

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.";
      SprintfBox(INFORMATION, szTitle, szMsg, szFileName);
   endif;

   // Close the open file.
   CloseFile(nvFileHandle);

endprogram
```

```
// Source file: Is5fn200.rul
```

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

Enter the path and the name of the directory you want to delete.

nFlag

Use this parameter to specify what you want the function to delete. These constants are available:

ALLCONTENTS

Deletes the directory in szDir even if it contains files. 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

- You cannot delete the current directory.
- You cannot delete files on a network system where you lack the appropriate rights.
- DeleteDir cannot delete read-only, hidden, or system files.
- 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

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
```

DeleteFile

Syntax

DeleteFile (szFile);

Description

The DeleteFile function deletes one or more files in the target directory. The system variable TARGETDIR contains the target directory. 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
Enter the filename of the file(s) you want to delete. You must use the system variable TARGETDIR to tell DeleteFile the target directory path. Use wild card characters 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

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
```

ExistsDir

Syntax

ExistsDir (szPath);

Description

The ExistsDir function checks for the existence of a specified directory on the target system.

Parameters

szPath
Enter the fully-qualified path of the directory you are looking for 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

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
```

ExistsDisk

Syntax

ExistsDisk (szDisk);

Description

The ExistsDisk function checks for the existence of a specified disk drive on the target system.

Parameters

szDisk
Enter the letter of the disk you are checking for.

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

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
```

FileCompare

Syntax

FileCompare (szFileName1, szFileName2, nCompareFlag);

Description

The FileCompare function compares the size, dates, or versions of two files.

Parameters

szFileName1
The name of the first file you want to compare. InstallShield uses the system variable SRCDIR as the path of the file.

szFileName2
The name of the second file you want to compare. InstallShield uses the system variable TARGETDIR as the path of the file.

nCompareFlag
Specify the options you want to include in the comparison. The following constants are available:

COMPARE_SIZE
Compares the size of the two files.

COMPARE_DATE
Compares the dates of the two files.

COMPARE_VERSION
Compares the version resource of the two files.

Return values

EQUALS
The first file's date, size, or version is equal to the that of the second file.

FILE_NOT_FOUND
The function was unable to find one or both files.

GREATER_THAN
The first file is newer or larger than the second file, depending on the constant used in the parameter nCompareFlag:

COMPARE_VERSION
File one is newer than file two.

COMPARE_DATE
File one is newer than file two.

COMPARE_SIZE
File one is larger than file two.

LESS_THAN
The first file is older or smaller than the second file.

OTHER_FAILURE
An unspecified error occurred.

{button ,JI(`LANGREF.HLP>Examples',`FileCompare_example')} Example {button ,AL(`FindAllFiles;FindFile',0,`',`')}	See also

FileCompare example

```
/*--------------------------------------------------------------------------*\
 *
 * This example illustrates the usage of the FileCompare function.
 *
 * The first call to FileCompare compares FILE_COMP1 in SDIR and FILE_COMP2 in
 * TDIR by version.
 *
 * The second call checks if FILE_COMP1 was created earlier than FILE_COMP2.
 *
 * The third call checks if FILE_COMP1 is smaller than FILE_COMP2.
 *
 * NOTE: In order for this script to run properly, it is advised to set the
 *       preprocessor constants to valid directories and files on the target
 *       system.
 *
\*--------------------------------------------------------------------------*/

#define  SDIR        "C:\\EXAMPLE"
#define  TDIR        "C:\\EXAMPLE\\SOURCE"
#define  FILE_COMP1  "EXAMPLE.DLL"
#define  FILE_COMP2  "EXAMPLE2.DLL"

   STRING szFileName1, szFileName2, szTitle;
   NUMBER nCompareFlag, nResult;

program

   // Set the SRCDIR and TARGETDIR system variables.
   SRCDIR       = SDIR;
   TARGETDIR    = TDIR;
   szFileName1  = FILE_COMP1;
   szFileName2  = FILE_COMP2;

   szTitle      = "FileCompare Example";
   nCompareFlag = COMPARE_VERSION;

/*--------------------------------------------------------------------------*\
 *
 * The following compares the version of the two files.  For each result
 * returned, a message is displayed.
 *
\*--------------------------------------------------------------------------*/
   nResult = FileCompare(szFileName1, szFileName2, nCompareFlag);

   switch (nResult)

      case EQUALS:
         SprintfBox(INFORMATION, szTitle, "%s is the same version as %s.",
                    szFileName1, szFileName2);

      case FILE_NOT_FOUND:
         SprintfBox(INFORMATION, szTitle, "One or both files could not be " +
                   "found.");

      case GREATER_THAN:
         SprintfBox(INFORMATION, szTitle, "%s is a newer version than %s.",
                    szFileName1, szFileName2);
      case LESS_THAN:
         SprintfBox(INFORMATION, szTitle, "%s is an older version than %s.",
                    szFileName1, szFileName2);

      case OTHER_FAILURE:
         SprintfBox(INFORMATION, szTitle, "FileCompare failed.");
   endswitch;

   nCompareFlag = COMPARE_DATE;

/*--------------------------------------------------------------------------*\
 *
 * The following compares the dates of the two files.  A message is displayed
 * if szFileName1 was created earlier than szFileName2 .
 *
\*--------------------------------------------------------------------------*/
   if (FileCompare(szFileName1, szFileName2, nCompareFlag) = LESS_THAN) then

      SprintfBox(INFORMATION, szTitle, "%s was created earlier than %s.",
                 szFileName1, szFileName2);
   endif;

   nCompareFlag = COMPARE_SIZE;
/*--------------------------------------------------------------------------*\
 *
 * The following compares the sizes of the two files.  A nessage is displayed
 * if szFileName1 is larger than szFileName2
 *
\*--------------------------------------------------------------------------*/
   if (FileCompare(szFileName1, szFileName2, nCompareFlag) = GREATER_THAN) then

      SprintfBox(INFORMATION, szTitle, "%s is smaller than %s.", szFileName1,
                 szFileName2);
   endif;

endprogram

// Source file: Is5fn070.rul
```

FileDeleteLine

Syntax

FileDeleteLine (szFileName, nStartLineNum, nEndLineNum);

Description

The FileDeleteLine function deletes a range of lines (including the starting line and ending line) from a text file using a starting and ending line number. This function works on line-oriented text files with lines whose length is less than 512 bytes (Windows 3.1) or 1,024 bytes (Windows NT 4.0 and Windows 95). FileDeleteLine will not work with binary files. You can use FileDeleteLine with the FileGrep function to search and delete text lines in a file.

When using FileDeleteLine, there is no need to open or close the file with OpenFile or CloseFile.

Parameters

szFileName
The name of the file that contains the lines you want to delete. InstallShield uses the system variable SRCDIR as the path of the file.

nStartLineNum
The number of the first line you want to delete.

nEndLineNum
The number of the last line for deletion. You can set this parameter to DELETE_EOF. This action causes InstallShield to delete from nStartLineNum to the end of the file.

Return values

0
FileDeleteLine successfully deleted the specified lines from the file.

< 0
FileDeleteLine failed due to one of these conditions:

FILE_NOT_FOUND
InstallShield could not find the file in szFileName.

FILE_RD_ONLY
The file is write protected.

LINE_NUMBER
One of the line numbers you specified is less than zero, or the line number does not exist in the file.

OUT_OF_DISK_SPACE
There is insufficient space on the disk drive to complete the specified operation.

OTHER_FAILURE
Some other unspecified error has occurred.

{button ,JI(`LANGREF.HLP>Examples',`FileDeleteLine_example')} Example

{button ,AL(`FileGrep;FileInsertLine',0,`',`')} See also

FileDeleteLine example

```
/*--------------------------------------------------------------------------*\
 *
 * This example illustrates the usage of the FileDeleteLine function.
 *
 * FileGrep is used to search for a line in a file that contains the word
 * PATH.  FileDeleteLine is then used to delete that particular line.
 * FileInsertLine is then called to insert a new string into the EXAMPLE_BAT
 * file.
 *
 * NOTE: In order for this script to run properly, it is advised to set the
 *       preprocessor constants to a valid directory and file on the target
 *       system.
 *
\*--------------------------------------------------------------------------*/

#define  SDIR         "C:\\EXAMPLE"
#define  EXAMPLE_BAT  "EXAMPLE.BAT"

   STRING szFileName, szSearchStr, szTitle, svReturnLine, szNewString, szMsg;
   NUMBER nvLineNumber, nFlag, nInsertFlag;

program

   SRCDIR = SDIR;

   szFileName  = EXAMPLE_BAT;
   szSearchStr = "PATH";
   nFlag       = RESTART;
   szTitle     = "FileGrep, FileDeleteLine, & FileInsertLine";

   if (FileGrep (szFileName, szSearchStr, svReturnLine, nvLineNumber,
                 nFlag) < 0) then

      MessageBox("FileGrep failed.", SEVERE);
   else

      szMsg = "FileGrep found %s in %s.\n\nLine: %d";
      SprintfBox(INFORMATION, szTitle, szMsg, szSearchStr, svReturnLine,
                 nvLineNumber);
   endif;

   // This deletes the line identified by FileGrep above.
   if (FileDeleteLine (szFileName, nvLineNumber, nvLineNumber) < 0) then

      MessageBox("FileDeleteLine failed.", SEVERE);
   else

      szMsg = "FileDeleteLine successfully deleted line %d.";
      SprintfBox(INFORMATION, szTitle, szMsg, nvLineNumber);
   endif;

   // This sets the new string to be inserted.
   szNewString = "PATH = C:\\WINDOWS\\BIN;C:\\BIN;C:\\ISHIELD;";
   nInsertFlag = AFTER;

   if (FileInsertLine (szFileName, szNewString, nvLineNumber, nInsertFlag) < 0) then

      MessageBox("FileInsertLine failed.", SEVERE);
   else

      szMsg = "FileInsertLine successfully inserted line %s.";
      SprintfBox(INFORMATION, szTitle, szMsg, szNewString);
   endif;

endprogram

// Source file: Is5fn071.rul
```

FileGrep

Syntax

FileGrep (szFileName, szSearchStr, svReturnLine, nvLineNumber, nFlag);

Description

The FileGrep function searches a file for the specified string. If found, the entire line and the line number are returned. The search is not case-sensitive. FileGrep works on line-oriented text files with lines whose length is less than 512 bytes (Windows 3.1) or 1,024 bytes (Windows NT 4.0 and Windows 95). FileGrep will not work with binary files.

When using FileGrep, there is no need to open or close the file with OpenFile or CloseFile.

Parameters

szFileName
The name of the file you want to search. InstallShield uses the system variable SRCDIR as the path of the file.

szSearchStr
The string you are searching for in the file.

svReturnLine
FileGrep uses svReturnLine to return the line where szSearchStr was found.

nvLineNumber
FileGrep uses nvLineNumber to return the line number where szSearchStr was found. Line numbering starts on line 0.

nFlag
Specify the flag you want to set. These constants are available:

CONTINUE
FileGrep retrieves the next occurrence (if any) of the search string.

RESTART
Use this constant the first time you call FileGrep. FileGrep returns the first instance of the string in svReturnLine.

Return values

0
FileGrep found the specified string.

< 0
FileGrep failed due to one of these conditions:

END_OF_FILE
The end of file was reached without finding the search string.

FILE_NOT_FOUND
InstallShield was unable to find the file in szFileName.

FILE_LINE_LENGTH
The line exceeds the maximum length allowed.

OTHER_FAILURE
An unspecified error has occurred.

{button ,JI(`LANGREF.HLP>Examples',`FileGrep_example')} Example

{button ,AL(`FileDeleteLine;FileInsertLine;FindFile',0,`',`')} See also

FileGrep example

```
/*--------------------------------------------------------------------------*\
 *
 * This example illustrates the uses of the FileGrep function.
 *
 * FileGrep is called to search for the first line in a file containing the
 * word PATH.  The results are displayed in a message box. Note that the
 * FileGrep function is not case sensitive.
 *
\*--------------------------------------------------------------------------*/

#define  SOURCE_DIR  "C:\\EXAMPLE"
#define  SOURCE_FILE "EXAMPLE.BAT"

   STRING    svLine, szNewString, svReturnLine, szMsg;
   NUMBER  nvLineNumber, nResult;

program

   SRCDIR = SOURCE_DIR;

   nResult = FileGrep(SOURCE_FILE, "PATH", svReturnLine, nvLineNumber,
                      RESTART);

   switch(nResult)
      case FILE_NOT_FOUND:

         MessageBox("FILE NOT FOUND", WARNING);
         abort;
      case FILE_LINE_LENGTH:

         MessageBox("FILE LINE LENGTH", WARNING);
         abort;
      case OTHER_FAILURE:

         MessageBox("OTHER FAILURE", WARNING);
         abort;
   endswitch;

   while(nResult != END_OF_FILE)

      szMsg = "FileGrep successful.\n\nLine #%d '%s'";
      SprintfBox(INFORMATION, "FileGrep", szMsg, nvLineNumber, svReturnLine);

      nResult = FileGrep(SOURCE_FILE, "PATH", svReturnLine, nvLineNumber,
                         CONTINUE);

   endwhile;

endprogram

// Source file: Is5fn072.rul
```

FileInsertLine

Syntax

FileInsertLine (szFileName, szInsertLine, nLineNumber, nInsertFlag);

Description

The FileInsertLine function inserts or replaces a line using line numbers. You can use FileInsertLine with FileGrep, which finds lines and returns their line numbers.

FileInsertLine works on line-oriented text files with lines that are no longer than 512 bytes (Windows 3.1) or 1,024 bytes (Windows NT and Windows 95). InstallShield assumes that any line longer than the maximum allowed length indicates a binary file and causes FileInsertLine to fail.

When using FileInsertLine, there is no need to open or close the file with OpenFile or CloseFile.

Parameters

szFileName
The name of the file you want to insert szInsertLine into. FileInsertLine uses the system variable SRCDIR as the path of the file.

szInsertLine
The string you are inserting into the file.

nLineNumber
The line number where you want to insert szInsertLine. The first possible line number is 0.

nInsertFlag
Specify where you want to place szInsertLine. The following constants are available:

BEFORE
Insert the line before nLineNumber.

AFTER
Insert the line after nLineNumber.

APPEND
Append szInsertLine to the line indicated by nLineNumber.

REPLACE
Replace the line indicated by nLineNumber with szInsertLine.

Return values

0
FileInsertLine successfully inserted the line into the specified file.

FILE_LINE_LENGTH
Indicates that the length of the line exceeds the maximum length allowed for text files.

FILE_NOT_FOUND
FileInsertLine was unable to find the file in szFileName.

FILE_RD_ONLY
Indicates that the file is write protected.

LINE_NUMBER
Indicates that one of the line numbers you specified is less than zero, or the line number does not exist in the file.

OUT_OF_DISK_SPACE
Indicates that there is insufficient space on the disk drive to complete the specified operation.

OTHER_FAILURE
An unspecified error has occurred.

{button ,JI(`LANGREF.HLP>Examples',`FileInsertLine_example')} Example

{button ,AL(`FileDeleteLine;FileGrep;FindAllFiles;FindFile',0,`',`')} See also

FileInsertLine example

```
/*--------------------------------------------------------------------------*\
 *
 * This example illustrates the usage of the FileInsertLine function.
 *
 * The AskText dialog box is displayed to prompt the user for a line to add.
 * This line is then added to the top of the TARGET_FILE text file.
 *
 * FileInsertLine is then called again to append the same line to the first,
 * leaving two copies of the same line at the top of the TARGET_FILE file.
 *
 * NOTE: In order for this script to run properly, it is advised to set the
 *       preprocessor constants to a valid directory and file on the target
 *       system.
 *
\*--------------------------------------------------------------------------*/

#define  SDIR         "C:\\EXAMPLE"
#define  TARGET_FILE  "EXAMPLE.BAT"

   STRING szQuestion, szFileName, szInsertLine, szTitle, szMsg;
   NUMBER nLineNumber, nInsertFlag;

program

   SRCDIR       = SDIR;
   szQuestion   = "Please type a line to insert into EXAMPLE.BAT.";
   szInsertLine = "This line will be inserted.";
   AskText(szQuestion, szInsertLine, szInsertLine);


   szFileName   = TARGET_FILE;
   nLineNumber  = 0;
   nInsertFlag  = BEFORE;

/*--------------------------------------------------------------------------*\
 *
 * The following inserts the szInsertLine string to the top of the text file.
 *
\*--------------------------------------------------------------------------*/
   if (FileInsertLine(szFileName, szInsertLine, nLineNumber,
                      nInsertFlag) < 0) then

      MessageBox("FileInsertLine failed.", SEVERE);
   else

      szTitle = "FileInsertLine Example";
      szMsg   = "%s successfully inserted into line %d of %s file.";
      SprintfBox(INFORMATION, szTitle, szMsg, szInsertLine,
                 nLineNumber, szFileName);
   endif;

   nInsertFlag  = APPEND;

/*--------------------------------------------------------------------------*\
 *
 * The following appends the same string to the same line.
 *
\*--------------------------------------------------------------------------*/
```

```
   if (FileInsertLine(szFileName, szInsertLine, nLineNumber,
                      nInsertFlag) < 0) then

      MessageBox("FileInsertLine failed.", SEVERE);
   else

      szTitle = "FileInsertLine Example";
      szMsg   = "%s successfully appended into line %d of %s file.";
      SprintfBox(INFORMATION, szTitle, szMsg, szInsertLine,
                 nLineNumber, szFileName);
   endif;


endprogram

// Source file: Is5fn073.rul
```

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

The name of the directory where you want to begin the search.

nOp

Specify if you want the subdirectories of szDir included in the list. The following mutually exclusive constants are available:

EXCLUDE_SUBDIR

Subdirectories are not included in the list.

INCLUDE_SUBDIR

Subdirectories are included in the list.

listDirs

The name of a valid string list where the directory names will be placed.

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

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
```

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

Enter the name of the directory where you want to start searching for the files.

szFileName

Enter the file specification you want to search for. You can use wild card characters in this parameter.

svResult

InstallShield sets this parameter to the fully-qualified path of the first matching file it found. If the function fails, InstallShield does not alter this parameter.

nOp

Use this parameter to specify whether you want the search to start at the beginning of the directory in szDir or if you want it to resume from the point where it stopped after a previous search. These mutually exclusive constants are available:

CONTINUE

Resumes the search where the previous search stopped. You must call this function again with nOp set to CONTINUE to search for additional occurrences of szFileName.

RESET

Starts the szFileName search from the beginning of the directory in szDir.

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

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
```

FindFile

Syntax

FindFile (szPath, szFileName, svResult);

Description

The FindFile function searches a subdirectory for a specified file. InstallShield returns the first file that matches the file specification in the parameter svResult.

Parameters

szPath
Enter the name of the subdirectory you want to search. Only the subdirectory you specify in this parameter is searched.

szFileName
Enter the name of the file you want to search for. You can use wild card characters in this parameter.

svResult
Returns the name of the first file that matches szFileName. This parameter contains the filename only—it does not contain the path of the file.

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

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
```

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

Enter the fully-qualified name of the file for which you want to retrieve the information.

nType

Use this parameter to specify what type of file information you want. If the information you want is a number, GetFileInfo places it in nvResult. If the information is a string, GetFileInfo places it in svResult. These constants are available:

FILE_ATTRIBUTE

Sets nvResult to a value indicating whether the file is normal, read-only, hidden, system, directory, and/or archived. See Comments.

FILE_DATE

Returns the file date in svResult in the format YYYY\MM\DD.

FILE_SIZE

Returns the size in bytes in nvResult.

FILE_TIME

Returns the file time in svResult in the format HH:MM:SS.

nvResult

If the information you request is a number, it appears in this variable. For example, if you ask for the size of a file, GetFileInfo places the number of bytes in nvResult.

svResult

If you ask for the time or date, they appear as string values in svResult. Time is returned as HH:MM:SS. The date is returned as YYYY\MM\DD.

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

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.

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
                                                                      //
```

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
Enter the name of the handle that opens the file.

svLine
Enter a string variable large enough to hold the lines of text you are receiving.

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 for 16-bit setups and 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

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
```

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 you are opening. Use this handle to identify the file when you use other file-related InstallShield functions.

szPath
Enter only the path of the file you want to open.

szFileName
Enter only the 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

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
```

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:

- Text file in append mode.
- Text file in read-only mode.
- Binary file in read/write mode.
- 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

Use this parameter to specify which mode you want to use to open a file. These constants are available:

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',0,`',`')} See also

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
```

ReadBytes

Syntax

ReadBytes (nFileHandle, svString, nIndex, nBytes);

Description

The ReadBytes function reads a specific number of bytes from a file starting at the current file pointer location. When this function returns, InstallShield relocates the file pointer to the new position at the end of the bytes read from the file.

You must open the file in binary mode by calling OpenFileMode and OpenFile before you can read from the file.

The parameter nIndex is an index into the value specified by the svString. Use the parameter nBytes to specify how many bytes beyond the parameter nIndex to read from the file. If the nIndex plus nBytes is a value larger than the length of the svString, only the number of bytes from the index of the string to the end of the string are read from the file. For example, if svString is declared to be 100 bytes long, the parameter nIndex is declared as 50 bytes and the parameter nBytes is 75 bytes, only the bytes between 49 and 99 (50 bytes instead of 75 bytes) are read.

Parameters

nFileHandle
Enter the file handle to a file opened in binary mode.

svString
Contains the bytes read from the file and returned by this function. This variable must be long enough to accommodate the requested number of bytes.

nIndex
Enter the index into svString where the bytes are written. In most cases you enter 0 in this parameter, so the bytes are copied to the first location of svString.

nBytes
Enter the number of bytes you want to read from the file. Bytes are read starting from the current location of the file pointer. InstallShield relocates the file pointer as the bytes are read. The maximum number of bytes that can be read at one time is 32K.

Return values

X
Indicates that the function successfully read bytes from the file, where X is the actual number of bytes returned in svString.

< 0
Indicates that the function was unable to successfully read from the file.

{button ,JI(`LANGREF.HLP>Examples',`ReadBytes_example')} Example

{button ,AL(`OpenFile;OpenFileMode;WriteBytes',0,`',`')} See also

ReadBytes example

```
/*--------------------------------------------------------------------------*\
 *
 * This example illustrates the usage of the ReadBytes and SeekBytes function.
 *
 * SeekBytes is called to search the binary file and to set the cursor to the
 * location in the file to be read.  ReadBytes then reads this portion of the
 * file, setting a string to the bytes read.  This string is then displayed in
 * a message box.
 *
 * NOTE: The EXAMPLE_DIR and EXAMPLE_BIN constants must be set to an actual
 *       directory and binary file on the target system.
 *
\*--------------------------------------------------------------------------*/

#define EXAMPLE_DIR "EXAMPLE"
#define EXAMPLE_BIN "EXAMPLE.BIN"

   STRING svString;
   NUMBER nvFileHandle, nFileHandle, nIndex, nBytes, nPosition;

program

   // Set the file mode to read/write.
   OpenFileMode(FILE_MODE_BINARY);

   // Open a binary file.
   if (OpenFile(nvFileHandle, EXAMPLE_DIR, EXAMPLE_BIN) < 0) then

      SprintfBox(SEVERE, "CopyBytes Example", "Could not open %s.",
                 EXAMPLE_BIN);
      abort;
   endif;

   // Set SeekBytes variables.
   nPosition = 15;
   nFileHandle = nvFileHandle;
/*--------------------------------------------------------------------------*\
 *
 * The following sets the cursor to the location of the company name in the
 * binary file.  Starting from the beginning of the file, the cursor is moved
 * fifteen bytes to the beginning of the bytes which contain the company name.
 *
\*--------------------------------------------------------------------------*/
   SeekBytes(nFileHandle, nPosition, FILE_BIN_START);

   // Set ReadBytes variables.
   nIndex = 0;
   nBytes = 28;
/*--------------------------------------------------------------------------*\
 *
 * The following reads the first twenty-eight bytes after the cursor in the
 * binary file.  These bytes are set to svString, and the cursor is relocated.
 *
\*--------------------------------------------------------------------------*/
   if (ReadBytes(nFileHandle, svString, nIndex, nBytes) < 0) then

      MessageBox("ReadBytes failed.", SEVERE);
   else
```

```
      // Display the string.
      SprintfBox(INFORMATION, "ReadBytes Example", "bytes read: %s", svString);
   endif;

   // Close the file.
   CloseFile(nvFileHandle);

endprogram

// Source file: Is5fn127.rul
```

RenameFile

Syntax

RenameFile (szFileOld, szFileNew);

Description

The RenameFile function changes the name of a file. Enter only a filename in each parameter, not the path. RenameFile uses the system variable SRCDIR as the path for the parameter szFileOld, and the system variable TARGETDIR as the path for the parameter szFileNew.

You can also use RenameFile to move a file. To move a file, make SRCDIR equal to the current directory of the file and assign the new directory to TARGETDIR.

You can rename only one file with each RenameFile statement. You cannot use wild card characters in this function.

Parameters

szFileOld

Enter only the name of the file you want to rename. Do not enter the path. RenameFile uses the path in the system variable SRCDIR.

szFileNew

Enter only the new name of the file. Do not enter the path. RenameFile uses the path in the system variable TARGETDIR.

Return values

0

Indicates that the function successfully changed the filename.

< 0

Indicates that the function was unable to change the filename.

{button ,JI(`LANGREF.HLP>Examples',`RenameFile_example')} Example

{button ,AL(`CopyFile;FindFile',0,`',`')} See also

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.";
      SprintfBox(INFORMATION, 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
```

SeekBytes

Syntax

SeekBytes (nFileHandle, nBytes, nPosition);

Description

The SeekBytes function repositions the file pointer within an open binary file. You can move the file pointer a specific number of bytes relative to its current position or relative to the beginning or end of the file. Before calling SeekBytes, you must open the file in binary mode by calling OpenFileMode and OpenFile. When you are finished writing bytes to the file, call CloseFile to close the file.

Parameters

nFileHandle

Enter the file handle of a file opened in the binary mode.

nBytes

Enter the number of bytes you want to move the file pointer relative to the position specified by nPosition. If you enter a positive number, this function moves the file pointer toward the end of the file. If you enter a negative number, this function moves the file pointer toward the beginning of the file.

nPosition

Use this parameter to specify where you want to move the pointer relative to where it is currently located. These constants are available:

FILE_BIN_CUR

The file pointer is moved relative to the current location of the file pointer.

FILE_BIN_END

The file pointer is moved backwards starting from the end of the file.

FILE_BIN_START

The file pointer is moved forward starting from the beginning of the file.

Return values

0

Indicates that the function successfully repositioned the pointer.

< 0

Indicates that the function was unable to reposition the pointer.

{button ,JI(`LANGREF.HLP>Examples',`SeekBytes_example')} Example

{button ,AL(`CloseFile;OpenFile;OpenFileMode;ReadBytes;WriteBytes',0,`',`')} See also

SeekBytes example

```
/*--------------------------------------------------------------------------*\
 *
 * This example illustrates the usage of the ReadBytes and SeekBytes function.
 *
 * SeekBytes is called to search the binary file and to set the cursor to the
 * location in the file to be read.  ReadBytes then reads this portion of the
 * file, setting a string to the bytes read.  This string is then displayed in
 * a message box.
 *
 * NOTE: The EXAMPLE_DIR and EXAMPLE_BIN constants must be set to an actual
 *       directory and binary file on the target system.
 *
\*--------------------------------------------------------------------------*/

#define EXAMPLE_DIR "EXAMPLE"
#define EXAMPLE_BIN "EXAMPLE.BIN"

   STRING svString;
   NUMBER nvFileHandle, nFileHandle, nIndex, nBytes, nPosition;

program

   // Set the file mode to read/write.
   OpenFileMode(FILE_MODE_BINARY);

   // Open a binary file.
   if (OpenFile(nvFileHandle, EXAMPLE_DIR, EXAMPLE_BIN) < 0) then

      SprintfBox(SEVERE, "CopyBytes Example", "Could not open %s.",
                 EXAMPLE_BIN);
      abort;
   endif;

   // Set SeekBytes variables.
   nPosition = 15;
   nFileHandle = nvFileHandle;
/*--------------------------------------------------------------------------*\
 *
 * The following sets the cursor to the location of the company name in the
 * binary file.  Starting from the beginning of the file, the cursor is moved
 * fifteen bytes to the beginning of the bytes which contain the company name.
 *
\*--------------------------------------------------------------------------*/
   SeekBytes(nFileHandle, nPosition, FILE_BIN_START);

   // Set ReadBytes variables.
   nIndex = 0;
   nBytes = 28;
/*--------------------------------------------------------------------------*\
 *
 * The following reads the first twenty-eight bytes after the cursor in the
 * binary file.  These bytes are set to svString, and the cursor is relocated.
 *
\*--------------------------------------------------------------------------*/
   if (ReadBytes(nFileHandle, svString, nIndex, nBytes) < 0) then

      MessageBox("ReadBytes failed.", SEVERE);
   else
```

```
      // Display the string.
      SprintfBox(INFORMATION, "ReadBytes Example", "bytes read: %s", svString);
   endif;

   // Close the file.
   CloseFile(nvFileHandle);

endprogram

// Source file: Is5fn127.rul
```

SetFileInfo

Syntax

SetFileInfo (szPathFile, nType, nResult, szResult);

Description

The SetFileInfo function sets a file's date, time, or attributes. Attributes indicate whether the file is normal, archived, hidden, read-only, or system. You can change only one of these types of characteristics with a single SetFileInfo statement. You must use a separate statement to make each change. However, in one SetFileInfo statement you can change more than one of the file's attributes.

When the parameter nType is FILE_DATE or FILE_TIME and the parameter nResult is 0 (zero), SetFileInfo retrieves the file's date or time. However, the GetFileInfo function is specifically designed to retrieve file information and is recommended for that purpose.

Parameters

szPathFile

Enter the path and filename of the file whose characteristic you want to change.

nType

Use this parameter to determine which file characteristic to change. These following constants are available:

FILE_ATTRIBUTE

Indicates the file's attributes (ARCHIVED, HIDDEN, NORMAL, READONLY, or SYSTEM).

FILE_DATE

Indicates the date of the file. To change the file's date, set nResult equal to zero and enter the new date in szResult. If nResult is not equal to zero, the function will return the file's existing date in szResult.

FILE_TIME

Indicates the time of the file. To change the file's time, set nResult equal to zero and enter the new time in szResult. If nResult is not equal to zero, the function will return the file's existing time in szResult.

nResult

Enter the new values that you want in nResult. To change the file's date or time, enter 0 (zero) in this parameter. To change the file's access attributes, use one of these constants:

FILE_ATTR_ARCHIVED

Indicates that the file will be an archived file.

FILE_ATTR_HIDDEN

Indicates that the file will be a hidden file.

FILE_ATTR_NORMAL

Indicates that the file will be a normal file.

FILE_ATTR_READONLY

Indicates that the file will be a read-only file.

FILE_ATTR_SYSTEM

Indicates that the file will be a system file.

szResult

Enter the new or existing file date or time. The date format is YYYY/MM/DD. The time format is HH:MM:SS, using a 24-hour clock format, with midnight at 00:00:00 (or 24:00:00). The seconds must be a multiple of 2.

Return values

0

Indicates that the function successfully set the file's characteristic.

< 0

Indicates that the function was unable to set the file's characteristic.

Comments

When the characteristic you want to change is a string, enter a zero in the parameter nResult, as shown in the examples below. When the attribute you want is a number, enter a null string ("") in the szResult parameter, as shown below.

{button ,JI(`LANGREF.HLP>Examples',`SetFileInfo_example')} Example

{button ,AL(`FindAllFiles;FindFile;GetFileInfo',0,`',`')} See also

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
```

WriteBytes

Syntax

WriteBytes (nFile, svString, nIndex, nBytes);

Description

The WriteBytes function writes a specific number of bytes to a file opened in the binary mode. This function starts writing bytes at the current file pointer location. Before calling WriteBytes, you must open the file by calling OpenFileMode(FILE_MODE_BINARY) and then calling OpenFile.

The parameter nIndex is an index into svString; nBytes specifies how many bytes beyond the value of nIndex you want to write to the file. If nIndex plus nBytes exceeds the length of svString, InstallShield writes only the number of bytes from the index into the string to the end of the string. For example, if svString is 100 bytes long, nIndex is 50 and nBytes is 75, only the bytes between 51 and 100 (50 bytes instead of 75 bytes) are written to the file.

Parameters

nFile
Enter the file handle of the file opened in binary mode.

svString
The string containing the bytes you want to write to the output file. You cannot enter a string literal in this parameter—first assign the string to svString. You can specify an index into this string for the starting location of the bytes you want to write.

nIndex
Enter the starting location in svString where the bytes are written to the output file. The first byte is at index location 0.

nBytes
Enter the number of bytes you want to write to the output file.

Return values

X
Where X is the number of bytes actually written.

< 0
Indicates that the function was unable to write the bytes.

{button ,JI(`LANGREF.HLP>Examples',`WriteBytes_example')} Example

{button ,AL(`ReadBytes;SeekBytes',0,`',`')}   See also

WriteBytes example

```
/*--------------------------------------------------------------------------*\
 *
 * This example illustrates the usage of the WriteBytes function.
 *
 * WriteBytes is called to write bytes from szString to a binary file.
 *
 * NOTE: In order for this script to run properly, it is necessary for the
 *       preprocessor constants to be set to a valid directory and file on the
 *       target system.
 *
\*--------------------------------------------------------------------------*/

#define EXAMPLE_DIR "EXAMPLE"
#define EXAMPLE_BIN "EXAMPLE.BIN"

   STRING  szQuestion, svString, svResult[28];
   NUMBER  nvFileHandle, nPosition, nFileHandle, nFile, nIndex, nBytes;

program

   // Set the file mode to read/write.
   OpenFileMode(FILE_MODE_BINARY);

   // Open the file and get the handle.
   if (OpenFile(nvFileHandle, "EXAMPLE", "EXAMPLE.BIN") < 0) then
      MessageBox("Could not open the file.", SEVERE);
      abort;
   endif;

   szQuestion = "Please enter your company name. You may enter up to " +
                "twenty-seven characters.";

   // Ask user for company name.
   AskText(szQuestion, "InstallShield Corporation", svResult);

   // Set SeekBytes variables.
   nPosition = 15;
   nFileHandle = nvFileHandle;

   // Set the cursor to the company name in the binary file.
   SeekBytes(nFileHandle, nPosition, FILE_BIN_START);

   // Set WriteBytes variables.
   nFile    = nvFileHandle;
   svString = svResult;
   nIndex   = 0;
   nBytes   = 27;

/*--------------------------------------------------------------------------*\
 *
 * The following writes the user's company name into the example binary file.
 * WriteBytes begins where SeekBytes set the cursor, writing over 27 bytes
 * with the characters entered in AskText.
 *
\*--------------------------------------------------------------------------*/

   if (WriteBytes (nFile, svString, nIndex, nBytes) < 0) then
      MessageBox("WriteBytes failed.", SEVERE);
      abort;
   else
      MessageBox("Bytes successfully written to file.", INFORMATION);
   endif;

   // Close the file
   CloseFile(nvFileHandle);

endprogram

// Source file: Is5fn185.rul
```

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

Enter the file handle of an open file. The handle is obtained from OpenFile or CreateFile.

szLine

Enter a string containing the text you want 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 "quote." ", you can use the following code:

```
WriteLine(nvFileHandle, 'This string contains a double "quote."');
```

{button ,JI(`LANGREF.HLP>Examples',`WriteLine_example')} Example

{button ,AL(`CloseFile;CreateFile;OpenFile;OpenFileMode',0,`',`')} See also

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.";
      SprintfBox(INFORMATION, szTitle, szMsg, szFileName);
   endif;

   // Close the open file.
   CloseFile(nvFileHandle);

endprogram
```

```
// Source file: Is5fn200.rul
```

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

Enter the name of file you want to copy. You can copy multiple files using wild card characters in this parameter, but you cannot include a path. This parameter uses SRCDIR as its path.

szTargetFile

Currently, szTargetFile is ignored. However, a string value must be present. Enter a null string ("") or wildcard expression such as `"*.*"`.

nOp

Use this parameter to specify the type of copy operation you want to perform. These constants are available:

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

- 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).
- 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

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 varables.
   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
```

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.

GetDiskSpace

Syntax

GetDiskSpace (szDrive);

Description

Use the GetDiskSpace function to determine how much free disk space is available on a particular drive.

Parameters

szDrive

Enter the letter of the drive you want to check followed by a colon.

Return values

XXXX

The number of bytes free on the specified drive. The maximum value returned is 2 GB. Free disk space greater than that still returns as 2 GB.

< 0

Indicates that the function was unable to return 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

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
```

GetEnvVar

Syntax

GetEnvVar (szParameter, svValue);

Description

The GetEnvVar function retrieves the current value of an environment variable.

Parameters

szParameter
Enter the name of the variable you want to retrieve a value for.

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 functions;Advanced batch file functions',0,`',`')} See also

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
```

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',0,`',`')} See also

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
```

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

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
```

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

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
```

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:

```
svResult[0]=64+nvResult;
```

nvResult: 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 his parameter is passed *to* 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. For information about filtering language-dependent files based on the target system's language, 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.
IS_WIN32S - Operating system is Win32s.
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 Share.exe is loaded.
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: Returns the major version of Microsoft Windows.
svResult: A string in the format ##.## for the major and minor versions of Windows.

nItem: WINMINOR
nvResult: Returns the minor version of Microsoft Windows.
svResult: A string in the format ##.## for the major and minor versions 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. See the Win32s SDK documentation for more information.
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',0,`',`')} See also

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
```

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
The name of a valid string list. This list will contain the valid drive letters when the function is completed.

nDriveType
Specify the type of drive you are searching for. The following constants are available:

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

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
```

GetWindowHandle

Syntax

GetWindowHandle (nHwndFlag);

Description

The GetWindowHandle function gets the handle of the main window of the setup.

Parameters

nHwndFlag
HWND_INSTALL is the only constant available for this parameter. It specifies that you want to retrieve the window handle of InstallShield's main window.

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

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, nInstallHwnd;

program

   nHwndFlag = HWND_INSTALL;
/*--------------------------------------------------------------------------*\
 *
 * GetWindowHandle retrieves the handle of the installation main window.
 *
\*--------------------------------------------------------------------------*/
   nInstallHwnd = GetWindowHandle(nHwndFlag);

   if (nInstallHwnd < 0) then

      MessageBox("GetWindowHandle failed.", SEVERE);
   else

      SprintfBox(INFORMATION, "INFO", "Window Handle: %i", nInstallHwnd);
   endif;

endprogram

// Source file: Is5fn088.rul
```

Is

Syntax

Is (nIsFlag, szIsData);

Description

The Is function retrieves information commonly needed in a script.

Parameters

nIsFlag

Use one of the following constants to specify the type of information you want to retrieve. You can specify only one constant in each Is function call.

DIR_WRITEABLE
Can I write to the directory specified in szIsData?

FILE_EXISTS
Does the file specified in szIsData exist?

FILE_LOCKED
Is the file locked?

FILE_WRITEABLE
Can I write to the file specified in szIsData?

MATH_COPROCESSOR
Does a math coprocessor exist in the target system?

PATH_EXISTS
Does the path specified in szIsData 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 szIsData 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?

szIsData

The information you pass to this parameter varies depending on the constant used in the parameter nIsFlag, as shown below:

DIR_WRITEABLE
Enter the fully-qualified path to be checked.

FILE_EXISTS
Enter the fully-qualified filename.

FILE_LOCKED
Enter the fully-qualified filename.

FILE_WRITEABLE
Enter the fully-qualified filename.

MATH_COPROCESSOR
szIsData is ignored.

PATH_EXISTS

Enter the fully-qualified path.

USER_ADMINISTRATOR
szIsData is ignored.

VALID_PATH
Enter the fully-qualified path.

WINDOWS_SHARED
szIsData 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

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
```

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.

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
The name of a valid number list.

nItem
The numeric element you want to add to the list.

nPlacementFlag
Specify where you want to put nItem with respect to the current element. The element you are adding to the list will go either before or after the current element. The following constants are available:

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

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;

   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
```

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
Enter the name of a valid string list.

szString
Enter the string you want to add to the list.

nPlacementFlag
Specify where you want to put szString with respect to the current element. The string you are adding to the list will go either before or after the current element. The following constants are available:

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

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
```

ListCount

Syntax

ListCount (listID);

Description

Use the ListCount function to obtain the number of elements in a list.

Parameters

listID
The name of a valid 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

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
```

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

Specify the type of list you want to build. The following list types are available:

NUMBERLIST

Specifies the list you are building is a number list.

STRINGLIST

Specifies the list you are building is 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

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);

   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
```

ListCurrentItem

Syntax

ListCurrentItem (listID, nvItem);

Description

The ListCurrentItem function retrieves the current element from the number list you specify in listID.

Parameters

listID
Enter the name of a valid number list whose current element you want to retrieve.

nvItem
Contains the number of the current element in the list returned by the function.

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

- This function works only with number lists.
- 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

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
```

ListCurrentString

Syntax

ListCurrentString (listID, svString);

Description

The ListCurrentString function retrieves the current element from the string list you specify in listID.

Parameters

listID
Enter the name of a valid string list whose current element you want to retrieve.

svString
Contains the number of the current element in the list returned by the function.

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

- This function works only with string lists.
- 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

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
```

ListDeleteItem

Syntax

ListDeleteItem (listID);

Description

The ListDeleteItem function removes the current element from the number list you specify in listID.

Parameters

listID
Enter the name of a valid number list whose current element you want to remove.

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

- This function works only with number lists.
- 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

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
```

ListDeleteString

Syntax

ListDeleteString (listID);

Description

The ListDeleteString function removes the current element from the string list you specify in listID.

Parameters

listID
Enter the name of a valid string list whose current element you want to remove.

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

- This function works only with string lists.
- 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

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
```

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

Enter the name of the string or number list you want to destroy.

Return values

0

Indicates that the function successfully destroyed the list, removing all the elements of the specified list from memory.

< 0

Indicates that the function was unable to destroy the list.

Comments

- This function works with both string and number lists. Once you destroy a list, do not use that listID in any list function.
- 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

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

)
```

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
Enter the name of a valid number list whose elements you want to search.

nItem
Enter the item you are searching for 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

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
```

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
Enter the name of a valid string list whose elements you want to search.

szString
Enter the string you are searching for in the list. InstallShield performs a case-sensitive comparison.

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

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
```

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
Enter the name of a valid number list whose first element you want to retrieve.

nvItem
Contains the first element of the number list returned by this function.

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

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
```

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
Enter the name of a valid string list whose first element you want to retrieve.

svString
Contains the first element of the string list returned by this function.

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

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
Enter the name of a valid number list whose element you want to retrieve.

nvItem
Contains the item after the current element in the number list. This 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

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
```

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
Enter the name of a valid string list whose element you want to retrieve.

svString
Contains the string after 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

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
Enter the name of a valid empty string list created with the ListCreate function. This function reads the file and dumps each line of the file into each element of the string list.

szFile
Enter the fully-qualified name of the file you are creating the list from in this parameter.

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

- This function detects the newline characters at the end of each string and uses the characters as delimiters for each element in the list.
- This function operates on string lists and text files only.

{button ,JI(`LANGREF.HLP>Examples',`ListReadFromFile_example')} Example

{button ,AL(`ListProcessing',0,`',`')} See also

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
```

ListSetCurrentItem

Syntax

ListSetCurrentItem (listID, nItem);

Description

The ListSetCurrentItem function assigns the value of nItem to the current element in a number list.

Parameters

listID
Enter the name of a valid number list whose current element you want to update.

nItem
Enter the numeric value you want to replace the current element with.

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

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
```

ListSetCurrentString

Syntax

ListSetCurrentString (listID, szString);

Description

The ListSetCurrentString function assigns the value of szString to the current element in the string list.

Parameters

listID
Enter the name of a valid string list whose current element you want to update.

szString
Enter the string you want to replace the current element with.

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

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
```

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
Enter the name of a valid list (either string or item) whose current element you want to update.

nIndex
Enter the number of the location you want to set as the current location. The numbering convention for this parameter begins with zero (0). In addition to a number, you can enter one of these 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

- 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.
- This function works on both string and number lists.

{button ,JI(`LANGREF.HLP>Examples',`ListSetIndex_example')} Example

{button ,AL(`ListProcessing',0,`',`')} See also

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
```

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
Enter the name of a valid string list you would like to write into a file.

szFileName
Enter the fully-qualified name of the file you are creating on the disk to accept the list elements.

Return values

0
Function successful.

< 0
Function failed.

{button ,JI(`LANGREF.HLP>Examples',`ListWriteToFile_example')} Example

{button ,AL(`ListProcessing',0,`',`')} See also

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
```

InstallShield
www.installshield.com

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

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 Functions;Long Filename Format;Long Filenames and Double Quotation Marks;Long Filenames and 16 bit Applications',0,`',`')} See also

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 Functions;Long Filename Format;Long Filenames and Double Quotation Marks;Long Filenames and 16 bit Applications',0,`',`')} See also

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

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 Functions;Long Filename Format;Long Filenames and Double Quotation Marks;Long Filenames and 16 bit Applications'), , ,)} See also

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, szPath is returned unmodified and no error code is returned

Parameters

svPath

Enter the short filename that you want to convert. When you call the function, the long filename is returned in this variable.

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',0,`',`')} See also

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
```

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 quotes 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

Enter the long filename that you want to convert. When you call the function, the converted filename is returned in this variable.

nParameter

Determines whether the quotation marks are added to the long path or removed from the long path. Use one of these constants:

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',0,`',`')} See also

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
```

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, szPath is returned unmodified and no error code is returned

Parameters

svPath

Enter the long filename that you want to convert. When you call the function, the short filename is returned in this variable.

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

- For an explanation of long filenames, see Long Filenames.
- Because LongPathToShortPath can succeed only if the specifed 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 inless the current folder is "C:\Program Files". Use the ChangeDirectory function to change the current folder when necessary before calling LongPath to ShortPath 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',0,`',`')} See also

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.

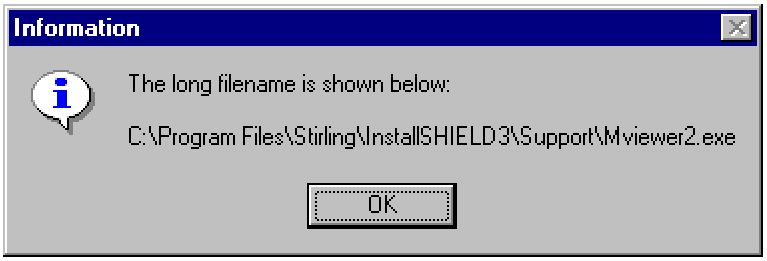
Relative path

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.

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
```



The long filename is shown below:

C:\Program Files\Stirling\InstallSHIELD3\Support\Mviewer2.exe

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.

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

Specify the type of operation you want to perform. The following constants are available:

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 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;VerUpdateFile;XCopyFile',0,`',`')} See also

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
```

DoInstall

Syntax

DoInstall (szInsFile, szCmdLine, lWait);

Description

The DoInstall function launches another InstallShield setup. The second setup is executed immediately when this function is called. The parameter lWait specifies whether or not the calling script will wait for the launched setup to complete before continuing to execute.

The setup called by DoInstall must have been created with the same major and minor version (and preferably build number) of InstallShield as the setup that is launching it. Attempts to use DoInstall to launch a setup created with any other version of InstallShield will not be successful.

If the second setup launches another executable application, the first setup has no control over the launched application and has no way to monitor the status of the launched application.

Parameters

szInsFile
The fully-qualified name of the compiled script file (.ins) you want to launch.

szCmdLine
A string containing an InstallShield command line. You can specify any valid startup InstallShield command line here; note, however, that the -f parameter is not appropriate in this context and should not be used.

lWait
Specifies whether or not the calling setup will wait for the launched setup to terminate before continuing. The following constants are available:

NOWAIT
The   calling will should continue immediately after launching the second setup.

WAIT
The calling setup will wait for the first setup to terminate before continuing.

Return values

1 or 2
The called setup successfully terminated. Control resumes in the calling setup with the statement after the DoInstall function.

-1
InstallShield found the specified setup but was unable to launch it.

-2
InstallShield was unable to find the .ins script file specified by szInsFile.

All other negative values
An unspecified error has occurred.

Comments

- When a second setup is launched with DoInstall, that second setup will be same setup type (16-bit or 32-bit) as the setup that launched it. This holds true even if the second setup was not designed for the same target platform as the first setup and regardless of which InstallShield engine files are present in the directory where the Setup.ins file is located. To avoid errors in setups that are launched with DoInstall, be sure that both the launching setup and the launched setup are of the same type; that is, both are intended to be 16-bit or 32-bit.

n The following files must be present in the folder where the Setup.ins file is located: _user1.cab, _sys1.cab, Os.dat, and Layout.bin. If these files are not present, the second setup will not be launched successfully. For this reason, it is recommended that you copy all the files created by the Media Build Wizard into the folder from which you will be launching the second setup. If you are launching the second setup from the installation support folder (SUPPORTDIR) of the first setup, it is recommended that you place these files in the appropriate folder in the setup files pane of the first setup so that they will be decompressed automatically in the installation support folder of the first setup when that setup initializes.

{button ,JI(`LANGREF.HLP>Examples',`DoInstall_example')} Example

{button ,AL(`LaunchApp;LaunchAppAndWait',0,`',`')} See also

DoInstall example

```
/*--------------------------------------------------------------------------*\
 *
 * This example illustrates the use of the DoInstall function.
 *
 * The following script runs the MessageBox example script using the DoInstall
 * function.
 *
 * Note: To make this example work correctly you must also do the following:
 *
 * 1.   Create a second setup project. This setup will be launched by the
 *      DoInstall function. This project should include an up to date built
 *      media.
 *
 * 2.   Create a new folder named 'Second' in the disk1 folder of this setup.
 *
 * 3.   Copy the disk# folder(s) from the second setup into the newly created
 *      'Second' directory.
 *
 *      The second setup should then be launched successfully.
\*--------------------------------------------------------------------------*/

#define SECOND_INSTALL_PATH SRCDIR ^ "Second\\Disk1"
#define SECOND_INSTALL_FILENAME "Setup.ins"

NUMBER nReturn;

STRING szTemp;

program

   MessageBox ("About to launch the second setup.", INFORMATION);

   nReturn = DoInstall (SECOND_INSTALL_PATH ^ SECOND_INSTALL_FILENAME, "", WAIT);

   if( nReturn=1 || nReturn=2) then

        MessageBox("The second setup was launched successfully.", INFORMATION);

    else

        SprintfBox(SEVERE,"","DoInstall failed with a return code of %d.",nReturn);

    endif;

endprogram

// Source file: Is5fn058.rul
```

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

Specify the event you want to trap. The following constants are available:

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

The name of the label to which you want the program to jump if the specified button or accelerator key is pressed. Do not define this label as a numeric value or as a string variable.

If you enter -1 in this parameter, a currently defined handler will be canceled and the default handler will be reinstated. This is useful when you have a custom handler you want to use in a portion of the script, but then want to revert to the default handler.

Return values

0

Handler successfully created the handle.

< 0

Handler was unable to create the handle.

Comments

- Do not define the label name as a numeric value or as a string variable.
- 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.
- 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.

- As with Help handlers, you can also define and execute custom and Sd dialog-specific EXIT handlers.
- You can create custom exit handlers for any dialog box except for those created by AskDestPath and AskOptions. These two are built-in dialog boxes and will use the default exit routine only if the user presses the Cancel button.
- In the script, EXIT and HELP handler labels and their associated code must be placed before the endprogram statement.

{button ,JI(`LANGREF.HLP>Examples',`Handler_example')} Example

{button ,AL(`Do',0,`',`')} See also

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
```

MessageBeep

Syntax

MessageBeep (nReserved);

Description

The MessageBeep function sounds a standard warning beep.

Parameters

nReserved

This parameter must be 0.

Return values

This function has no return values.

{button ,JI(`LANGREF.HLP>Examples',`MessageBeep_example')} Example

{button ,AL(`MessageBox',0,`',`')} See also

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
```

SendMessage

Syntax

SendMessage (nHwnd, nMsg, nwParam, nlParam);

Description

The SendMessage function sends a message to one or more windows. SendMessage does not return control to the setup script until the message has been processed. The SendMessage function is a direct pass-through to the Windows API SendMessage. Consult the Windows programming documentation for detailed information.

Parameters

nHwnd
Enter the handle that identifies the window you want to receive the message.

nMsg
Enter the message you want to send to the window(s).

nwParam
Enter any additional message information.

nlParam
Enter any additional message information.

Return values

The InstallShield SendMessage function returns the value it receives from calling the Windows API of the same name. The return value depends on the message received by the Windows API SendMessage. Consult the Windows programming documentation for detailed information on messages returned by the Windows API SendMessage.

Comments

To send a message using the parameter nMsg or to handle return values, you must define constants in your script that are equivalent to the constants defined in Windows.h. You cannot use `#include` to include Windows.h in your script.

{button ,JI(`LANGREF.HLP>Examples',`SendMessage_example')} Example

{button ,AL(`CmdGetHwndDlg;FindWindow;GetWindowHandle',0,`',`')} See also

SendMessage example

```
/*--------------------------------------------------------------------------*\
 *
 * This example illustrates the usage of the FindWindow and SendMessage
 * functions.
 *
 * FindWindow is called to retrieve a window handle.  This handle is used by
 * SendMessage to send commands to the APPLICATION window.
 *
 * NOTE: In order for this script to work properly, the APPLICATION constant
 *       should be set to the Windows notepad executable, and the LICENSE
 *       constant should be set to any standard Notepad text file.
 *
\*--------------------------------------------------------------------------*/

#define  APPLICATION  "EXAMPLE\\NOTEPAD.EXE"
#define  LICENSE      "EXAMPLE\\LICENSE.TXT"
#define  WINDOW       "NOTEPAD"

// Define Windows messages to InstallShield
#define  WM_SYSCOMMAND       0x0112
#define  SC_MINIMIZE         0xF020
#define  SC_MAXIMIZE         0xF030

   HWND hWnd;

program

   // Launch the application window.
   LaunchApp(APPLICATION, LICENSE);

   // Disable background for visibility.
   Disable(BACKGROUND);

/*--------------------------------------------------------------------------*\
 *
 * The following sets hWnd to the handler specified in the WINDOW constant.
 *
\*--------------------------------------------------------------------------*/
   hWnd = FindWindow(WINDOW, "");

   if (hWnd != NULL) then

      Delay(3);
/*--------------------------------------------------------------------------*\
 *
 * Send a system command to the window.  This command maximizes APPLICATION.
 *
\*--------------------------------------------------------------------------*/
      SendMessage(hWnd, WM_SYSCOMMAND, SC_MAXIMIZE, 0);

     Delay(3);
/*--------------------------------------------------------------------------*\
 *
 * Send a system command to the window.  This command minimizes APPLICATION.
 *
\*--------------------------------------------------------------------------*/
      SendMessage(hWnd, WM_SYSCOMMAND, SC_MINIMIZE, 0);
   endif;
```

```
endprogram

// Source file: Is5fn162.rul
```

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

Contains the formatted string returned by Sprintf. You can use svResult as you would use any other string variable.

szFormat

Enter a string that includes a format specifier for each argument to be included in the message.

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. If you use more than nine format specifiers and arguments, InstallShield will report a compilation error message. If the number of format specifiers does not match the number of arguments or if the type of 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

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
```

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

Use this parameter to specify which action you want to perform after terminating the setup. These constants are available:

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

- 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.
- 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.
- 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

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
```

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
Enter 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

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
```

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
Enter 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

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
```

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 the desired path string you must save it to an appropriate file if needed.

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.

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

Enter the directory you want to add to the path buffer. You can enter either the full path and directory, or the directory name only.

szRefDir

Enter the directory name you are using as a reference to locate szDir in the path. You can enter either the fully-qualified path or just the directory name.

bRefDir

Use this parameter to specify if szRefDir includes a fully-qualified path or the directory name only. If you enter FULL in this parameter, InstallShield assumes szRefDir contains a fully-qualified path. If you enter PARTIAL, InstallShield assumes szRefDir contains the directory name only.

bPosition

Enter where you want szDir added to the path temporary buffer. You can use one of these two constants: BEFORE or AFTER. BEFORE specifies that you want to add szDir before szRefDir. AFTER specifies that you want to add szDir after szRefDir.

If you enter a null string ("") in szRefDir, this parameter specifies if szDir is added as the first or last entry of the path buffer. If you enter BEFORE, the directory is added as the first directory in the path temporary buffer. If you enter AFTER, the directory is added as the last directory 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 functions',0,`',`')}	See also

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
```

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

Enter the directory you want to remove from the path buffer. You can enter either the full path and directory name, or the directory name only.

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.

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 functions',0,`',`')} See also

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
```

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
Enter the name of the directory you are searching for in the path buffer. You can enter either the fully-qualified directory and path, or the directory name only.

svResult
Contains the full directory and path found in the path buffer returned by the function.

bDir
Use this parameter to specify whether szDir contains a fully-qualified or an unqualified directory name. These constants are available:

FULL
Specifies szDir contains a fully-qualified directory name.

PARTIAL
Specifies szDir contains the directory name only.

bSearch
Use this parameter to specify if you want the function to continue the search from where it left off after the previous search, or to start the search from the beginning of the path buffer.

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 functions',0,`',`')} See also

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
```

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
Contains the contents of the temporary path string buffer returned by the function.

Return values

0
Indicates that the function successfully retrieved the path string currently stored in the temporary path string 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 functions',0,`',`')} See also

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
```

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

Enter the name of the directory you want to reposition in the path buffer. You can enter either the full path and directory name, or the directory name only.

szRefDir

Enter the directory relative to where szDir will be repositioned. If you enter a null string ("") in this parameter, szDir becomes the first or the last item.

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

Use this parameter to specify where you want the function to move szDir in the path temporary buffer. These constants are available:

AFTER

Specifies that szDir is added in the search path after szRefDir.

BEFORE

Specifies that szDir is added in the search path before szRefDir.

If szRefDir is a null string (""), the parameter bPosition indicates whether szDir becomes the first or last entry in the path buffer. If you enter BEFORE, the directory is added as the first directory in the path buffer. If you enter AFTER, the directory becomes the last directory 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 functions',0,`',`')} See also

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
```

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.

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.

The PathSet function validates the data you pass to it. The value must contain a drive letter, a colon, and a root directory (a backslash).

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

The path to store in a temporary path buffer. You can then use other Path functions to modify/ manipulate it. szString 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 functions',0,`',`')} See also

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
```

Initialization file functions

Initialization file functions obtain information from and copy information to the initialization and profile files. An initialization file is a special ASCII file that contains key name-value pairs. The key name-value pairs represent run-time options for applications. You can also access and update private initialization file and system initialization files. The following list briefly describes each initialization file function.

Changes made to .ini files with AddProfString, ReplaceProfString, or WriteProfString can be logged for uninstallation and removed by unInstallShield. However, there are some important restrictions to be aware of. For more information, see Uninstalling initialization (.ini) file entries.

AddProfString
Adds a non-unique key to a section of the .ini file.

GetProfInt
Returns an integer from an .ini file.

GetProfString
Returns a string from an .ini file.

ReplaceProfString
Replaces a string in a profile (.ini) file.

WriteProfString
Writes a string to an .ini file.

Related Function

SdShowFileMods
Creates a dialog box displaying proposed file changes and offering options on how to proceed.

AddProfString

Syntax

AddProfString (szFileName, szSectionName, szKeyName, szValue);

Description

The AddProfString function unconditionally adds a profile string to an .ini file. Use AddProfString only to add non-unique keys, such as those found in the [386Enh] section of the System.ini file (device = ...). AddProfString adds the line KEY=VALUE to the beginning of the specified .ini file section. It does not replace or update an existing key. To update an existing non-unique key, call ReplaceProfString. To add a unique key or to update an existing unique key's value in an .ini file, call WriteProfString.

Changes made to .ini files can be logged for uninstallation and removed by unInstallShield. However, there are some important restrictions to be aware of. For more information, see Uninstalling initialization (.ini) file entries.

Parameters

szFileName
The fully-qualified name of the .ini file to which you are adding the profile string. If you do not specify a path, Windows assumes the file is located in the Windows directory.

szSectionName
The name of the section of the .ini file where you want to add the profile string. Do not include section name delimiting brackets ([ ]). If the section does not exist, InstallShield creates it for you.

szKeyName
The name of the key you are placing in the section. The line szKeyName = szValue is placed at the beginning of the section, even if a szKeyName entry already exists.

szValue
The value you are placing after szKeyName.

Return values

0
AddProfString successfully added the specified profile string to the .ini file.

< 0
AddProfString was unable to add the profile string.

Comments

AddProfString does not use the Windows API to change the .ini files. The Windows API cannot handle the types of changes possible with AddProfString.

{button ,JI(`LANGREF.HLP>Examples',`AddProfString_example')}   Example

{button ,AL(`GetProfString;ReplaceProfString;WriteProfString',0,`',`')}   See also

AddProfString example

```
/*--------------------------------------------------------------------------*\
 *
 * This example illustrates the use of the AddProfString and GetProfString
 * functions.
 *
 * AddProfString is used to add a profile string to the EXAMPLE_INI file.
 *
 * GetProfString is used to find the value of a key within that .INI file.
 *
 * NOTE: In order for this script to run properly, you must set the
 *       EXAMPLE_INI constant to an existing file in the target system.
 *
\*--------------------------------------------------------------------------*/

#define EXAMPLE_INI   "Example\\Example.ini"
#define NEW_SECTION   "New Section"
#define NEW_KEY       "New Key"
#define NEW_VALUE     "Test"

   STRING szFileName, szSectionName, szKeyName, szValue, svResult, szFile;

program

   szFileName    = EXAMPLE_INI;
   szSectionName = NEW_SECTION;
   szKeyName     = NEW_KEY;
   szValue       = NEW_VALUE;

   // Add profile string to the EXAMPLE_INI file.
   if (AddProfString (szFileName, szSectionName, szKeyName, szValue) = 0) then

      // Retrieve the value of szKeyName from the EXAMPLE_INI file.
      if (GetProfString(szFileName, szSectionName, szKeyName, svResult) = 0) then
         // Display szKeyName and its current value.
         SprintfBox(INFORMATION, "GetProfString", "%s is set to: %s", szKeyName,
svResult);
     else
         MessageBox("GetProfString failed.", SEVERE);
     endif;

   else
      MessageBox("AddProfString failed.", SEVERE);
   endif;

endprogram

// Source file: Is5fn001.rul
```

GetProfInt

Syntax

GetProfInt (szFile, szSectionName, szKeyName, nvValue);

Description

The GetProfInt function retrieves an integer from an .ini file. GetProfInt works like the Windows API GetPrivateProfileInt with the nDefault parameter specified as 0.

Parameters

szFile
The name of the profile (.ini file) you want to search that contains szSectionName and szKeyName. If you do not specify a path, InstallShield searches for the profile (.ini file) in the Windows directory (WINDIR).

szSectionName
A string that identifies the section name in the .ini file you are searching through. Do not include section name delimiting brackets ([ ]). The search for this string is *not* case sensitive.

szKeyName
A string that contains the key name identifier you are looking for. The search for this string is *not* case sensitive.

nvValue
GetProfInt returns an integer value in nvValue. Due to limitations of the GetPrivateProfileInt function, this function can return only a 16-bit value from the profile. Therefore the maximum value that can be returned is 65,535; larger values may not be returned correctly. If you need to return a larger value, use the general file handling functions, such as FileGrep and FileInsertLine, and then convert the returned string into an integer by calling StrToNum.

Return values

GetProfInt always returns 0.

Comments

As with the Windows API function GetPrivateProfileInt, no error will be returned if an error occurs because the file, section, or key name cannot be found; instead nvValue will contain 0. For that reason, it is not possible to distinguish between an error and a returned value of zero. To distinguish between zero and an error, call GetPrivateProfileInt directly and specify an alternate default value. Consult the Windows or Win32 SDK for more information regarding the usage of GetPrivateProfileInt.

Under Windows NT (*not* Windows 95), some calls to the GetPrivateProfileInt function (and therefore the GetProfInt function) are mapped automatically to the Windows registry instead of the profile. Consult the Win32 SDK for more information about mapping behavior.

{button ,JI(`LANGREF.HLP>Examples',`GetProfInt_example')} Example

{button ,AL(`AddProfString;GetProfString;ReplaceProfString;WriteProfString',0,`',`')} See also

GetProfInt example

```
/*--------------------------------------------------------------------------*\
 *
 * This example illustrates the usage of the GetProfInt function.
 *
 * GetProfInt is called to retrieve the value of a key in the EXAMPLE_INI
 * file.
 *
 * NOTE: In order for this script to run properly, it is advised to set the
 *       preprocessor constants to a valid directory and file on the target
 *       system.
 *
\*--------------------------------------------------------------------------*/

#define  EXAMPLE_INI  "EXAMPLE\\EXAMPLE.INI"

   STRING szFile, szSectionName, szKeyName;
   NUMBER nvValue;

program

   szFile        = EXAMPLE_INI;
   szSectionName = "Old Section";
   szKeyName     = "Old IntKey";

/*--------------------------------------------------------------------------*\
 *
 * GetProfInt retrieves the value of szKeyName under the szSectionName
 * section.
 *
\*--------------------------------------------------------------------------*/
   if (GetProfInt(szFile, szSectionName, szKeyName, nvValue) < 0) then

      MessageBox("GetProfInt failed.", SEVERE);
   else

      SprintfBox(INFORMATION, "GetProfInt Example", "The value of %s is: %d.",
szKeyName, nvValue);
   endif;

endprogram

// Source file: Is5fn085.rul
```

GetProfString

Syntax

GetProfString (szFile, szSectionName, szKeyName, svResult);

Description

The GetProfString function retrieves a profile string from the specified .ini file. GetProfString works like the Windows API GetPrivateProfileString.

Parameters

szFile

The fully-qualified path of the .ini file that contains the szSectionName and szKeyName you are searching for. If you do not specify a path in this parameter, InstallShield looks for the file in the Windows directory.

szSectionName

The name of the application section of the .ini file you want to search through. Do not include section name delimiting brackets ([]).

szKeyName

The name of the key you want to retrieve from the application section. If you enter a null string (""), InstallShield returns all the application's key names (separated by null characters) in svResult. Key names are not case sensitive.

svResult

GetProfString returns the value retrieved from the .ini file in svResult.

Return values

0

GetProfString successfully returned the value of the profile string.

< 0

GetProfString was unable to return the value.

Comments

A null string ("") in the parameter szKeyName causes all key names in the given section to be enumerated. The key names are returned in the parameter svResult. InstallShield returns all key names in svResult separated by null characters. svResult must be long enough to accept all the key names. Use the StrGetTokens function to extract the individual key names from this string. If InstallShield cannot find the values in szSectionName or szKeyName, it returns a null string ("") in svResult.

GetProfString uses the functions provided by your operating environment's API to access the .ini file. Therefore, InstallShield's functionality may be limited by the operating environment.

{button ,JI(`LANGREF.HLP>Examples',`GetProfString_example')} Example

{button ,AL(`AddProfString;ReplaceProfString;WriteProfString',0,`',`')} See also

GetProfString example

```
/*---------------------------------------------------------------------------*\
 *
 * This example illustrates the use of the AddProfString and GetProfString
 * functions.
 *
 * AddProfString is used to add a profile string to the EXAMPLE_INI file.
 *
 * GetProfString is used to find the value of a key within that .INI file.
 *
 * NOTE: In order for this script to run properly, you must set the
 *       EXAMPLE_INI constant to an existing file in the target system.
 *
\*---------------------------------------------------------------------------*/

#define EXAMPLE_INI   "Example\\Example.ini"
#define NEW_SECTION   "New Section"
#define NEW_KEY       "New Key"
#define NEW_VALUE     "Test"

   STRING szFileName, szSectionName, szKeyName, szValue, svResult, szFile;

program

   szFileName    = EXAMPLE_INI;
   szSectionName = NEW_SECTION;
   szKeyName     = NEW_KEY;
   szValue       = NEW_VALUE;

   // Add profile string to the EXAMPLE_INI file.
   if (AddProfString (szFileName, szSectionName, szKeyName, szValue) = 0) then

      // Retrieve the value of szKeyName from the EXAMPLE_INI file.
      if (GetProfString(szFileName, szSectionName, szKeyName, svResult) = 0) then
         // Display szKeyName and its current value.
         SprintfBox(INFORMATION, "GetProfString", "%s is set to: %s", szKeyName,
svResult);
     else
         MessageBox("GetProfString failed.", SEVERE);
     endif;

   else
      MessageBox("AddProfString failed.", SEVERE);
   endif;

endprogram

// Source file: Is5fn001.rul
```

ReplaceProfString

Syntax

ReplaceProfString (szFile, szSectionName, szKeyName, szOrigValue, szReplaceValue);

Description

The ReplaceProfString function replaces a profile string in an .ini file. This function can replace values of duplicate keys (non-unique keys) such as those found in the [386Enh] section of the System.ini file (device = ...). The function searches for a szKeyName = szOrigValue, and replaces the line. If it is not found, it adds the szKeyName = szReplaceValue line to the beginning of the szSectionName section.

If you are adding unique keys (i.e., keys that are all different for a given section), use the WriteProfString function.

Use this function to replace only non-unique key names, such as the device= line in the System.ini file.

Changes made to .ini files can be logged for uninstallation and removed by unInstallShield. However, there are some important restrictions to be aware of. For more information, see Uninstalling initialization (.ini) file entries.

Parameters

szFile

Enter the name of the initialization file. If you do not include a fully-qualified filename in this parameter, ReplaceProfString searches the Windows directory for the file. If the file does not exist, this function creates the file in the Windows directory.

If this parameter contains a fully-qualified filename and the file does not exist, InstallShield creates it for you. Be aware, however, that the path specified with the filename must already exist.

szSectionName

Enter the section where you want to search for szKeyName. Do not include section name delimiting brackets ([ ]). If the section does not exist, InstallShield creates it for you. The name of the section is case-independent—the string may be any combination of uppercase and lowercase letters.

szKeyName

Enter the name of the key you want to place in the .ini file. If the key does not exist, InstallShield creates it for you.

szOrigValue

Enter the value of the key=value line you want to search for and replace.

szReplaceValue

Enter the new value you want to associate with szKeyName.

Return values

0

Indicates that the function successfully replaced or added the profile string.

< 0

Indicates that the function was unable to replace or add the profile string.

Comments

Windows .ini files are text files. You must use the appropriate version of InstallShield with each operating system.

{button ,JI(`LANGREF.HLP>Examples',`ReplaceProfString_example')} Example

{button ,AL(`AddProfString;WriteProfString',0,`',`')} See also

ReplaceProfString example

```
/*--------------------------------------------------------------------------*\
 *
 * This example illustrates the use of the ReplaceProfString function.
 *
 * ReplaceProfString is called for the first time to replace the szKeyName key
 * value szOrigValue with the value szReplaceValue.
 *
 * ReplaceProfString is called again to replace the newly set value of
 * szKeyName, szReplaceValue, with szOrigValue.
 *
 * NOTE: In order for this script to run properly, it is advised to set the
 *       EXAMPLE_INI constant to a valid initialization file on the target
 *       system.
 *
\*--------------------------------------------------------------------------*/

#define EXAMPLE_INI "EXAMPLE\\EXAMPLE.INI"

   STRING szFile, szSectionName, szKeyName, szOrigValue, szReplaceValue;

program

   szFile = EXAMPLE_INI;
   szSectionName = "Old Section";
   szKeyName = "Old Key";
   szOrigValue = "Sample";
   szReplaceValue = "Newer Example Value";

/*--------------------------------------------------------------------------*\
 *
 * The following replaces szOrigValue with szReplaceValue.
 *
\*--------------------------------------------------------------------------*/
   if (ReplaceProfString (szFile, szSectionName, szKeyName, szOrigValue,
                          szReplaceValue) < 0) then

      MessageBox("ReplaceProfString failed.", SEVERE);
      abort;
   else
      SprintfBox (INFORMATION, "Replacement Successful",
                  "Original:  %s\nNew:  %s", szOrigValue, szReplaceValue);
   endif;

/*--------------------------------------------------------------------------*\
 *
 * The following replaces szReplaceValue with szOrigValue.
 *
\*--------------------------------------------------------------------------*/
   if (ReplaceProfString(szFile, szSectionName, szKeyName, szReplaceValue,
                         szOrigValue) < 0) then;

      MessageBox("ReplaceProfString failed.", SEVERE);
   else

      SprintfBox (INFORMATION, "Replacement Successful",
                  "Original:  %s\nNew:  %s", szReplaceValue, szOrigValue);
   endif;
```

```
endprogram

// Source file: Is5fn130.rul
```

WriteProfString

Syntax

WriteProfString (szFile, szSectionName, szKeyName, szValue);

Description

The WriteProfString function writes a profile string to a specific .ini file. Depending on the values passed to WriteProfString, it can create a section, delete an entire section, create a unique KEY=VALUE entry, delete a KEY=VALUE entry, or update a key's value.

You can use this function to write integer values to the .ini files by converting the integer to a string before writing. A common use of WriteProfString is to associate file extensions with executable files in the [Extensions] section of the Win.ini file under Windows 3.1. Use the AddProfString and ReplaceProfString functions when you want to modify the System.ini file.

Because of the way in which Windows 95 caches file changes you must flush the cache buffer after calls to WriteProfString. (See Comments section below.)

Changes made to .ini files can be logged for uninstallation and removed by unInstallShield. However, there are some important restrictions to be aware of. For more information, see Uninstalling initialization (.ini) file entries.

Parameters

szFile
Enter the fully-qualified name of the .ini file. If the .ini file does not exist, this function creates the file. The specified directory must already exist.

szSectionName
Enter the name of the section where szValue will be copied. Do not include section name delimiting brackets ([ ]). If the section does not exist, this function creates it for you. This parameter is case-independent—you can enter any combination of uppercase and lowercase letters.

szKeyName
Enter the key you want to associate with szValue. If you enter a null string ("") in this parameter, the entire section, including all entries within the section under szSectionName, is deleted. If szKeyName does not exist, it will be created as long as szValue is not a null string (""). If szKeyName exists, it will be updated or deleted, depending in the value of szValue.

szValue
Enter the value you want to write to the .ini File. If you enter a null string (""), the entry in szKeyName is deleted. If you want to delete the szValue but retain the key entry, use a string containing a blank space in this parameter.

Return values

0
Indicates that the function successfully wrote the string to the specified .ini file.

< 0
Indicates that the function was unable to write the string to the specified .ini file.

Comments

- The WriteProfString function uses the Windows API WritePrivateProfileString to access the .ini file. Therefore, its functionality is limited by the functionality provided by the Windows API. Consult the Microsoft Windows manual for more information on .ini files.
- Windows 95 caches .ini files, which can cause a delay in writing changes to the specified files. This in turn

can interfere with subsequent file operations, such as calls to CopyFile and XCopyFile. Therefore, you should flush the cache buffer after using WriteProfString if you are using file operations shortly afterward. 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',`WriteProfString_example')}	Example

{button ,AL(`AddProfString;GetProfString;GetProfInt;ReplaceProfString',0,`',`')}	See also

WriteProfString example

```
/*--------------------------------------------------------------------------*\
 *
 * This example illustrates the use of the WriteProfString function by
 * updating a field in an initialization file in the Windows directory.
 * If the file does not exist, it will be created.
 *
\*--------------------------------------------------------------------------*/

// Define the initialization file name.
#define EXAMPLE_INI "EXAMPLE.INI"

   STRING  szFile, szSectionName, szKeyName, szValue;

program

   // Specify the fully qualified path and filename of the target file.
   // If you replace WINDIR with an invalid path, WriteProfString will fail.
   szFile = WINDIR ^ EXAMPLE_INI;

   // Specify the section, keyword, and value for the call to WriteProfString.
   szSectionName = "Windows";
   szKeyName     = "Keyboard";
   szValue       = "English";

   // Update a field in the test .ini file.
   if (WriteProfString(szFile, szSectionName, szKeyName, szValue) < 0) then
      SprintfBox(SEVERE, "WriteProfString",
                 "%s could not be updated", szFile);
   else

      SprintfBox(INFORMATION, "WriteProfString", "%s was modified.", szFile);
   endif;

endprogram

// Source file: Is5fn186.rul
```

Registry functions

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.

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 limitation 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
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.

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.

Return values

0
Indicates that the function was successful.

< 0
Indicates that the function failed.

{button ,JI(`LANGREF.HLP>Examples',`DeinstallSetReference_example')} Example

{button ,AL(`DeinstallStart;CommitSharedFiles;Disable;Enable;InstallationInfo;RegDBGetItem;RegDBSetItem;SdFinishReboot;VerUpdateFile',0,`',`')} See also

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
```

DeinstallStart

Syntax

DeinstallStart (szDefLogPath, svLogFile, szKey, lStyle);

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

Enter the fully-qualified path where you want the log file 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 for you when 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

Enter the name of the application uninstallation key to be created.

lStyle

Enter 0—no other value is allowed. This parameter is reserved for future use.

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',0,`',`')} See also

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
```

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

Enter the name of your company. InstallShield uses szCompany to create a \<company> key under the [HKEY_LOCAL_MACHINE]\Software key in the registry.

szProduct

Enter the name of the product you are installing. 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

Enter 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

Enter the name of the main executable file for your application. Even if your product uses several executables, enter 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 3.1uses 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',0,`',`')} See also

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
```

RegDBConnectRegistry

Syntax

RegDBConnectRegistry (szRemoteSystem, nKeyType, nReserved);

Description

The RegDBConnectRegistry function creates a connection to a remote registry. If you are trying to open a registry on a remote Windows NT system, you must have administrator privileges. If you are trying to open a registry on a remote Windows 95 system, it must already have Remote Administration enabled. This function is intended for use by system administrators for network installations.

Once you have opened the connection, you can create, delete, or retrieve registry keys, value names, and value pairs on a remote registry much as you would on a local registry. RegDBConnectRegistry allows you to edit only one registry root key each time the remote registry is opened, and you can edit only keys and values under either HKEY_LOCAL_MACHINE or HKEY_USERS. When you call RegDBConnectRegistry, you must specify which root key you want to be able to edit. if you want to edit the other root key or one its subkeys, you must close and re-open the connection.

To edit a remote registry, you must use only the general registry-related functions (listed below), which are designed to work with all registry keys.

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.

RegDBGetKeyValueEx
Retrieves a value from under a key in the registry.

RegDBKeyExist
Checks if a key exists.

RegDBQueryKey
Queries a key for its subkeys and value names.

RegDBSetKeyValueEx
Sets registry entries.

Parameters

szRemoteSystem
Enter the name of the system you want to connect to, such as `"RemoteSys"`.

nKeyType
Enter one of the following constants: HKEY_LOCAL_MACHINE or HKEY_USERS.

nReserved
Reserved for future use. Enter 0 (zero) in this parameter.

Return values

0
Indicates this function successfully established a connection to the system registry.

REGDB_ERR_CONNECTIONEXISTS

A connection to a remote registry already exists. It must be closed using RegDBDisConnectRegistry before you can call RegDBConnectRegistry again.

REGDB_ERR_CORRUPTEDREGISTRY
Indicates that the remote registry is corrupted and cannot be accessed.

REGDB_ERR_INITIALIZATION
Indicates that the registry services could not be initialized. Make sure Remote Administration is enabled and that you have appropriate privileges to be able to write to the registry.

REGDB_ERR_INVALIDHANDLE
The key name provided for the remote registry is not allowed.

REGDB_ERR_INVALIDNAME
Indicates that the system in szRemoteSystem could not be found. Check the name and try again.

-1
Other error.

Comments

Since you set the root key by calling RegDBConnectRegistry, you cannot call RegDBSetDefaultRoot once you have established a connection to a remote registry. Once you have called RegDBDisConnectRegistry, all calls to registry-related functions will affect the local registry, and you can then call RegDBSetDefaultRoot to change the root key.

unInstallShield will uninstall all keys under any key that is logged for uninstallation. Keys created automatically by InstallShield are logged for uninstallation. When you call RegDBSetKeyValueEx to create keys above which there are no keys logged for uninstallation, your keys will not be uninstalled by unInstallShield, regardless of whether logging was enabled or not (you can Enable and Disable logging). However, when you call RegDBCreateKeyEx to create a key above which there are no keys logged for uninstallation while logging is enabled, your key will be logged for uninstallation. Any keys you create under the logged key will be uninstalled when the logged key is removed. Refer to the individual function descriptions for more information.

{button ,JI(`LANGREF.HLP>Examples',`RegDBConnectRegistry_example')} Example

{button ,AL(`RegDBCreateKeyEx;RegDBDeleteKey;RegDBDisConnectRegistry;RegDBSetKeyValueEx',0,`',`')}
See also

RegDBConnectRegistry example

```
/*--------------------------------------------------------------------------*\
 *
 * This example illustrates the usage of the RegDBConnectRegistry and
 * RegDBDisConnectRegistry functions.
 *
 * NOTE: In order for this script to run properly, it is necessary for the
 *       preprocessor constants to be set to a valid remote computer with
 *       remote administration enabled.  Both computers also must have the
 *       remote-registry service enabled.
 *
\*--------------------------------------------------------------------------*/

#define REMOTE "IShield_NT1"

   STRING szRemoteMachine, szKey, szTitle, szMsg;
   NUMBER nKeyType, nReturn;

program

   szTitle = "RegDBConnectRegistry & RegDBDisConnectRegistry";
   szRemoteMachine = REMOTE;
   nKeyType        = HKEY_LOCAL_MACHINE;

   szMsg = "Setup will now connect to %s.";
   SprintfBox(INFORMATION, szTitle, szMsg, szRemoteMachine);

/*--------------------------------------------------------------------------*\
 *
 * Connect to the remote computer's registry.  All registry-related function
 * calls hereafter will alter only the remote computer.
 *
\*--------------------------------------------------------------------------*/

   nReturn = RegDBConnectRegistry(szRemoteMachine, nKeyType, 0);

   if (nReturn < 0) then

      szMsg = "RegDBConnectRegistry failed.\n\nCould not connect to remote " +
              "system.";
      MessageBox(szMsg, SEVERE);
      abort;
   else

      szMsg = "Successfully connected to %s.";
      SprintfBox(INFORMATION, szTitle, szMsg, szRemoteMachine);
   endif;

   // Create a key on the remote computer.
   szKey = "SOFTWARE\\InstallShield\\Test Key";
   nReturn = RegDBCreateKeyEx(szKey, "");

   if (nReturn < 0) then

      szMsg = "RegDBCreateKeyEx failed.\n\n\Could not create key on remote " +
              "machine.";
      MessageBox(szMsg , SEVERE);
   else
```

```
      szMsg = "Successfully created %s on %s.";
      SprintfBox(INFORMATION, szTitle, szMsg, szKey, szRemoteMachine);

      nReturn = RegDBKeyExist(szKey);

      if (nReturn < 0) then

         szMsg = "RegDBKeyExist failed.\n\nRemote key does not exist.";
         MessageBox(szMsg, SEVERE);
      else

         szMsg = "%s exists.";
         SprintfBox(INFORMATION, szTitle, szMsg, szKey);
      endif;
   endif;

   // Delete the created key on the remote computer.
   nReturn = RegDBDeleteKey(szKey);

   if (nReturn < 0) then

      MessageBox("RegDBDeleteKey failed.\n\nRemote key could not be deleted.",
                 INFORMATION);
   else

      szMsg = "Successfully deleted %s on %s.";
      SprintfBox(INFORMATION, szTitle, szMsg, szKey, szRemoteMachine);
   endif;

/*--------------------------------------------------------------------------*\
 *
 * Disconnect from the remote registry.  All registry-related functions
 * hereafter only alter the local registry.
 *
\*--------------------------------------------------------------------------*/
   nReturn = RegDBDisConnectRegistry(0);

   if (nReturn < 0) then
      MessageBox("RegDBDisConnectRegistry failed.\n\nRemote registry still " +
                 "connected.", SEVERE);
   else
      MessageBox("RegDBDisConnectRegistry successful.\n\nRemote registry " +
                 "disconnected.", INFORMATION);
   endif;

endprogram

// Source file: Is5fn536.rul
```

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

Enter the name of the key you want to create under one of the four root keys. Separate different levels in the subkey with a double backslash (\\).

szClass

Enter the class name you are associating 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.

{button ,JI(`LANGREF.HLP>Examples',`RegDBCreateKeyEx_example')} Example

{button ,AL(`Disable;Enable;RegDBSetDefaultRoot;RegDBDeleteKey;RegDBGetKeyValueEx;RegDBKeyExist;RegDBSetKeyValueEx',0,`',`')} See also

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
```

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
Enter the name of the key you want 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',0,`',`')} See also

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
```

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
Enter the name of the registry key that contains the value name you want 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',0,`',`')} See also

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
```

RegDBDisConnectRegistry

Syntax

RegDBDisConnectRegistry (nReserved);

Description

The RegDBDisConnectRegistry function closes a connection to a remote registry that you established by calling RegDBConnectRegistry.

After calling RegDBDisConnectRegistry, all calls to the InstallScript registry-related functions affect the local system's registry. Refer to special registry-related functions for more information.

Parameters

nReserved
Reserved for future use. Enter 0 (zero) in this parameter.

Return values

0
Indicates that this function successfully closed a connection to the registry on the remote system.

< 0
Indicates that this function failed to close the registry connection.

{button ,JI(`LANGREF.HLP>Examples',`RegDBDisConnectRegistry_example')}   Example

{button ,AL(`RegDBConnectRegistry;RegDBCreateKeyEx;RegDBDeleteKey;RegDBSetKeyValueEx',0,`',`')}   See also

RegDBDisConnectRegistry example

```
/*--------------------------------------------------------------------------*\
 *
 * This example illustrates the usage of the RegDBConnectRegistry and
 * RegDBDisConnectRegistry functions.
 *
 * NOTE: In order for this script to run properly, it is necessary for the
 *       preprocessor constants to be set to a valid remote computer with
 *       remote administration enabled.  Both computers also must have the
 *       remote-registry service enabled.
 *
\*--------------------------------------------------------------------------*/

#define REMOTE "IShield_NT1"

   STRING szRemoteMachine, szKey, szTitle, szMsg;
   NUMBER nKeyType, nReturn;

program

   szTitle = "RegDBConnectRegistry & RegDBDisConnectRegistry";
   szRemoteMachine = REMOTE;
   nKeyType        = HKEY_LOCAL_MACHINE;

   szMsg = "Setup will now connect to %s.";
   SprintfBox(INFORMATION, szTitle, szMsg, szRemoteMachine);

/*--------------------------------------------------------------------------*\
 *
 * Connect to the remote computer's registry.  All registry-related function
 * calls hereafter will alter only the remote computer.
 *
\*--------------------------------------------------------------------------*/

   nReturn = RegDBConnectRegistry(szRemoteMachine, nKeyType, 0);

   if (nReturn < 0) then

      szMsg = "RegDBConnectRegistry failed.\n\nCould not connect to remote " +
              "system.";
      MessageBox(szMsg, SEVERE);
      abort;
   else

      szMsg = "Successfully connected to %s.";
      SprintfBox(INFORMATION, szTitle, szMsg, szRemoteMachine);
   endif;

   // Create a key on the remote computer.
   szKey = "SOFTWARE\\InstallShield\\Test Key";
   nReturn = RegDBCreateKeyEx(szKey, "");

   if (nReturn < 0) then

      szMsg = "RegDBCreateKeyEx failed.\n\n\Could not create key on remote " +
              "machine.";
      MessageBox(szMsg , SEVERE);
   else
```

```
      szMsg = "Successfully created %s on %s.";
      SprintfBox(INFORMATION, szTitle, szMsg, szKey, szRemoteMachine);

      nReturn = RegDBKeyExist(szKey);

      if (nReturn < 0) then

         szMsg = "RegDBKeyExist failed.\n\nRemote key does not exist.";
         MessageBox(szMsg, SEVERE);
      else

         szMsg = "%s exists.";
         SprintfBox(INFORMATION, szTitle, szMsg, szKey);
      endif;
   endif;

   // Delete the created key on the remote computer.
   nReturn = RegDBDeleteKey(szKey);

   if (nReturn < 0) then

      MessageBox("RegDBDeleteKey failed.\n\nRemote key could not be deleted.",
                 INFORMATION);
   else

      szMsg = "Successfully deleted %s on %s.";
      SprintfBox(INFORMATION, szTitle, szMsg, szKey, szRemoteMachine);
   endif;

/*--------------------------------------------------------------------------*\
 *
 * Disconnect from the remote registry.  All registry-related functions
 * hereafter only alter the local registry.
 *
\*--------------------------------------------------------------------------*/
   nReturn = RegDBDisConnectRegistry(0);

   if (nReturn < 0) then
      MessageBox("RegDBDisConnectRegistry failed.\n\nRemote registry still " +
                 "connected.", SEVERE);
   else
      MessageBox("RegDBDisConnectRegistry successful.\n\nRemote registry " +
                 "disconnected.", INFORMATION);
   endif;

endprogram

// Source file: Is5fn536.rul
```

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
Enter the value name whose value you want 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 svString.

< 0
Indicates that the function was unable to retrieve the value.

{button ,JI(`LANGREF.HLP>Examples',`RegDBGetAppInfo_example')}   Example

{button ,AL(`InstallationInfo;RegDBSetAppInfo',0,`',`')} See also

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
```

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

Enter the predefined item you want to retrieve. These constants are available:

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.

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 REGBD_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',0,`',`')} See also

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 uninstallation name key: %s.", svValue);
   endif;

endprogram

// Source file: Is5fn241.rul
```

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
Enter the key name whose value you want to retrieve. Separate different levels in the subkey with a double backslash (\\).

szName
Enter the value name under szKey for which you want to retrieve the value. To retrieve the default value for 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',0,`',`')} See also

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
```

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
Enter the name of the key whose existence you want to check. 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',0,`',`')} See also

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
```

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

This string indicates subkey(s) under one of the root keys set by using RegDBSetDefaultRoot. Use backslashes to specify deeper levels of subkeys. This can be a null string (""), in which case the data retrieved is the root key.

nItem

This indicates which items should be placed in the list. Available constants:

REGDB_KEYS

The string list will contain a list of all the subkeys under this key.

REGDB_NAMES

The string list will contain the names of all named values for this key. This constant is not valid under Windows 3.1.

listResults

String list which will contain the results of the query.

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',0,`',`')} See also

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
```

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
Enter the value name whose information you want to set.

nType
Enter the type of data you are setting:

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
Enter the value you are setting for the value name.

nSize
Enter the size, in bytes, of the data you are passing. If you enter -1 in this parameter InstallShield automatically determines 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',0,`',`')} See also

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
```

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
Enter the name of the key you want 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

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.

{button ,JI(`LANGREF.HLP>Examples',`RegDBSetDefaultRoot_example')} Example

{button ,AL(`RegDBCreateKeyEx;RegDBDeleteKey;RegDBGetKeyValueEx;RegDBKeyExist;RegDBSetKeyValueEx' ,0,`',`')} See also

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).
   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
      szMsg = "Successfully deleted %s in HKEY_LOCAL_MACHINE";
      SprintfBox(INFORMATION, szTitle, szMsg, szKey);
   endif;

/*--------------------------------------------------------------------------*\
 *
 * Set the root key to HKEY_CLASSES_ROOT again.
 *
\*--------------------------------------------------------------------------*/

   nRootKey = HKEY_CLASSES_ROOT;
   if (RegDBSetDefaultRoot(nRootKey) < 0) then
      MessageBox("Second call to RegDBSetDefaultRoot failed.", SEVERE);
   else
      MessageBox("Root key successfully set to HKEY_CLASSES_ROOT.", INFORMATION);
   endif;

   if (RegDBDeleteKey(szKey) < 0) then
      MessageBox("RegDBDeleteKey failed.", SEVERE);
   else
      szMsg = "Successfully deleted %s in HKEY_CLASSES_ROOT";
      SprintfBox(INFORMATION, szTitle, szMsg, szKey);
   endif;

endprogram

// Source file: Is5fn246.rul
```

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

Enter the item you want to set. These constants are available:

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

Enter the value you are setting 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 REGBD_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',0,`',`')} See also

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
      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 uninstallation name key: %s.", svValue);
   endif;

endprogram

// Source file: Is5fn241.rul
```

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:

- When they are created using RegDBCreateKeyEx.
- While logging is enabled.
- When they are created automatically by InstallShield during the installation process.
- 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
Enter the name of the key you want to set after you have created it 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
Enter the value name for the value data you want to set. To set the default value of the key specified in szKey, enter a null string ("").

nType
Enter the type of data you are setting. Available constants are:

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
Enter the value you are associating 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

Enter the size, in bytes, of the data you are setting. 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' ,0,`',`')} See also

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
```

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.

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.

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 and company name.

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.

DialogSetInfo

Syntax

DialogSetInfo (nInfoType, szInfoString, nParameter);

Description

The DialogSetInfo function changes the following display elements in InstallShield dialog boxes:

- The image to be displayed
- The style of the check boxes used to obtain end-user selections
- The precision of the values indicating available hard drive space

Changes made by a call to DialogSetInfo remain in effect for the remainder of the setup or until they are changed again by a subsequent call to DialogSetInfo.

Parameters

nInfoType

Use one of the following constants to specify the display feature to be modified:

DLG_INFO_USEDECIMAL

By default, the values displayed to indicate component sizes, available disk space, and required disk space are rounded to the nearest kilobyte or megabyte. Specify this constant when you want these values displayed to the nearest tenth of a kilobyte or megabyte. The following dialogs are affected by this parameter: ComponentDialog, SdComponentDialog, SdComponentDialog2, SdComponentDialogAdv and SdComponentMult.

DLG_INFO_KUNITS

By default, the values displayed to indicate component sizes, available disk space, and required disk space are displayed as a measurement in megabytes. Specify this constant to display these measurements in kilobytes. The following dialogs are affected by this parameter: ComponentDialog, SdComponentDialog, SdComponentDialog2, SdComponentDialogAdv and SdComponentMult.

DLG_INFO_ALTIMAGE

Specifies an alternate bitmap or metafile image to be displayed in the dialog box. If nParameter is set to TRUE, szInfoString should specify the image to be displayed in the dialog box. This parameter applies to all InstallShield dialog boxes that display the standard installation image on the left side of the dialog. For more information, See "When nInfoType is DLG_INFO_ALTIMAGE" in the nParameter description below.

Display effects that have been set with SetDisplayEffect do not apply to alternate images, which are always displayed without any special effects.

DLG_INFO_CHECKSELECTION

Specifies that the selection method will be determined by the constant passed to nParameter.

szInfoString

This string specifies an alternate image when DLG_INFO_ALTIMAGE is passed in nInfoType. The standard bitmap measures 120 x 258. An alternate bitmap should be about this size as well. The bitmap will be centered in the normal location regardless of its size. If the bitmap is larger than 120 x 258, pixels beyond the 120 x 258 range will be clipped automatically. If the bitmap is smaller than 120 x 258, it will be displayed correctly; but it will not be resized or extended.

This parameter is ignored when the default bitmap is being restored or when nInfoType is not DLG_INFO_ALTIMAGE.

nParameter

This parameter works with the parameter nInfoType to control various features of the dialog. When nInfoType is

DLG_INFO_CHECKSELECTION, nParameter specifies the check box style. Use one of the following constants in nParameter to specify check box style:

CHECKBOX
Specifies Windows 3.1-style check boxes.

CHECKBOX95
Specifies Windows 95-style check boxes.

CHECKLINE
Specifies checkline-style check boxes.

CHECKMARK
Specifies checkmark-style check boxes.

When nInfoType is DLG_INFO_ALTIMAGE:

-1
Specifies that dialogs should display the default bitmap.

TRUE
Specifies that the bitmap indicated by szInfoString should be used in subsequent dialogs, as described above under szInfoString.

When nInfoType is either DLG_INFO_KUNITS or DLG_INFO_USEDECIMAL:

TRUE
Specifies that sizes should be displayed as indicated by nInfoType.

FALSE
Specifies that sizes should be displayed in the default style.

Return values

0
Indicates that the function successfully set the style specified.

< 0
Indicates that the function was unable to set the style.

Comment

- To preview the effects of a call to DialogSetinfo, run the InstallShield Dialog Sampler, change the attributes of the dialog boxes (by clicking on Attributes button), then examine the changes in dialogs such as SdComponentDialog2 and SdComponentMult. DialogSetInfo
- You must call DialogSetInfo each time you want to change a particular aspect of a single dialog.
- You can use the DLG_INFO_ALTIMAGE parameter to enable a 16-color, 256-color, or true color (24-bit) bitmap or metafile. Note that color distortion may occur when a 256-color bitmap is displayed on a 16-color system or when a true color bitmap is displayed on a 256-color system. It is recommended that you specify an alternate image that is compatible with the color mode of the target system.

{button ,JI(`LANGREF.HLP>Examples',`DialogSetInfo_example')} Example

{button ,AL(`ComponentDialog;SdComponentDialog;SdComponentDialog2;SdComponentDialogAdv;SdComponentMult;SdAskOptions;SdAskOptionsList;Modifying Sd dialog boxes with DialogSetInfo',0,`',`')} See also

DialogSetInfo example

```
/*--------------------------------------------------------------------------*\
 *
 * This example illustrates the usage of the DialogSetInfo function.
 *
 *The szBmpPath parameter will point to this bitmap with
 * the following concatenated expression:
 * "@" + SUPPORTDIR ^ "[bitmap_filename or reference_number]"
 *
\*--------------------------------------------------------------------------*/

#define ALTBMP "setup.bmp"
#define BMP_PATH "C:\\EXAMPLE"

   STRING  szText, szMsg, szBmpPath;
   STRING  svReturnText;
   NUMBER  nReturn;

program

start:

   // Disable Back button for the first dialog.
   Disable (BACKBUTTON);

   // Display the AskText dialog with its default bitmap.
   szText  = "Default Bitmap.";
   szMsg   = "The bitmap on the left is the default.";
   nReturn = AskText (szMsg, szText, svReturnText);

   // Enable the Back button.
   Enable (BACKBUTTON);

  showalternate:

   szBmpPath = "@"+ BMP_PATH ^ ALTBMP;

/*--------------------------------------------------------------------------*\
 *
 * Set the alternate bitmap for the AskText dialog box.
 *
\*--------------------------------------------------------------------------*/
   DialogSetInfo (DLG_INFO_ALTIMAGE, szBmpPath, TRUE);

   // Set the text for display in the AskText dialog box.
   szText = "Alternate Bitmap.";
   szMsg = "The bitmap on the left is a custom bitmap.  This alternate " +
           "bitmap was displayed using the DLG_INFO_ALTBITMAP option in " +
           "DialogSetInfo.";

   // Display the AskText dialog box with the alternate
   // bitmap. Handle Back button.
   nReturn = AskText (szMsg, szText, svReturnText);

   if (nReturn = BACK) then
      DialogSetInfo (DLG_INFO_ALTIMAGE, "", -1);
      goto start;
   endif;
   // Disable Next button for final dialog.
   Disable (NEXTBUTTON);

/*--------------------------------------------------------------------------*\
 *
 * Call DialogSetInfo to reinstate the default bitmap.  Do this by passing a
 * null string as the second parameter, and -1 as the last parameter.
 *
\*--------------------------------------------------------------------------*/
   DialogSetInfo (DLG_INFO_ALTIMAGE, "", -1);

   // Display AskText with default bitmap back in effect.
   szText  = "Original Bitmap.";
   szMsg   = "The bitmap on the left is the original.";
   nReturn = AskText (szMsg, szText, svReturnText);

   // Enable Next button in case user chooses Back.
   Enable (NEXTBUTTON);

   // Handle Back button.
   if (nReturn = BACK) then
      goto showalternate;
   endif;

endprogram

// Source file: is5fn207.rul
```

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

Enter the text you want to display in the title of the dialog box. To display the default title, "Choose Folder," enter a null string ("").

szMsg

Enter the text you want to display in the dialog box. The text is considered a static control. You can use the %P place holder in your message string to insert the product name (if any) that has been specified by a call to SdProductName function. To display the default instructions for this dialog box, pass a null string ("").

svDir

Enter the name of the directory you want to appear as the default directory. This variable contains the chosen directory after the function executes.

nReserved

Reserved for future use. Enter 0 (zero) in nReserved.

Return values

NEXT

Indicates that the Next button was clicked.

BACK

Indicates that the Back button was clicked.

Comments

You can use the SdProductName function to replace the %P place holder with your product name.

{button ,JI(`LANGREF.HLP>Examples',`SdAskDestPath_example')}   Example

{button ,AL(`SdConfirmNewDir',0,`',`')}   See also

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
```

SdAskOptions

Syntax

SdAskOptions (szTitle, szMsg1, szMsg2, szId, szComponents, nExclusiveFlag);

Description

The SdAskOptions function creates a dialog box that offers installation options. You can use check boxes or radio buttons as selection buttons. The information shown beside the button is retrieved from a group of options. The default number of options is four. You can add or subtract the number of options as necessary in the group.

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 SdAskOptions.

When using SdAskOptions 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
Enter the text you want to appear in the title of the dialog box. To display the default title, "Select Components," enter a null string ("").

szMsg1
Enter a message you want to appear in the dialog box. This static field has an ID of 801. To display the default instructions for this dialog box, enter a null string ("").

szMsg2
Enter a second message you want to appear in the dialog box. This static field has an ID of 802.

szId
Enter an alternate numeric dialog ID in this parameter. Use only numeric IDs expressed in string form (for example, ID 13001 as "13001"). You can copy the SdAskOptions dialog resource, make limited changes to it, give it a unique numeric ID, and call that dialog by passing its ID as a string in szId. Refer to the Comments section, below. To create the standard four-option SdAskOptions dialog box, Enter a null string ("").

szComponents
Enter the name of the component that contains the subcomponent you want to display. The subcomponent are preceded by check boxes or radio buttons. To display all top-level components, enter a null string (""). For more information about specifying components and subcomponents in function calls, click here.

nExclusiveFlag
Use this parameter to specify the type of button you want to appear in the dialog box. These constants are available:

EXCLUSIVE
Specifies radio buttons. Do not use EXCLUSIVE mode if any of szComponents' subcomponents are required components.

NONEXCLUSIVE
Specifies check boxes.

Return values

NEXT

Indicates that the Next button was clicked.

BACK

Indicates that the Back button was clicked.

Comments

You can create more than one dialog box of the SdAskOptions type by copying the SdAskOptions dialog resource (located in _isres.dll) using a resource editor, making limited changes to the copy, and giving it a unique ID. When you call SdAskOptions and pass the ID of the customized copy of the dialog in the parameter szId, the customized copy is displayed. Limit your changes to editing existing static text fields and adding static text fields. Adding controls that require handling is not recommended because it requires changing the SdAskOptions source script.

{button ,JI(`LANGREF.HLP>Examples',`SdAskOptions_example_examples')} Example

{button ,AL(`SdSetupType',0,`',`')} See also

SdAskOptions example

```
/*--------------------------------------------------------------------------*\
 *
 *  This example illustrates the use of SdAskOptions, which creates a dialog
 *  box that offers installation options.
 *
 *  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. 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 COMPONENT          ""
#define ASKDESTTITLE       "Destination Location"
#define ASKDESTMSG         "Choose a destination location for the application."
#define SDASKOPTSTITLE     "Component Selection"
#define SDASKOPTSMSG1      "Select components to install."
#define SDASKOPTSMSG2      "Your selections will be used to effect file transfer."
#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
   // Get a TARGETDIR target location.
    TARGETDIR = PROGRAMFILES ^ APPBASE_PATH;
    Disable( BACKBUTTON );
    AskDestPath(ASKDESTTITLE, ASKDESTMSG, 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 );

    SdAskOptions(SDASKOPTSTITLE, SDASKOPTSMSG1, SDASKOPTSMSG1,
                 "", COMPONENT, NONEXCLUSIVE);

    // 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: Is5fn132.rul
```

SdAskOptionsList

Syntax

SdAskOptionsList (szTitle, szMsg, szComponents, nStyle);

Description

The SdAskOptionsList function creates a dialog box that displays a list of components for a Custom installation.

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 SdAskOptionsList.

When using SdAskOptionsList 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
Enter the text you want to appear in the title of the dialog box. To display the default title, "Select Components," enter a null string ("").

szMsg
Enter the text you want to appear in the dialog box. To display the default instructions for this dialog box, enter a null string ("").

szComponents
Enter a null string ("") to display all main components. Specify a component name to display all of that component's sub-components. For more information about specifying components and subcomponents in function calls, click here.

nStyle
Use one of the following two constants in this parameter:

EXCLUSIVE
Allows the user to select only one item from the list. Do not use EXCLUSIVE mode if any of szComponents' subcomponents are required components.

NONEXCLUSIVE
Allows the user to select more than one item from the list, including multiple non-contiguous selections. Also causes two buttons to be displayed: Select All and Clear All, which allow selection of all options or clearing of all selections with the click of a button.

Return values

NEXT
Indicates that the Next button was clicked.

BACK
Indicates that the Back button was clicked.

{button ,JI(`LANGREF.HLP>Examples',`SdAskOptionsList_example')} Example

{button ,AL(`ComponentAddItem;;ComponentDialog;SdDisplayTopics',0,`',`')} See also

SdAskOptionsList example

```
/*---------------------------------------------------------------------------*\
 *
 *  This example demonstrates SdAskOptionsList. It also includes calls to
 *  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.
 *
 *  IMPORTANT: Make sure there are no passwords assigned to any components
 *  since this script does not call ComponentValidate.
 *
 *  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. PlaceWindow calls in the user-defined
 *  SetUpFileTransfer function handle placement of billboards, the progress
 *  indicator, and the information gauges.
 *
 *  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 to install."
#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"

// Global variable declarations.
STRING  svLogFile,  szProgram;
BOOL    bInitStepsDone;
NUMBER  nvDisk, nResult;

// Function declarations.
prototype SetUpFileTransfer();
prototype HandleComponentError(NUMBER);
prototype FinishSetup();

program
    // Set up the screen, target location, etc.
    SetUpFileTransfer();

    // Disable the Back button, which is not needed.
    Disable( BACKBUTTON );

    // Let user select from top-level components.
    SdAskOptionsList(COMP_SELECT_TITLE, COMP_SELECT_MSG, "", NONEXCLUSIVE);

    // 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.
    nResult=ComponentMoveData( MEDIA, nvDisk, 0);

    // Handle any error in ComponentMoveData call.
    HandleComponentError( nResult );

    // Add icon to Start Programs menu and wrap things up in general.
    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_COMPONENTDIALOG 1
#define SD_ASKOPTIONSLIST  1

#include "sddialog.rul"

// Source file: Is5fn616.rul
```

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
Enter the text you want to appear in the title of the dialog box. To display the default title, "Welcome," enter a null string ("").

szMsg
Enter a null string ("") in the parameter szMsg, unless you use a resource editor to modify the SdBitmap dialog box to display a message. See the Comments section, below.

szBitmap
Enter the name of the bitmap you want to display. InstallShield searches in SUPPORTDIR for the bitmap, unless you specify a full qualified path and filename for the bitmap.

Return values

NEXT
Indicates that the Next button was clicked.

BACK
Indicates that the Back button was clicked.

Comments

- 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.
- 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.
- 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.
- This function does not support transparent bitmaps. If you use a transparent bitmap with this function, the

transparent portions will be displayed normally.

- 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

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
```

SdComponentDialog

Syntax

SdComponentDialog (szTitle, szMsg, svDir, szComponents);

Description

Use the SdComponentDialog function to create a dialog box that displays a list of components in the current media that the user can install and the amount of disk space each component will occupy. This dialog provides the same functionality as that of SdComponentDialogAdv, except for the way in which the check box selection is provided.

View sample dialog

The destination directory can be changed by using the Browse button; the available disk space on other drives can be checked by using Disk Space button.

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 SdComponentDialog.

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 SdComponentDialog to get end-user selections.
4. Assign to MEDIA the value saved in step 1.

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 SdComponentDialog 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 SdComponentDialog 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

Enter the text you want to appear in the title of the dialog box. To display the default title, "Select Components," enter a null string ("").

szMsg

Enter the text you want to appear in the dialog box. To display the default instructions for this dialog box, enter a null string ("").

svDir

Enter the name of a folder you want to appear as the default folder. This variable contains the chosen folder after the function executes. 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

Enter a null string ("") to display all main components. Specify a component name to display all of that component's sub-components. 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

The Disk Space... button has an ID of 101. This button automatically displays the available disk space dialog. You can remove this button/option if you desire. The Directory static field requires an ID of 851. The list box ID has a multiple selection style.

{button ,JI(`LANGREF.HLP>Examples',`SdComponentDialog_example')} Example

{button ,AL(`Displaying icons in component dialogs;ComponentAddItem;ComponentDialog;ComponentGetItemSize;ComponentIsItemSelected;ComponentListItems;ComponentSelectItem;ComponentTotalSize;SdComponentDialog2;SdComponentMult;SdComponentDialogAdv',0,`',`')} See also

SdComponentDialog 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;
        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
```

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.

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

Enter the text you want to appear in the title of the dialog box. To display the default title, "Select Components," enter a null string ("").

szMsg

Enter the text you want to appear in the dialog box. To display the default instructions for this dialog box, enter a null string ("").

szDir

Enter 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

Enter a null string ("") to display all main components. Specify a component name to display all of that component's sub-components. 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

- The Change button will be active only if the selected component has any subcomponents. Otherwise, it will turn gray.
- 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(`Displaying icons in component dialogs;ComponentAddItem;ComponentDialog;ComponentGetItemSize;ComponentIsItemSelected;ComponentListItems;ComponentSelectItem;SdComponentDialog;SdComponentMult;SdComponentDialogAdv',0,`',`')} See also

SdComponentDialog2 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
```

SdComponentDialogAdv

Syntax

SdComponentDialogAdv (szTitle, szMsg, svDir, szComponents);

Description

The SdComponentDialogAdv function creates a dialog box that displays a list of components in the current media that the end user can install and the amount of disk space each component would occupy. This dialog provides the same functionality as SdComponentDialog, except for the method by which the check box selection is provided.

View sample dialog

The destination directory can be changed by using the Browse button, whereas the available disk space on another drive can be checked by using Disk Space button.

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 SdComponentDialogAdv.

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 SdComponentDialogAdv to get end-user selections.
4. Assign to MEDIA the value saved in step 1.

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 SdComponentDialogAdv 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 SdComponentDialogAdv , 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

Enter the text you want to appear in the title of the dialog box. To display the default title, "Select Components," enter a null string ("").

szMsg

Enter the text you want to appear in the dialog box. To display the default instructions for this dialog box, enter a null string ("").

svDir

Enter the name of a folder you want to appear as the default folder. This parameter contains the chosen folder after the function executes. 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

Enter a null string ("") to display all main components. Specify a component name to display all of that component's sub-components. 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

The selection check boxes have IDs in the range 501 - 599. There are four check boxes built into this dialog box. If the list contains less than four items, the remaining check boxes are hidden. If the list contains more than four items, you must manually modify the dialog to display additional check boxes.

{button ,JI(`LANGREF.HLP>Examples',`SdComponentDialogAdv_example')} Example

{button ,AL(`Displaying icons in component dialogs;ComponentAddItem;ComponentDialog;ComponentGetItemSize;ComponentIsItemSelected;ComponentListItems;ComponentSelectItem;ComponentTotalSize;SdComponentDialog;SdProductName;SdSetupType',0,`',`')} See also

SdComponentDialogAdv example

```
/*--------------------------------------------------------------------------*\
 *
 *  This example illustrates the use of SdComponentDialogAdv, which creates
 *  a dialog box offering component selection and setting/changing of the
 *  destination location.
 *
 *  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. 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 COMPONENT          ""
#define SDCMPDLGTITLE      "Component Selection"
#define SDCMPDLGMSG        "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
   // Get a TARGETDIR target location.
    TARGETDIR = PROGRAMFILES ^ APPBASE_PATH;
    Disable( BACKBUTTON );
    // Let user select components and set target location.
    SdComponentDialogAdv( SDCMPDLGTITLE, SDCMPDLGMSG,
                          TARGETDIR, COMPONENT);
    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 );

    // 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: Is5fn134.rul
```

SdComponentMult

Syntax

SdComponentMult (szTitle, szMsg, szTargetDir, szComponents);

Description

The SdComponentMult function creates a dialog box that provides the end user with an option to select from a list of components and subcomponents in the current media. The dialog box has two child windows. If the selected component has subcomponents, they are displayed in the second window. The dialog also displays the needed disk space (depending on the components and subcomponents being selected) and the free disk space on the target directory to assist in the installation process. The description of a component and/or subcomponent can be viewed by clicking on its name in the Description field.

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 SdComponentMult.

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 SdComponentMult to get end-user selections.
4. Assign to MEDIA the value saved in step 1.

When using SdComponentMult , 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
Enter the text you want to appear in the title of the dialog box. To display the default title, "Select Components," enter a null string ("").

szMsg
Enter the text you want to appear in the dialog box. To display the default instructions for this dialog box, enter a null string ("").

szTargetDir
Enter the name of a target folder—the folder where you want to install the application. Note that the destination folder specified by svTargetDir is not assigned automatically to TARGETDIR or any other system variable. To apply the value of svTargetDir to the setup, you must assign it to TARGETDIR or to a script-defined variable, if one is in use.

szComponents
Enter a null string ("") to display all main components. Specify a component name to display all of that component's sub-components. 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

The Disk Space... button has an ID of 101. This button automatically displays the available disk space dialog. You can remove this button/option if you desire. The Directory static field requires an ID of 851. The list box ID has a multiple selection style.

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',`SdComponentMult_example')} Example

{button ,AL(`Displaying icons in component dialogs;ComponentAddItem;ComponentDialog;ComponentGetItemSize;ComponentIsItemSelected;ComponentListItems;ComponentSelectItem;SdComponentDialog;SdComponentDialog2;SdComponentDialogAdv',0,`',`')} See also

SdComponentMult 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%ld", nResult );
        nDone = AskYesNo( COMPTOTSIZEMSG1 + COMPTOTSIZEMSG2, YES );
    endwhile;

    ListDestroy( listCompList );

endprogram

#include "sddialog.rul"

// Source file: Is5fn021.rul
```

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

Enter the text you want to appear in the title of the dialog box. To display the default title, "Confirm New Folder," enter a null string ("").

szDir

Enter the name of the directory to confirm. You can use the information returned by the SdAskDestPath function.

nReserved

Reserved for future use. Enter 0 (zero) in nReserved.

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',0,`',`')} See also

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
```

SdConfirmRegistration

Syntax

SdConfirmRegistration (szTitle, szName, szCompany, szSerial, nReserved);

Description

The SdConfirmRegistration function creates a dialog box that displays the User Name, Company Name, and Serial Number. If a null string ("") is entered in any field in the dialog box, the displayed field will be empty.

View sample dialog

When using SdConfirmRegistration or other 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
Enter the text you want to appear in the title of the dialog box. To display the default title, "Registration Confirmation," enter a null string ("").

szName
Enter the name returned by either the SdRegisterUserEx or SdRegisterUser functions.

szCompany
Enter the company name returned by either the SdRegisterUserEx or SdRegisterUser functions.

szSerial
Enter the serial number returned by the SdRegisterUserEx function. If is entered, this field is cleared and no entry is displayed.

nReserved
Reserved for future use. Enter 0 (zero) in nReserved.

Return values

YES
Indicates that the Yes button was clicked.

NO
Indicates that the No button was clicked.

{button ,JI(`LANGREF.HLP>Examples',`SdConfirmRegistration_example')} Example

{button ,AL(`SdRegisterUser;SdRegisterUserEx',0,`',`')} See also

SdConfirmRegistration example

```
/*--------------------------------------------------------------------------*\
 *
 * This example illustrates the uses of the SdRegisterUser and
 * SdConfirmRegistration functions.
 *
 * SdRegisterUser is called to prompt the user for his/her name and company
 * name.  These entries are then confirmed when SdConfirmRegistration is
 * called.
 *
\*--------------------------------------------------------------------------*/

   #include "Sddialog.h"

   STRING  szTitle, szMsg, svName, svCompany, szName, szCompany, szSerial;
   NUMBER  nReturn;

program

   Disable(BACKBUTTON);

   nReturn = NO;
   // This loop continues until the user selects Next in the Confirm
   // Registration dialog box (SdConfirmRegistration).
   while (nReturn = NO)

/*--------------------------------------------------------------------------*\
 *
 * Set up variables and get registration info from user.
 *
\*--------------------------------------------------------------------------*/
      szTitle   = "SdRegisterUser Example";
      szMsg     = "Please Register your product now.";
      SdRegisterUser(szTitle, szMsg, svName, svCompany);

/*--------------------------------------------------------------------------*\
 *
 * Set up variables and confirm registration information with user.
 *
\*--------------------------------------------------------------------------*/
      szTitle   = "SdConfirmRegistration Example";
      szName    = svName;
      szCompany = svCompany;
      szSerial  = "";

      nReturn   = SdConfirmRegistration(szTitle, szName, szCompany,
                                        szSerial, 0);
   endwhile;

   nReturn = NO;

   // At this point, the appropriate registration information
   // is available in the szName and szCompany variables.

endprogram

#define SD_SINGLE_DIALOGS      1

#define SD_REGISTERUSER        1
```

```
#define SD_CONFIRMREGISTRATION 1

#include "Sddialog.rul"

// Source file: Is5fn139.rul
```

SdDisplayTopics

Syntax

SdDisplayTopics (szTitle, szMsg, listTopics, listDetails, nReserved);

Descriptions

The SdDisplayTopics function creates a dialog box that displays information based on topic data. The dialog box provides a heading and then topics of titles and descriptions. You can modify the font style of description text to distinguish it from title (topic) text. The message and topic titles are always bold. You can use this dialog box to display Help topics, examples, etc.

View sample dialog

When using SdDisplayTopics , 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
Enter the text you want to appear in the title of the dialog box. To display the default title, "Custom Installation Help," enter a null string ("").

szMsg
Enter the message you want to appear in the Topics dialog box. To display the default instructions for this dialog box, enter a null string ("").

listTopics
Enter the name of a string list that contains the topic choices you want to display.

listDetails
Enter the name of a string list that contains the descriptions of the topics you want to display.

nReserved
Reserved for future use. Enter 0 (zero) in nReserved.

Return values

NEXT
Indicates that the Next button was clicked.

BACK
Indicates that the Back button was clicked.

Comments

- The message static field must have an ID of 801. The topic identifiers must have numbers in the range of 802 - 849. Description fields have an ID range of 851 - 899.
- The spacing of the static description fields is fixed by the size of the dialog. You cannot dynamically change the spacing in the listDetails list. If the number of topics and descriptions is less than the number static fields, nothing will appear in the white space, but the size of the dialog box will not change.

{button ,JI(`LANGREF.HLP>Examples',`SdDisplayTopics_example')} Example

{button ,AL(`SdSetupType',0,`',`')} See also

SdDisplayTopics example

```
/*--------------------------------------------------------------------------*\
 *
 * This example illustrates the usage of the SdDisplayTopics dialog box.
 *
 * SdDisplayTopics is called with listTopics and listDetails,  displaying the
 * topics involved and a detailed description.
 *
\*--------------------------------------------------------------------------*/

   #include "Sddialog.h"

   STRING  szTitle, szMsg, szString;
   LIST    listDetails, listTopics;

program

   // Create list and add strings to it.  These are the topics.
   listTopics = ListCreate (STRINGLIST);
   ListAddString (listTopics, "ExApp3 Program:", AFTER);
   ListAddString (listTopics, "ExApp3 Help:", AFTER);
   ListAddString (listTopics, "ExApp3 Examples:", AFTER);

   // Create list and add strings to it.  These are the descriptions
   // for the topics.
   listDetails = ListCreate (STRINGLIST);

   szString = "Includes all the files to run and write ExApp3.";
   ListAddString (listDetails, szString, AFTER);

   szString = "A computer-based tutorial demonstrating how to create " +
              "programs using ExApp3.";
   ListAddString (listDetails, szString, AFTER);

   szString = "Several examples applications created using ExApp3.";
   ListAddString (listDetails, szString, AFTER);

   szTitle = "SdDisplayTopics Example";
   szMsg   = "Custom setup options allow you to control which parts of " +
             "ExApp3 are to be installed. The options are as follows:";
/*--------------------------------------------------------------------------*\
 *
 * The following displays the topics and details of each topic in a static
 * text field of a dialog box.
 *
\*--------------------------------------------------------------------------*/
   SdDisplayTopics (szTitle, szMsg, listTopics, listDetails, 0);

endprogram

   #define SD_SINGLE_DIALOGS 1
   #define SD_DISPLAYTOPICS  1
   #include "Sddialog.rul"

// Source file: Is5fn140.rul
```

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

Enter the text you want to appear in the title of the dialog box. To display the default title, "Setup Complete," enter a null string ("").

szMsg1

Enter 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, enter a null string ("").

szMsg2

Enter the text to display at the bottom of the dialog box, giving the user information about what to do. To display the default instructions, "Click Finish to complete Setup," enter a null string ("").

szOpt1

Enter the text to display beside the first check box. Enter a null string ("") to hide the check box.

szOpt2

Enter the text to display beside the second check box. Enter a null string ("") 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

- 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.
- 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.
- 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.
- SdFinishReboot, which automatically calls CommitSharedFiles, is preferable to SdFinish.

{button ,JI(`LANGREF.HLP>Examples',`SdFinish_example')}   Example

{button ,AL(`CommitSharedFiles;SdFinishReboot;SdProductName;System;VerUpdateFile',0,`',`')}   See also

SdFinish example

```
/*--------------------------------------------------------------------------*\
 *
 * This example illustrates the uses of the SdProductName, SdWelcome, and
 * SdFinish functions.
 *
 * SdProductName is called to set the %P parameter to the product name.  This
 * allows the product name to replace %P in text fields.
 *
 * SdWelcome is then called to display a welcome message.
 *
 * SdFinish is called to complete the setup process and to give the user final
 * options.
 *
\*--------------------------------------------------------------------------*/

   #include "Sddialog.h"

   STRING   szProductName, szTitle, szMsg, svDir, szComponents;
   STRING   szMsg1, szMsg2, szOpt1, szOpt2;
   BOOL     bvOpt1, bvOpt2;
   NUMBER   nReturn;

program

   Disable(BACKBUTTON);
/*--------------------------------------------------------------------------*\
 *
 * Set the product name for use as %P in text fields.
 *
\*--------------------------------------------------------------------------*/
   szProductName = "My Application";
   SdProductName(szProductName);

/*--------------------------------------------------------------------------*\
 *
 * Display the SdWelcome dialog box.  This box makes use of the %P place
 * holder.  The default message is displayed.
 *
\*--------------------------------------------------------------------------*/
SdWelcomeLabel:
   szTitle = "SdWelcome Example";
   SdWelcome(szTitle, "");
   Enable(BACKBUTTON);

SdSetupTypeLabel:
   szTitle = "SdProductName Example";
   szMsg   = "Choose the type of installation for %P.";
   svDir   = "C:\\EXAMPLE";

   // Another Sd dialog box that makes use of the %P place holder.
   if (SdSetupType(szTitle, szMsg, svDir, 0) = BACK) then

      goto SdWelcomeLabel;
   endif;


/*--------------------------------------------------------------------------*\
 *
 * Display the SdFinish dialog box.  This box makes use of the %P place
 * holder in allowing %P to be passed as parameters in strings.
 *
\*---------------------------------------------------------------------------*/
   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.";
   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
```

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

Enter the text you want to appear in the title of the dialog box. To display the default title, "Setup Complete," enter a null string ("").

szMsg1

Enter 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, enter a null string ("").

nDefOption

Specify a default radio button option selection. The following options are available:

SYS_BOOTWIN

Restart Windows (Windows 3.1 only).

SYS_BOOTMACHINE

Reboot the machine.

0

Do not reboot or restart Windows.

szMsg2

Enter 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

Reserved for future use. Enter 0 (zero) in the parameter nReserved.

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

- 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).
- 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.
- 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',0,`',`')} See also

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
```

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

Enter the text you want to display as the title of the dialog box. To display the default title, "Software License Agreement," enter a null string ("").

szMsg

Enter the message you want to display in the static text field above the multi-line edit field. To display the default instructions, enter a null string ("").

szQuestion

Enter the text you want 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, enter a null string ("").

szLicenseFile

Specify the text file containing your license agreement. Enter the filename of your text file, such as License.txt. Make sure you add the license file 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

- This function cannot return NO, since, if the user selects the No button, the Exit Setup dialog is displayed.
- You can also specify szLicenseFile by entering the fully-qualified name, in quotes, or a Universal Naming Convention (UNC) path.
- 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

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
```

SdOptionsButtons

Syntax

SdOptionsButtons (szTitle, szMsg, listButtons, listDescription);

Description

The SdOptionsButtons function displays a dialog box containing up to four bitmap buttons with accompanying text representing selection options. SdOptionsButtons is ideal for allowing the end user to choose a setup type.

View sample dialog

When using SdOptionsButtons , 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
The text to display as the dialog box title. To display the default title, "Select Components," enter a null string ("").

szMsg
The message you want to be displayed in the dialog box. To display the default instructions for this dialog box to, enter a null string ("").

listButtons
The list of formatted strings specifying the bitmaps to be displayed on the buttons. Refer to the Comments section, below.

listDescription
The list strings forming the descriptions of the corresponding bitmaps in the listButtons list.

Return values

BACK
Indicates that the Back button was clicked.

101
The first (top) option button was clicked.

102
The second option button was clicked.

103
The third option button was clicked.

104
The fourth option button was clicked.

Comments

- Because the user must click on the optional buttons provided to continue, call the Disable function to disable the Next button whenever you use SdOptionsButtons.
- The bitmap buttons displayed in the SdOptionsButtons dialog box are created by passing a string list of

bitmap IDs as the parameter listButtons to the SdOptionsButtons function. Each member of the string list is a specially formatted string of the form:

```
  "@<bitmap ID>;<bitmap icon flag>;<mask color>"
```

The @ symbol begins every bitmap ID string. The <bitmap ID> is the ID number of a bitmap resource you have placed into _isres.dll. Microsoft Visual C++ 5.0 allows you to modify bitmap resources in _isres.dll. Bitmap resource numbers must be greater than zero. When giving your bitmap resources ID numbers, do not use a bitmap ID of 1200 because it is reserved for use by InstallShield. InstallShield provides four bitmap resources, 12001-12004, which are used in the example below. The <bitmap icon flag> is either 1 (TRUE) or 0 (FALSE), indicating whether or not the color specified in the <mask color> field will be transparent. RGB 255, 0, 255 (purple) commonly is used as a mask color.

The optimal size for a bitmap displayed in an option button is 28 x 28 pixels.

{button ,JI(`LANGREF.HLP>Examples',`SdOptionsButtons_example')}   Example

{button ,AL(`ListCreate;ListAddString;ListDestroy;Enable;Disable',0,`',`')}See also

SdOptionsButtons example

```
/*--------------------------------------------------------------------------*\
 *
 * This example illustrates the uses of the SdOptionsButtons dialog box.
 *
 * Two lists are created for the button icons and descriptions.  These lists
 * are used in SdOptionsButtons to prompt the user for a setup type.  The
 * result is returned from SdOptionsButtons, and a message is displayed.
 *
\*--------------------------------------------------------------------------*/

   #include "Sddialog.h";

   STRING  szTitle, szMsg;
   LIST    listButtons, listDescrip;
   NUMBER  nReturn;

program

   Disable (BACKBUTTON);
   Disable (NEXTBUTTON);

   // Create the list of buttons and their descriptions.
   listButtons = ListCreate (STRINGLIST);
   listDescrip = ListCreate (STRINGLIST);

   // Set button list.
   // Add the bitmap buttons to the listButtons list.
   ListAddString (listButtons, "@12001;1;255,0,255", AFTER);
   ListAddString (listButtons, "@12002;1;255,0,255", AFTER);
   ListAddString (listButtons, "@12003;1;255,0,255", AFTER);
   ListAddString (listButtons, "@12004;1;255,0,255", AFTER);

   // Set description list.
   ListAddString (listDescrip, "Typical\n" +
                               "Recommended for most computers.", AFTER);
   ListAddString (listDescrip, "Portable\n" +
                               "The application will be set up with "  +
                               "options that are useful for portable " +
                               "computers.", AFTER);
   ListAddString (listDescrip, "Compact\n" +
                               "To save disk space, none of the " +
                               "optional components will be " +
                               "installed.", AFTER);
   ListAddString (listDescrip, "Custom\n" +
                               "For advanced users and system " +
                               "administrators only.  You can " +
                               "customize all available Setup " +
                               "options.", AFTER);

/*--------------------------------------------------------------------------*\
 *
 * The following displays the four icons from the list above, prompting the
 * user for a setup type.
 *
\*--------------------------------------------------------------------------*/
   nReturn = SdOptionsButtons (szTitle, szMsg, listButtons, listDescrip);

   // Display a message dependent upon the icon chosen.
   switch(nReturn)

      case 101:

         MessageBox("Typical installation selected.", INFORMATION);
      case 102:

         MessageBox("Portable installation selected.", INFORMATION);
      case 103:

         MessageBox("Compact installation selected.", INFORMATION);
      case 104:

         MessageBox("Custom installation selected.", INFORMATION);
      default:

         MessageBox("SdOptionsButtons:\n\n An error occurred.", SEVERE);
   endswitch;

   Enable(NEXTBUTTON);

   // Destroy the lists.
   ListDestroy (listButtons);
   ListDestroy (listDescrip);

endprogram

#include "Sddialog.rul"

// Source file: Is5fn145.rul
```

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
Enter the name of your product. 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',0,`',`')} See also

SdProductName example

```
/*--------------------------------------------------------------------------*\
 *
 * This example illustrates the uses of the SdProductName, SdWelcome, and
 * SdFinish functions.
 *
 * SdProductName is called to set the %P parameter to the product name.  This
 * allows the product name to replace %P in text fields.
 *
 * SdWelcome is then called to display a welcome message.
 *
 * SdFinish is called to complete the setup process and to give the user final
 * options.
 *
\*--------------------------------------------------------------------------*/

   #include "Sddialog.h"

   STRING   szProductName, szTitle, szMsg, svDir, szComponents;
   STRING   szMsg1, szMsg2, szOpt1, szOpt2;
   BOOL     bvOpt1, bvOpt2;
   NUMBER   nReturn;

program

   Disable(BACKBUTTON);
/*--------------------------------------------------------------------------*\
 *
 * Set the product name for use as %P in text fields.
 *
\*--------------------------------------------------------------------------*/
   szProductName = "My Application";
   SdProductName(szProductName);

/*--------------------------------------------------------------------------*\
 *
 * Display the SdWelcome dialog box.  This box makes use of the %P place
 * holder.  The default message is displayed.
 *
\*--------------------------------------------------------------------------*/
SdWelcomeLabel:
   szTitle = "SdWelcome Example";
   SdWelcome(szTitle, "");
   Enable(BACKBUTTON);

SdSetupTypeLabel:
   szTitle = "SdProductName Example";
   szMsg   = "Choose the type of installation for %P.";
   svDir   = "C:\\EXAMPLE";

   // Another Sd dialog box that makes use of the %P place holder.
   if (SdSetupType(szTitle, szMsg, svDir, 0) = BACK) then

      goto SdWelcomeLabel;
   endif;


/*--------------------------------------------------------------------------*\
 *
 * Display the SdFinish dialog box.  This box makes use of the %P place
 * holder in allowing %P to be passed as parameters in strings.
 *
\*-------------------------------------------------------------------------*/
   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.";
   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
```

SdRegisterUser

Syntax

SdRegisterUser (szTitle, szMsg, svName, svCompany);

Description

The SdRegisterUser function creates a dialog box that retrieves the user name and company name. InstallShield will fill in the name and company fields with the default values found in the registry (32-bit Windows) or User.dll (16-bit Windows).

The Next button becomes enabled only when data exists in both edit fields. If InstallShield can locate default name and company values from the system, the Next button is automatically enabled.

View sample dialog

When using SdRegisterUser 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
Enter the text you want to appear in the title of the dialog box. To display the default title, "User Information," enter a null string ("").

szMsg
Enter the text you want to appear in the dialog box. This text is considered a static control. To display the default instructions for this dialog box, enter a null string ("").

svName
Returns the name entered by the user. You can use this information and write it to another file or display it in a confirmation dialog if you desire.

svCompany
Returns the company name entered by the user. You can use this information and write it to another file or display it in a confirmation dialog if you desire.

Return values

NEXT
Indicates that the Next button was clicked.

BACK
Indicates that the Back button was clicked.

{button ,JI(`LANGREF.HLP>Examples',`SdRegisterUser_example')} Example

{button ,AL(`SdConfirmRegistration;SdRegisterUserEx',0,`',`')} See also

SdRegisterUser example

```
/*--------------------------------------------------------------------------*\
 *
 * This example illustrates the uses of the SdRegisterUser and
 * SdConfirmRegistration functions.
 *
 * SdRegisterUser is called to prompt the user for his/her name and company
 * name.  These entries are then confirmed when SdConfirmRegistration is
 * called.
 *
\*--------------------------------------------------------------------------*/

   #include "Sddialog.h"

   STRING  szTitle, szMsg, svName, svCompany, szName, szCompany, szSerial;
   NUMBER  nReturn;

program

   Disable(BACKBUTTON);

   nReturn = NO;
   // This loop continues until the user selects Next in the Confirm
   // Registration dialog box (SdConfirmRegistration).
   while (nReturn = NO)

/*--------------------------------------------------------------------------*\
 *
 * Set up variables and get registration info from user.
 *
\*--------------------------------------------------------------------------*/
      szTitle   = "SdRegisterUser Example";
      szMsg     = "Please Register your product now.";
      SdRegisterUser(szTitle, szMsg, svName, svCompany);

/*--------------------------------------------------------------------------*\
 *
 * Set up variables and confirm registration information with user.
 *
\*--------------------------------------------------------------------------*/
      szTitle   = "SdConfirmRegistration Example";
      szName    = svName;
      szCompany = svCompany;
      szSerial  = "";

      nReturn   = SdConfirmRegistration(szTitle, szName, szCompany,
                                        szSerial, 0);
   endwhile;

   nReturn = NO;

   // At this point, the appropriate registration information
   // is available in the szName and szCompany variables.

endprogram

#define SD_SINGLE_DIALOGS      1

#define SD_REGISTERUSER        1
```

```
#define SD_CONFIRMREGISTRATION 1

#include "Sddialog.rul"

// Source file: Is5fn139.rul
```

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. InstallShield will fill in the name and company fields with the default values found in the registry (32-bit Windows) or User.dll (16-bit Windows).

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
Enter the text you want to appear in the title of the dialog box. To display the default title, "User Information," enter a null string ("").

szMsg
Enter the text you want to appear in the dialog box. This text is considered a static control. To display the default instructions for this dialog box, enter a null string ("").

svName
Returns the name entered by the user. You can use this information and write it to a file or display it in a confirmation dialog if you desire.

svCompany
Returns the company name entered by the user. You can use this information and write it to a file or display it in a confirmation dialog if you desire.

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 if you desire.

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',0,`',`')} See also

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
```

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
Enter the text you want to appear in the title of the dialog box. To display the default title, "Select Program Folder," enter a null string ("").

szMsg
Enter the text you want to appear in the dialog box. This text is considered a static control. To display the default instructions for this dialog box, enter a null string ("").

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',0,`',`')} See also

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
```

SdShowAnyDialog

Syntax

SdShowAnyDialog (szTitle, szID, nID, nReserved);

Description

The SdShowAnyDialog function displays a custom or modified dialog box. This function is recommended for advanced users only.

View sample dialog

When using SdShowAnyDialog 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
Enter the text you want to appear in the title of the dialog box. To display the default title, "Welcome," enter a null string ("").

szID
Enter the string identifier that identifies the dialog. If you enter a null string ("") in this parameter, the function uses the value in nID.

nID
Enter the numeric value that identifies the dialog. If you entered a value in szID, this parameter is ignored.

nReserved
Reserved for future use. Enter 0 (zero) in nReserved.

Return values

NEXT
Indicates that the Next button was clicked.

BACK
Indicates that the Back button was clicked.

Comments

- In order to use the SdShowAnyDialog function, you must know the ID of either the modified dialog box in _isres.dll or the custom dialog box in _isuser.dll that you wish to display.
- If the dialog has only static controls, you do not need to modify the SdShowAnyDialog script file. But if your dialog has any other controls, you must modify the Sdsadlg.rul file, located in the Include folder of your InstallShield program files folder, in order to process the feedback from the user.

{button ,JI(`LANGREF.HLP>Examples',`SdShowAnyDialog_example')} Example

{button ,AL(`SdShowDlgEdit1;SdShowDlgEdit2;SdShowDlgEdit3;SdShowInfoList;SdShowMsg',0,`',`')} See also

SdShowAnyDialog example

```
/*--------------------------------------------------------------------------*\
 *
 * This example illustrates the usage of the SdShowAnyDialog dialog box.
 *
 * SdShowAnyDialog is called twice.  The first time it is called with the ID
 * for the Welcome dialog box, and the second time with the ID for the Finish
 * dialog box.
 *
\*--------------------------------------------------------------------------*/

   #include "Sddialog.h"

   STRING   szTitle, szId;
   NUMBER   nId;

program

   Disable(BACKBUTTON);

   szTitle = "SdShowAnyDialog Example";
   szId    = "";
   nId     = SD_NDLG_WELCOME;
/*--------------------------------------------------------------------------*\
 *
 * The following calls the SdWelcome dialog box to be displayed.
 *
\*--------------------------------------------------------------------------*/
   SdShowAnyDialog (szTitle, szId, nId, 0);

   nId     = SD_NDLG_FINISH;
/*--------------------------------------------------------------------------*\
 *
 * The following calls the SdFinish dialog box to be displayed.
 *
\*--------------------------------------------------------------------------*/
   SdShowAnyDialog (szTitle, szId, nId, 0);

endprogram

   #define SD_SINGLE_DIALOGS 1
   #define SD_SHOWANYDIALOG  1

   #include "Sddialog.rul"

// Source file: Is5fn151.rul
```

SdShowDlgEdit1

Syntax

SdShowDlgEdit1 (szTitle, szMsg, szField1, svEdit1);

Description

The SdShowDlgEdit1 function creates a general dialog box that displays a message and one single-line edit field. You can specify a title for the dialog box.

View sample dialog

When using SdShowDlgEdit1 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
Enter the text you want to appear in the title of the dialog box. To display the default title, "Edit Data," enter a null string ("").

szMsg
Enter the message you want to appear in the dialog box. You can use the %P place holder in your message string. %P is replaced by the product name made available by calling the SdProductName function.

szField1
Enter a name (up to 11 characters) to be displayed to the left of the svEdit1 edit field. Enter a null string ("") to display "Field 1:" to the left of the edit field

svEdit1
This variable both initializes and saves the contents of the edit field.

Return values

NEXT
Indicates that the Next button was clicked.

BACK
Indicates that the Back button was clicked.

{button ,JI(`LANGREF.HLP>Examples',`SdShowDlgEdit1_example')} Example

{button ,AL(`SdShowAnyDialog;SdShowDlgEdit2;SdShowDlgEdit3;SdShowInfoList;SdShowMsg',0,`',`')} See also

SdShowDlgEdit1 example

```
/*--------------------------------------------------------------------------*\
 *
 * This example illustrates the usage of the SdShowDlgEdit1 dialog box.
 *
 * SdShowDlgEdit1 is called offering a single edit field.  When the dialog is
 * returned, svEdit1 is set to the string within the edit field.
 *
\*--------------------------------------------------------------------------*/

   #include "Sddialog.h"

   STRING szTitle, szMsg, szField1, svEdit1;

program

   szTitle  = "SdShowDlgEdit1 Example";
   szMsg    = "Please choose a directory to install YourApp into:";
   szField1 = "Target:";
   svEdit1   = "C:\\EXAMPLE\\TARGET\\YOURAPP";

/*--------------------------------------------------------------------------*\
 *
 * The following displays a message and an edit line field.
 *
\*--------------------------------------------------------------------------*/
   if (SdShowDlgEdit1(szTitle, szMsg, szField1, svEdit1) < 0) then

      MessageBox("SdShowDlgEdit1 failed.", SEVERE);
   endif;

   // Display svEdit1 string variable.
   SprintfBox(INFORMATION, szTitle, "svEdit1: %s", svEdit1);


endprogram

   #define  SD_SINGLE_DIALOGS 1
   #define  SD_SHOWDLGEDIT1   1
   #include "Sddialog.rul"

// Source file: Is5fn152.rul
```

SdShowDlgEdit2

Syntax

SdShowDlgEdit2 (szTitle, szMsg, szField1, szField2, svEdit1, svEdit2);

Description

The SdShowDlgEdit2 function creates a general dialog box that displays a message and two single-line edit fields. You can specify a title for the dialog box.

View sample dialog

When using SdShowDlgEdit2 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
Enter the text you want to appear in the title of the dialog box. If you enter a null string (""), the default title ("Edit Data") is displayed.

szMsg
Enter the message you want to appear in the dialog box. You can use the %P place holder in your message string. %P is replaced by the product name made available by calling the SdProductName function.

szField1
Enter a name (up to 11 characters) to be displayed to the left of the svEdit1 edit field.

szField2
Enter a name (up to 11 characters) to be displayed to the left of the svEdit2 edit field.

svEdit1
This variable both initializes and saves the contents of the first edit field.

svEdit2
This variable both initializes and saves the contents of the second edit field.

Return values

NEXT
Indicates that the Next button was clicked.

BACK
Indicates that the Back button was clicked.

{button ,JI(`LANGREF.HLP>Examples',`SdShowDlgEdit2_example')} Example

{button ,AL(`SdShowAnyDialog;SdShowDlgEdit1;SdShowDlgEdit3;SdShowInfoList;SdShowMsg',0,`',`')} See also

SdShowDlgEdit2 example

```
/*--------------------------------------------------------------------------*\
 *
 * This example illustrates the usage of the SdShowDlgEdit2 dialog box.
 *
 * SdShowDlgEdit2 is called offering two edit fields.  When the dialog is
 * returned, svEdit1 and svEdit2 are set to the string within the edit
 * fields.
 *
\*--------------------------------------------------------------------------*/

   #include "Sddialog.h"

   STRING szTitle, szMsg, szField1, szField2, svEdit1, svEdit2;

program

   szTitle  = "SdShowDlgEdit2 Example";
   szMsg    = "All files within the Source directory will be copied into " +
              "the Target directory.";
   szField1 = "Source:";
   szField2 = "Target:";
   svEdit1  = "C:\\EXAMPLE\\SOURCE";
   svEdit2  = "C:\\EXAMPLE\\TARGET";

/*--------------------------------------------------------------------------*\
 *
 * The following displays a message and two edit line fields.
 *
\*--------------------------------------------------------------------------*/
   if (SdShowDlgEdit2(szTitle, szMsg, szField1, szField2, svEdit1,
                      svEdit2) < 0) then

      MessageBox("SdShowDlgEdit2 failed.", SEVERE);
   endif;

   // Display svEdit string variables.
   SprintfBox(INFORMATION, szTitle, "svEdit1: %s\n\nsvEdit2: %s", svEdit1,
              svEdit2);


endprogram

   #define  SD_SINGLE_DIALOGS 1
   #define  SD_SHOWDLGEDIT2   1
   #include "Sddialog.rul"

// Source file: Is5fn153.rul
```

SdShowDlgEdit3

Syntax

SdShowDlgEdit3 (szTitle, szMsg, szField1, szField2, szField3, svEdit1, svEdit2, svEdit3);

Description

The SdShowDlgEdit3 function creates a general dialog box that displays a message and three single-line edit fields. You can specify a title for the dialog box in szTitle.

View sample dialog

When using SdShowDlgEdit3 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
Enter the text you want to appear in the title of the dialog box. If you enter a null string (""), the default title ("Edit Data") is displayed.

szMsg
Enter the message you want to appear in the dialog box. You can use the %P place holder in your message string to insert the product name (if any) that has been specified by a call to SdProductName function.

szField1
Enter a name (up to 11 characters) to be displayed to the left of the svEdit1 edit field.

szField2
Enter a name (up to 11 characters) to be displayed to the left of the svEdit2 edit field.

szField3
Enter a name (up to 11 characters) to be displayed to the left of the svEdit3 edit field.

svEdit1
This variable both initializes and saves the contents of the first edit field.

svEdit2
This variable both initializes and saves the contents of the second edit field.

svEdit3
This variable both initializes and saves the contents of the third edit field.

Return values

NEXT
Indicates that the Next button was clicked.

BACK
Indicates that the Back button was clicked.

{button ,JI(`LANGREF.HLP>Examples',`SdShowDlgEdit3_example')} Example

{button ,AL(`SdShowAnyDialog;SdShowDlgEdit1;SdShowDlgEdit2;SdShowInfoList;SdShowMsg',0,`',`')} See also

SdShowDlgEdit3 example

```
/*--------------------------------------------------------------------------*\
 *
 * This example illustrates the usage of the SdShowDlgEdit3 dialog box.
 *
 * The SdShowDlgEdit3 function is called providing three edit fields.  Strings
 * are passed to variables when SdShowDlgEdit3 is continued.
 *
\*--------------------------------------------------------------------------*/

   #include "Sddialog.h"

   STRING szTitle, szMsg, szField1, szField2, szField3, svEdit1, svEdit2;
   STRING svEdit3;

program

   szTitle   = "SdShowDlgEdit3 Example";
   szMsg     = "Please choose up to two valid directories in Backup: or " +
               "Alternate: to back up the source directory.\n\n";
   szField1  = "Directory";
   szField2  = "Backup";
   szField3  = "Alternate";
   svEdit1   = "C:\\YOURAPP";
   svEdit2   = "C:\\BACKUP1";
   svEdit3   = "C:\\BACKUP2";

/*--------------------------------------------------------------------------*\
 *
 * The following displays the three edit fields.  Whatever strings that are
 * left in the edit fields are set to the svEdit variables.
 *
\*--------------------------------------------------------------------------*/
   if (SdShowDlgEdit3 (szTitle, szMsg, szField1, szField2, szField3, svEdit1,
                       svEdit2, svEdit3) < 0) then

      MessageBox("SdShowDlgEdit3 failed.", SEVERE);
   endif;

   szMsg = "svEdit1: %s\n\nsvEdit2: %s\n\nsvEdit3: %s";
   SprintfBox(INFORMATION, szTitle, szMsg, svEdit1, svEdit2, svEdit3);


endprogram

   #define  SD_SINGLE_DIALOGS 1
   #define  SD_SHOWDLGEDIT3   1

   #include "Sddialog.rul"

// Source file: Is5fn154.rul
```

SdShowFileMods

Syntax

SdShowFileMods (szTitle, szMsg, szTargetFile, szAltFile, listChanges, nvSelection);

Description

The SdShowFileMods function creates a dialog box that displays changes you want to make to a file. These choices are available:

1. Make changes to the target file.
2. Make changes to the alternate file, which is a copy of the target file, but incorporates changes.
3. Do not make any changes. SdShowFileMods does not make changes to a file. You must write those changes into your script using the appropriate file functions.

View sample dialog

When using SdShowFileMods 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

Enter the text you want to appear in the title of the dialog box. To display the default title, "Modifying File," enter a null string ("").

szMsg

Enter the message you want to appear in the dialog box. To display the default instructions for this dialog box, enter a null string ("").

szTargetFile

Enter the name of the file you want to modify. This will be displayed with the first radio button.

szAltFile

Enter an alternate name you may want to give the file if you decide to make the changes. This will be displayed with the second radio button. To use the name of file specified in szTargetFile with the extension .bak, enter a null string ("").

listChanges

Enter the name of a string list that contains the list of changes you want to make to the file. This list is placed in a multi-line edit field that allows you to select the changes to be implemented.

nvSelection

Returns the ID value of the selection:

101

"Let Setup modify the <szTargetFile> file."

102

"Save the required changes to <szAltFile> file."

103

"Do not make any changes."

Return values

NEXT
Indicates that the Next button was clicked.

BACK
Indicates that the Back button was clicked.

{button ,JI(`LANGREF.HLP>Examples',`SdShowFileMods_example')} Example

{button ,AL(`ListCreate;ListAddString',0,`',`')} See also

SdShowFileMods example

```
/*--------------------------------------------------------------------------*\
 *
 * This example illustrates the usage of the SdShowFileMods function.
 *
\*--------------------------------------------------------------------------*/

   #include "Sddialog.h"

   STRING   szTitle, szMsg, szTargetFile, szAltFile;
   NUMBER   nvSelection;
   LIST     listID;

program

   // Set variables for SdShowFileMods.
   szTitle = "SdShowFileMods Example";
   szMsg = "Choose what option to take";
   szTargetFile = "EXAMPLE.TXT";
   szAltFile = "EXAMPLE.NEW";
   listID = ListCreate(STRINGLIST);
   ListAddString(listID, "PATH = C:\\EXAMPLE", AFTER);
   ListAddString(listID, "FILES = 40", AFTER);

/*--------------------------------------------------------------------------*\
 *
 * The following prompts the user for what method to take to make alterations
 * to a file.
 *
\*--------------------------------------------------------------------------*/
   SdShowFileMods(szTitle, szMsg, szTargetFile, szAltFile,
                  listID, nvSelection);

   switch(nvSelection)

      case 101:
         SprintfBox(INFORMATION, szTitle, "Setup modified the %s file.",
                    szTargetFile);

      case 102:
         SprintfBox(INFORMATION, szTitle, "The required changes were saved " +
                    "to the %s file.", szAltFile);

      case 103:
         SprintfBox(INFORMATION, szTitle, "No changes were made.");

   endswitch;


endprogram

   #define SD_SINGLE_DIALOGS 1
   #define SD_SHOWFILEMODS 1
   #include "Sddialog.rul"

// Source file: Is5fn155.rul
```

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
Enter the text you want to appear in the title of the dialog box. To display the default title, "Information," enter a null string ("").

szMsg
Enter the message you want to appear above the information box. If you enter null string (""), the word "Text" will appear above the list.

listID
Enter the name of the list whose contents you want to appear 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',0,`',`')} See also

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
```

SdShowMsg

Syntax

SdShowMsg (szMsg, bShow);

Description

The SdShowMsg function creates a general dialog box that displays a message or other information in a small window. This window automatically sizes according to the amount of text in the message. In addition, you can display and hide the text in the window at will.

View sample dialog

When using SdShowMsg 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

szMsg

Enter the text you want to appear in the dialog box. To display the default message, "Setup is searching for installed components," enter a null string ("").

bShow

Use this parameter to specify whether you want the message to be displayed. These constants are available:

TRUE

Shows the message.

FALSE

Hides the message.

Return values

No return value given.

Comments

SdShowMsg supports only a single line of text even when using "\n" in the string.

{button ,JI(`LANGREF.HLP>Examples',`SdShowMsg_example')} Example

{button ,AL(`SdShowAnyDialog;SdShowDlgEdit1;SdShowDlgEdit2;SdShowDlgEdit3;SdShowInfoList',0,`',`')} See also

SdShowMsg example

```
/*--------------------------------------------------------------------------*\
 *
 * This example illustrates the uses of the SdShowMsg dialog box.
 *
 * SdShowMsg is called to display a message for three seconds.  It is then
 * called again to remove the message from the screen.  The function is then
 * called again to display another message for another three seconds.
 *
\*--------------------------------------------------------------------------*/

   #include "Sddialog.h"

   STRING szMsg;

program

   // Display a message on the screen.
   szMsg = "This message will appear for three seconds.";
   SdShowMsg(szMsg, TRUE);

   // Delay for 3 seconds.
   Delay(3);

   // Remove message from screen.
   SdShowMsg(szMsg, FALSE);

   szMsg = "This is another message which will appear for a mere three seconds.";
   SdShowMsg(szMsg, TRUE);
   Delay(3);

endprogram

#define SD_SINGLE_DIALOGS 1
#define SD_SHOWMSG        1

#include "Sddialog.rul"

// Source file:Is5fn157.rul
```

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

Enter the title you want to appear in the caption bar of the dialog box. To display the default title, "Start Copying Files," enter a null string ("").

szMsg

Enter the message you want to appear in the static text field above the multi-line edit field. To display the default instructions for this dialog box, enter a null string ("").

listData

Place the string list in which you have collected the information retrieved from the user in the parameter listData. SdStartCopy will automatically place the strings separately into the multi-line edit field.

Return values

BACK

Indicates that the user selected the Back button.

NEXT

Indicates that the user selected the Next button.

Comments

If the list passed as the parameter listData is empty, the multi-line edit field will be hidden and only the static text field will be visible.

{button ,JI(`LANGREF.HLP>Examples',`SdStartCopy_example')} Example

{button ,AL(`SelectDir;SelectFolder;SetupType;SdSetupType;SdSelectFolder;SdRegisterUser;SdRegisterUserEx;SdAskDestPath',0,`',`')} See also

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
```

SdWelcome

Syntax

SdWelcome (szTitle, szMsg);

Description

The SdWelcome function creates and displays a general welcome dialog box.

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

Enter the text you want to appear in the title of the dialog box. You can use the %P place holder in your message string to insert the product name (if any) that has been specified by a call to SdProductName function. To display the default title, "Welcome," enter a null string ("").

szMsg

Enter the message you want to appear in the Welcome dialog box. To display the default welcome message, enter a null string ("").

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

SdWelcome example

```
/*--------------------------------------------------------------------------*\
 *
 * This example illustrates the uses of the SdProductName, SdWelcome, and
 * SdFinish functions.
 *
 * SdProductName is called to set the %P parameter to the product name.  This
 * allows the product name to replace %P in text fields.
 *
 * SdWelcome is then called to display a welcome message.
 *
 * SdFinish is called to complete the setup process and to give the user final
 * options.
 *
\*--------------------------------------------------------------------------*/

   #include "Sddialog.h"

   STRING   szProductName, szTitle, szMsg, svDir, szComponents;
   STRING   szMsg1, szMsg2, szOpt1, szOpt2;
   BOOL     bvOpt1, bvOpt2;
   NUMBER   nReturn;

program

   Disable(BACKBUTTON);
/*--------------------------------------------------------------------------*\
 *
 * Set the product name for use as %P in text fields.
 *
\*--------------------------------------------------------------------------*/
   szProductName = "My Application";
   SdProductName(szProductName);

/*--------------------------------------------------------------------------*\
 *
 * Display the SdWelcome dialog box.  This box makes use of the %P place
 * holder.  The default message is displayed.
 *
\*--------------------------------------------------------------------------*/
SdWelcomeLabel:
   szTitle = "SdWelcome Example";
   SdWelcome(szTitle, "");
   Enable(BACKBUTTON);

SdSetupTypeLabel:
   szTitle = "SdProductName Example";
   szMsg   = "Choose the type of installation for %P.";
   svDir   = "C:\\EXAMPLE";

   // Another Sd dialog box that makes use of the %P place holder.
   if (SdSetupType(szTitle, szMsg, svDir, 0) = BACK) then

      goto SdWelcomeLabel;
   endif;


/*--------------------------------------------------------------------------*\
 *
 * Display the SdFinish dialog box.  This box makes use of the %P place
 * holder in allowing %P to be passed as parameters in strings.
 *
\*---------------------------------------------------------------------------*/
   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.";
   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
```

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.

CommitSharedFiles

Syntax

CommitSharedFiles (lReserved);

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

lReserved
Enter 0 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

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
```

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.

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.

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

Enter the name of the folder to which you are adding the icon. If you are calling AddFolderIcon to add a shortcut icon to the Windows 95 Start Programs menu, use 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

Enter the name of the icon that you are adding to the folder or to the Windows 95 Start Programs menu. The name appears under 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 (below).

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. However, be aware that the Explorer Shell does not allow the following characters in item names: /, \, :, ?, <, >, or |.

szCommandLine

Enter one of the following:

- The fully-qualified name of the executable associated with the icon you are adding, including any command line parameters. Refer to the Comments section below for more information. If you are calling AddFolderIcon to add a shortcut icon to the Windows 95 Start Programs menu, enter the fully-qualified path of the links directory. The links directory is where your application stores its icon link files.
- The fully-qualified path if szItemName is a subfolder (Windows 95 and Windows NT 4.0 only).

szWorkingDir

Enter the directory where your application's program files are located. (Not applicable if szItemName is a subfolder.) If you enter a null string (""), the directory that contains the program file becomes the working directory. Refer to the Comments section below for more information.

szIconPath

Enter the filename for the icon you want to display. (Not applicable if szItemName is a subfolder.) Refer to the Comments section below for more information.

nIcon

Enter 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

Enter the shortcut key (in the form of a string) that allows you to quickly start your application. 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

Use this parameter to specify icon appearance. You can combine constants with the bitwise OR operator (|). The following constants are available:

REPLACE

Indicates that the current icon or item in the folder is replaced.

RUN_MAXIMIZED

Indicates that the program should be maximized when launched. This constant should not be specified in a setup that is running under Windows 3.1 or Windows NT 3.51.

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

- 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 quotes. 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 quotes
LongPathToQuote (szProgram, TRUE);

// Neccessary 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 quotes.

{button ,JI(`LANGREF.HLP>Examples',`AddFolderIcon_example')} Examples

{button ,AL(`CreateProgramFolder;DeleteProgramFolder;DeleteFolderIcon;ProgDefGroupType;SelectFolder',0,`',`')} See also

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.

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
```

AddFolderIcon example

```
/*--------------------------------------------------------------------------*\
 *
 * AddFolderIcon Example 2
 *
 * This example creates a cascading submenu on the StartUp menu and adds an
 * icon to it.
 *
 * 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.
 *
 *       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 change 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;
   STRING  szIconPath, szShortCutKey, szProgram, szParam;
   NUMBER  nIcon, nFlag, nResult;

program

/*--------------------------------------------------------------------------*\
 *
 * AddFolderIcon will automatically create a menu called "SubMenu Example" on
 * the StartUp menu.
 *
\*--------------------------------------------------------------------------*/
   szProgramFolder = FOLDER_STARTUP ^ "SubMenu Example";

/*--------------------------------------------------------------------------*\
 *
 * Place the "Notepad Example" shortcut in the "SubMenu Example" folder and
 * display it as an icon in the "SubMenu Example" cascading menu.  When this
 * shortcut is run, the executable will display maximized.
 *
\*--------------------------------------------------------------------------*/
   szItemName = "Notepad Example1";
   szProgram  = PROGRAM;
   szParam    = PARAM;

   LongPathToQuote (szProgram, TRUE);
   LongPathToShortPath (szParam);
   szCommandLine = szProgram + " " + szParam;

   szWorkingDir  = "";
   szIconPath    = "";
   nIcon         = 0;
   szShortCutKey = "";
   nFlag         = REPLACE|RUN_MAXIMIZED;

   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: Is5fn588.rul
```

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
```

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
Enter the name of the folder you want 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',0,`',`')}See also

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
```

DeleteFolderIcon

Syntax

DeleteFolderIcon (szProgramFolder, szItemName);

Description

The DeleteFolderIcon function removes a program icon from a folder.

Parameters

szProgramFolder
Enter the name of the folder that contains the icon you want to remove.

szItemName
Enter the name of the icon you want 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

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
```

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
Enter the name of the folder you want 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

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
```

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:

- You cannot use this function to exit Windows. To exit Windows you must call the System function.
- InstallShield's program group and program item functions work only with the Windows 3.1 Program Manager, or with 100% compatible shells.

Parameters

bSaveGroup

Specify whether you want the Program Manager to save its group information when the program exits. Use one of these two constants:

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

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
```

GetFolderNameList

Syntax

GetFolderNameList (szFolderName, listItemsID, listSubFoldersID);

Description

The GetFolderNameList function retrieves a list of program item shortcuts and a list of subfolder names from a specified folder. This function is recommended for use with Windows NT 4.0 and Windows 95. Under Windows 3.1, GetFolderNameList will return listItemsID but will ignore listSubFoldersID. (See Comments, below.) When retrieving item names and subfolder names under Windows NT, call ProgDefGroupType first to establish the group as either COMMON or PERSONAL. By default, InstallShield searches under COMMON.

Parameters

szFolderName

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 directory under the Windir environment variable.
- Windows NT 4.0 (if COMMON is selected with ProgDefGroupType): The ..\profiles\All Users\Start Menu\Programs directory under the Windir environment variable.
- 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.
- Windows 3.1: No default path applicable.

Under Windows 95 and Windows NT 4.0 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 items in szFolderName.

listSubFoldersID

Returns a list with the names of the subfolders in szFolderName.

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

- GetFolderNameList will fail under Windows 3.1 if szFolderName is a cascading menu. It is recommended that you use GetGroupNameList and GetItemNameList to retrieve information about program folders and program items under these platforms.
- The location of the Start menu is different under different languages. Nonetheless, InstallShield automatically selects the correct path.

{button ,JI(`LANGREF.HLP>Examples',`GetFolderNameList_example')} Example

{button ,AL(`AddFolderIcon;GetItemNameList;GetGroupNameList;SdShowInfoList',0,`',`')} See also

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
```

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

The name of a valid string list. InstallShield places the retrieved information into the list; that is, each folder or group 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

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
```

GetItemNameList

Syntax

GetItemNameList (szGroup, listID);

Description

The GetItemNameList retrieves all the program items names from a specified program group under the Program Manager shell. If you are using InstallShield to retrieve item names under the Explorer shell, this function will return an error. Instead, call the GetFolderNameList function.

Parameters

szGroup

The name of the group for which you want 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

The name of a valid string list. InstallShield uses this list to return the name of each program icon in the folder defined in szGroup.

Return values

0

GetItemNameList successfully retrieved all the program items.

< 0

GetItemNameList was unable to retrieve the program items.

{button ,JI(`LANGREF.HLP>Examples',`GetItemNameList_example')} Example

{button ,AL(`GetFolderNameList;GetGroupNameList;ListCreate;ListDestroy;ListGetFirstString;ListGetNextString',0,`' ,`')} See also

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
```

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.

Parameters

nType

Enter a constant to specify a program group type. The following constants are available:

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

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
```

QueryProgGroup

Syntax

QueryProgGroup (szGroupName, svGroupPath, nvItemCount);

Description

The QueryProgGroup function checks for the existence of a specific program group. On Windows NT systems, this function also determines whether a program group is of Common or Personal type. If InstallShield finds the program group, QueryProgGroup returns the following:

- The program group's attributes
- The attributes include the path of the group
- the number of items in the group
- The group type (Windows NT only)

Since there are no program groups in Windows 95, this function does not apply to it.

Parameters

szGroupName

Enter the name of the program group you are looking for. 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)"'
```

svGroupPath

Contains the fully-qualified path of the group specified in szGroupName. Under Windows NT, svGroupPath is ignored because it is stored in the registry.

nvItemCount

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, where 0 = Personal and 1 = Common.

Return values

0

Indicates that the function successfully found the program group.

< 0

Indicates that the function was unable to find the program group.

{button ,JI(`LANGREF.HLP>Examples',`QueryProgGroup_example')} Example

{button ,AL(`GetSystemInfo;HIWORD;LOWORD;QueryProgItem',0,`',`')} See also

QueryProgGroup example

```
/*--------------------------------------------------------------------------*\
 *
 * This example illustrates the usage of the QueryProgGroup function.
 *
 * QueryProgGroup is called to check for the existence of a specific program
 * group.
 *
 * 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 "InstallShield"

   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.
 *
 * svGroupPath contains the path name of the group.  This is invalid under
 * Windows NT.
 *
\*--------------------------------------------------------------------------*/
   if (QueryProgGroup (szGroupName, svGroupPath, nvItemCount) < 0) then

      MessageBox("QueryProgGroup failed.", SEVERE);
      abort;
   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: Is5fn124.rul
```

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

Enter the name of the folder containing 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:

- Windows 95: The . .\Start Menu\Programs directory under the Windir environment variable.
- Windows NT 4.0 (if COMMON is selected with ProgDefGroupType): The . .\profiles\All Users\Start Menu\Programs directory under the Windir environment variable.
- 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.
- Windows 3.1: Not applicable.
- 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

Enter the name of the program item or subfolder you are looking for.

svCmdLine

Contains 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

Contains the full path of the working directory of the program item. (Not applicable if szItemName is a subfolder.)

svIconPath

Contains the fully-qualified filename of the .ico file or .exe file. (Not applicable if szItemName is a subfolder.)

nvIconIndex

Contains the index of the icon used for the program item. (Not applicable if szItemName is a subfolder.)

svShortCutKey

Contains the item's shortcut key. (Not applicable if szItemName is a subfolder.)

nvMinimizeFlag

(Not applicable if szItemName is a subfolder.) Contains 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

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.
      Sprintf(szInfo, "The icon path of %s: %s", szItemName, svIconPath);
      ListAddString(listInfo, szInfo, AFTER);

      // Return icon index into string list.
      Sprintf(szInfo, "The index of the icon: %d", nvIconIndex);
      ListAddString(listInfo, szInfo, AFTER);

      // Return shortcut key into string list.
      Sprintf(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.
      Sprintf(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
```

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 name of the shell manager that is currently running. Only the program name is returned, even though a path may be specified in the Win.ini file.

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

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
```

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

Enter 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

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
```

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
Enter the name of the folder that contains the icon you want to replace.

szItemName
Enter the name of the icon you are replacing.

szNewItem
Enter the name of the icon as it should appear after the replacement.

szCmdLine
Enter the full path and filename of the icon you are replacing.

szWorkingDir
Enter the full path and filename of the icon you want to use to replace the current icon.

szIconPath
Enter the name of an alternate icon file or a valid Windows executable that contains the new icon.

nIcon
Enter the icon ordinal if you specified a Window executable icon. Otherwise, enter 0 in this parameter.

szShortCutKey
Enter 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
Use this parameter to specify how you want the icon to appear. You can combine constants with the bitwise OR operator (|). These constants are available:

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 appear maximized when the user first starts the program.

RUN_MINIMIZED
Causes the program window to appear 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

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
```

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 svResultFolder 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
Enter a title for the dialog box. If the title is a literal, you must enclose it in double quotation marks. To display the default title, "Select Program Folder," enter a null string ("").

szDefFolder
Enter the name of the folder you want to display as the default folder.

svResultFolder
Returns the name of the selected folder. If the selected folder does not exist, it is nonetheless returned in this parameter, but remember that this function cannot 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

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
```

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
Enter the name of the program folder you want to display.

nCommand
Use this parameter to specify how you want the folder window to be displayed. These constants are available:

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

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
```

ShowProgramFolder

Syntax

ShowProgramFolder (szFolder, nCommand);

Description

The ShowProgramFolder function displays a program folder.

Parameters

szFolder
Enter the name of the folder you want to display.

nCommand
Specifies the state of the folder. These constants are available:

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

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
```

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.

CopyBytes

Syntax

CopyBytes (svDest, nIndexDest, svSrc, nIndexSrc, nCount);

Description

The CopyBytes function copies a specified number of bytes from one string to another string. You can specify the offset indexes into the source and destination strings.

Parameters

svDest
The destination string.

nIndexDest
The offset index (starting point) in the destination string at which the bytes are to be inserted. The first byte in the string is at position 0.

svSrc
The source string.

nIndexSrc
The offset index (starting point) in the source string at which bytes begin to be copied. The first byte in the string is at position 0.

nCount
The total number of bytes you want to copy from svSrc to svDest.

Return values

0
CopyBytes successfully copied a specified number of bytes from one string to another.

< 0
CopyBytes was unable to copy the bytes.

Comments

CopyBytes is useful when you are working with binary files.

{button ,JI(`LANGREF.HLP>Examples',`CopyBytes_example')} Example

{button ,AL(`ReadBytes;SeekBytes;WriteBytes',0,`',`')} See also

CopyBytes example

```
/*--------------------------------------------------------------------------*\
 *
 * This example illustrates the the CopyBytes function.
 *
 * AskText is called to retrieve the date and assign it to the szSrc variable.
 * The file mode is then set and the EXAMPLE_BIN file is opened.  Then
 * CopyBytes is called, copying the last four bytes (the year) of szSrc into
 * svDest.  WriteBytes writes the svDest string into EXAMPLE_BIN.  This writes
 * over the old year of the file, updating it to the current one.
 *
 * NOTE: In order for this script to function properly, the EXAMPLE_DIR and
 *       EXAMPLE_BIN constants must be set a valid directory and file,
 *       respectively.
 *
\*--------------------------------------------------------------------------*/

#define EXAMPLE_DIR "C:\\EXAMPLE"
#define EXAMPLE_BIN "EXAMPLE.BIN"

   STRING szSrc, svDest;
   NUMBER nPosition, nvFileHandle, nIndexDest, nIndexSrc, nCount;

program

   // Prompt user to enter date.
   Name:
   AskText("Please enter today's date in this form, MM-DD-YYYY:", "06-27-1996",
szSrc);

   if (szSrc = "") then

      MessageBox("Please enter a valid date.", SEVERE);
      goto Name;
   endif;

   // Set the file mode to Read/Write.
   OpenFileMode(FILE_MODE_BINARY);

   // Open a binary file for editing.
   if (OpenFile(nvFileHandle, EXAMPLE_DIR, EXAMPLE_BIN) < 0) then

      SprintfBox(INFORMATION, "CopyBytes Example", "Could not open %s.",
EXAMPLE_BIN);
    abort;
   endif;

   // Set CopyBytes variables.
   nIndexDest = 0;
   nIndexSrc  = 6;
   nCount     = 4;

/*--------------------------------------------------------------------------*\
 *
 * The following copies an nCount number of bytes.  Starting from the seventh
 * byte in szSrc, four bytes are copied into the svDest string.
 *
\*--------------------------------------------------------------------------*/
```

```
   if (CopyBytes(svDest, nIndexDest, szSrc, nIndexSrc, nCount) < 0) then

      MessageBox("CopyBytes failed.", SEVERE);
   endif;

   // Set the cursor postion to the fifty-fourth byte in the file.
   nPosition = 53;
   SeekBytes(nvFileHandle, nPosition, FILE_BIN_START);

   // Write the four bytes of svDest into the file.
   WriteBytes(nvFileHandle, svDest, 0, nCount);

   // Close the file.
   CloseFile(nvFileHandle);

endprogram

// Source file: Is5fn037.rul
```

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
Enter a fully-qualified path name.

svDir
Returns the path name without the drive letter.

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

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
```

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
Enter the full path (including the colon) that contains the drive letter you want.

svDisk
Returns the drive letter, including 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.

Comments

There must be a colon (:) in szPath. Otherwise, the function fails.

{button ,JI(`LANGREF.HLP>Examples',`GetDisk_example')} Example

{button ,AL(`GetDir;GetDiskSpace;GetSystemInfo',0,`',`')} See also

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
```

NumToStr

Syntax

NumToStr (svString, nValue);

Description

The NumToStr function converts a number to a string.

Parameters

svString
Returns the string equivalent of nValue.

nValue
Enter the number you want to convert into 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

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
```

ParsePath

Syntax

ParsePath (svReturnString, szPath, nOperation);

Description

The ParsePath function retrieves the specified part of an existing path string without using direct string manipulation. The function works with any valid path, including short paths, long paths, and UNC paths that may or may not include a specific filename. These are some sample paths that can be parsed with this function.

- \\Path1\\Path2\\Filename.exe
- Filename
- Filename.exe
- \\Path1\\Path2\\Filename
- D:
- D:\\
- \\\\Server Name\\Share Name\\Share Directory
- Any other legal DOS path

Parameters

svReturnString

Contains the return value returned by this function based on the operation you specify.

szPath

Enter the path you want to parse (separate).

nOperation

Use this parameter to specify what you want to parse from the path. These constants are available:

DIRECTORY

Returns a path minus the disk drive letter and the filename. When parsing a path that does not include a filename, you must append a backslash to the end of the path before passing it to the function; otherwise the last part of the path will be interpreted as a filename and will not be included in the path that is returned by the function.

```
Incorrect:   szPath = "D:\\PATH1\\PATH2";
             ParsePath (svReturnString, szPath, DIRECTORY);


Correct:     szPath = "D:\\PATH1\\PATH2\\";
             ParsePath (svReturnString, szPath, DIRECTORY);
```

When this option is used with a UNC path, which has no drive letter, the path minus the filename will be returned.

DISK

Returns the disk drive letter followed by a colon. If this constant is specified in nOperation when szPath contains a UNC path, svReturnString will return a null string ("").

EXTENSION_ONLY

Returns the file extension. It does not include the period.

FILENAME

Returns the filename with its file extension.

FILENAME_ONLY
Returns only the filename without its file extension.

PATH
Returns the full disk drive and directory with no filename. When this option is used with a UNC path, which has no drive letter, the path minus the filename will be returned; this option is equivalent to the DIRECTORY option for UNC paths.

Return values

0
Indicates that the function successfully parsed the path string.

< 0
Indicates that the function was unable to parse the path string.

{button ,JI(`LANGREF.HLP>Examples',`ParsePath_example')} Example

{button ,AL(`DeleteFile;GetDisk',0,`',`')} See also

ParsePath example

```
/*--------------------------------------------------------------------------*\
 *
 * This example illustrates the usage of the ParsePath function.
 *
 * The ParsePath is called six times to retrieve various information from a
 * path statement.
 *
\*--------------------------------------------------------------------------*/

#define EXAMPLE_PATH "C:\\EXAMPLE\\README.TXT"

   STRING szPath, svReturnString, szTitle, szMsg;
   NUMBER nOperation;

program

/*--------------------------------------------------------------------------*\
 *
 * ParsePath is called to retrieve the disk letter from the path.
 *
\*--------------------------------------------------------------------------*/
   szPath = EXAMPLE_PATH;
   nOperation = DISK;
   if (ParsePath(svReturnString, szPath, nOperation) < 0) then

     MessageBox("ParsePath failed", SEVERE);
   endif;

   szTitle = "ParsePath example";
   szMsg   = "nOperation = DISK\n\nParsed Path: %s";
   SprintfBox(INFORMATION, szTitle, szMsg, svReturnString);

/*--------------------------------------------------------------------------*\
 *
 * ParsePath is called to retrieve the path.
 *
\*--------------------------------------------------------------------------*/
   nOperation = PATH;
   ParsePath(svReturnString, szPath, nOperation);

   szMsg   = "nOperation = PATH\n\nParsed Path: %s";
   SprintfBox(INFORMATION, szTitle, szMsg, svReturnString);

/*--------------------------------------------------------------------------*\
 *
 * ParsePath retrieves the directory name.
 *
\*--------------------------------------------------------------------------*/
   nOperation = DIRECTORY;
   ParsePath(svReturnString, szPath, nOperation);

   szMsg   = "nOperation = DIRECTORY\n\nParsed Path: %s";
   SprintfBox(INFORMATION, szTitle, szMsg, svReturnString);

/*--------------------------------------------------------------------------*\
 *
 * ParsePath retrieves the filename extension.
 *
```

```
\*---------------------------------------------------------------------------*/
   nOperation = FILENAME;
   ParsePath(svReturnString, szPath, nOperation);

   szMsg   = "nOperation = FILENAME\n\nParsed Path: %s";
   SprintfBox(INFORMATION, szTitle, szMsg, svReturnString);

/*---------------------------------------------------------------------------*\
 *
 * ParsePath retrieves the filename without the extension.
 *
\*---------------------------------------------------------------------------*/
   nOperation = FILENAME_ONLY;
   ParsePath(svReturnString, szPath, nOperation);

   szMsg   = "nOperation = FILE_NAME_ONLY\n\nParsed Path: %s";
   SprintfBox(INFORMATION, szTitle, szMsg, svReturnString);

/*---------------------------------------------------------------------------*\
 *
 * ParsePath retrieves only the file extension from the path.
 *
\*---------------------------------------------------------------------------*/
   nOperation = EXTENSION_ONLY;
   ParsePath(svReturnString, szPath, nOperation);

   szMsg   = "nOperation = EXTENSION_ONLY\n\nParsed Path: %s";
   SprintfBox(INFORMATION, szTitle, szMsg, svReturnString);

endprogram

// Source file: Is5fn115.rul
```

StrCompare

Syntax

StrCompare (szStringA, szStringB);

Description

The StrCompare function performs compares two strings. It is not case-sensitive.

Parameters

szStringA
Enter the first string to compare.

szStringB
Enter 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

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
```

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
Enter the string that you want to search for the szFindMe string.

szFindMe
Enter the string you are searching for.

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

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
```

StrGetTokens

Syntax

StrGetTokens (listID, szString, szDelimiterSet);

Description

The StrGetTokens function separates a string into a list of tokens. StrGetTokens uses delimiters to extract the tokens from the string. You type these delimiters in the parameter szDelimiterSet. When StrGetTokens gets the tokens from szString, it puts them in the list identified by listID.

There are a few things users should be aware of when trying to parse and rebuild a string. The first item on the target list is whatever precedes the first delimiter character in the string. If the delimiter character is the first character of the string, the first element of the list will be set to null. The string will then be parsed until the next delimiter character is reached. If the last character of the string happens to be the delimiter character, a null string ("") will be placed at the end of the list.

Use the list functions, such as ListGetFirstString and ListGetNextString to access each token in the list.

Parameters

listID
Enter the ID of a list you want to use to store the tokens. Before you use StrGetTokens, you need to create this list with the ListCreate function. After the StrGetTokens function executes, each element of this list will contain one token.

szString
Enter the string containing the tokens you want to separate.

szDelimiterSet
Enter a set of one or more delimiters. Each delimiter is one character (1 byte). If you enter a null string (""), the function searches for null characters as the delimiters. This is useful if you are using the GetProfString function.

Return values

0
Indicates that the function successfully separated the string and inserted the tokens into the specified list.

< 0
Indicates that the function was unable to separate the string and insert the tokens into the list.

{button ,JI(`LANGREF.HLP>Examples',`StrGetTokens_example')} Example

{button ,AL(`StrCompare;StrFind;StrLength;StrSub;StrToLower;StrToNum;StrToUpper',0,`',`')} See also

StrGetTokens example

```
/*--------------------------------------------------------------------------*\
 *
 * This example illustrates the usage of the StrGetTokens function.
 *
 * NOTE: In order for this script to run properly, it is advised to set the
 *       EXAMPLE_INI constant to a valid initialization file on the target
 *       system.
 *
\*--------------------------------------------------------------------------*/

#define EXAMPLE_INI "EXAMPLE\\EXAMPLE.INI"

   #include "Sddialog.h";

   LIST   listID;
   STRING szString, szDelimiterSet, svString, svResult, szPath;
   STRING szTitle, szMsg;
   NUMBER nResult;

program

   // Set necessary variables.
   listID         = ListCreate(STRINGLIST);
   szString       = "C:\\DOS;>C:\\WINDOWS|C:\\PROGRAM FILES\\>F:\\USERS";
   szDelimiterSet = "|>";

/*--------------------------------------------------------------------------*\
 *
 * The following divides the szString string into tokens that are inserted
 * into the listID list variable.  The tokens in szString are divided by
 * delimiters in szDelimiterSet.
 *
\*--------------------------------------------------------------------------*/

   if (StrGetTokens(listID, szString, szDelimiterSet) < 0) then
     MessageBox("StrGetTokens failed.", SEVERE);
   endif;

   // Display the tokens in the list.
   szTitle = "StrGetTokens Example";
   szMsg   = "The following is a list of the strings in listID:";
   SdShowInfoList(szTitle, szMsg, listID);

   // Destroy the list.
   ListDestroy(listID);

   // Return the keys of an .INI file into the svResult variable.
   szPath = EXAMPLE_INI;
   GetProfString(szPath, "Old Section", "", svResult);

   // Set new variables.
   listID = ListCreate(STRINGLIST);
   szDelimiterSet = "";

/*--------------------------------------------------------------------------*\
 *
 * The following divides szString into tokens that are returned
 * to the listID list variable.  The tokens in szString are divided by
 * delimiters in szDelimiterSet.
 *
\*---------------------------------------------------------------------------*/

   if (StrGetTokens(listID, svResult, szDelimiterSet) > 0) then
      MessageBox("StrGetTokens failed.", SEVERE);
   endif;

   // Display tokens in new list.
   szMsg   = "The following is a list of the keys in 'Old Section'.";
   SdShowInfoList(szTitle, szMsg, listID);

   ListDestroy(listID);

endprogram

#include "Sddialog.rul"

// Source file: Is5fn173.rul
```

StrLength

Syntax

StrLength (szString);

Description

Use the StrLength function to find the length of a string.

Parameters

szString
Enter the string for which you want to determine the length.

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

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
```

StrRemoveLastSlash

Syntax

StrRemoveLastSlash (svString);

Description

The StrRemoveLastSlash function removes the last backslash from a string.

Parameters

svString

Enter the string that contains the backslash you want to remove. After the function executes, this parameter will contain the modified 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. You can 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

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
```

StrSub

Syntax

StrSub (svSubStr, szString, nStart, nLength);

Description

The StrSub function extracts part of a string. StrSub allows you to specify the source string you wish to extract from, the position at which to begin the extraction, and the number of characters you want to extract.

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 string extracted by the function.

szString

Enter the source string.

nStart

Enter the numeric position at which you want to start extracting the substring. The position of the first character in szString is 0 (zero).

nLength

Enter the number of characters you want to copy from szString. This number cannot exceed the length of the source string.

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

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
```

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 converted string in lowercase letters.

szSource
Enter the string you want to convert to all lowercase.

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

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
```

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:

- If all of the characters in the string are in the range "0".."9", the number represented by the string is assigned to nvVar.
- 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.
- 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.
- 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 parameter szString.

szString
Enter the numeric string you want to convert into 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

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 nvVar.
 *
\*--------------------------------------------------------------------------*/
   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 nvVar.
 *
\*--------------------------------------------------------------------------*/
   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
```

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 converted string in uppercase letters.

szSource
Enter the string you want to convert to all uppercase.

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

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
```

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.

VerCompare

Syntax

VerCompare (szVersionInfo1, szVersionInfo2, nCompareFlag);

Description

The VerCompare function compares two strings that contain version information.

Parameters

szVersionInfo1
Enter 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
Enter the second version string in the same format.

nCompareFlag
Enter the constant VERSION to specify you are comparing version numbers. You cannot enter any other constant 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

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 %s.  Copying %s to %s.";
      SprintfBox(INFORMATION, szTitle, szMsg, szFileName, 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
```

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
Enter only the name of the file whose version you want to find. You cannot specify a path in this parameter.

svPath
Returns the fully-qualified path of the file, if found.

svVersionNumber
Returns the version number of the file, if found, in the following format:

```
Major version number.Minor version number
```

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 value of TARGETDIR before temporarily setting it to the desired 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

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, 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
```

VerGetFileVersion

Syntax

VerGetFileVersion (szFileName, svVersionNumber);

Description

The VerGetFileVersion function retrieves the version of a specified file.

Parameters

szFileName
Enter the fully-qualified name of the file whose version number you want to retrieve.

svVersionNumber
Returns the version number in this format:

```
Major version number.Minor version number
```

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

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
```

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

Enter only the name of the file you want to install. You cannot include path information 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. These constants are available:

VER_UPDATE_COND

Updates the file only if the file being replaced is an older version.

VER_UPDATE_ALWAYS

Updates the file even if the file being replaced is a newer version.

svInstalledFile

Contains 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

- 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 values of SRCDIR and TARGETDIR before temporarily setting them to the desired values. After the VerSearchAndUpdateFile function call, reset SRCDIR and TARGETDIR using VarRestore.
- 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

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
```

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

Enter the name of the target file—the file on the target system that you want to update. You can enter a fully-qualified filename or just the filename, without a path . If you enter just the filename, InstallShield searches the Windows or Win95 directory, the System directory, the directories specified by the PATH environmental 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

Use this parameter to specify 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. The constants listed below are available. 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 path and filename of the file that the function 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

- 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.
- 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 value of SRCDIR before temporarily setting it to the desired directory. After the VerUpdateFile function call, reset SRCDIR using VarRestore.
- 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;VerGetFileVersion;Installing shared files with XCopyFile and VerUpdateFile;Installing shared files with XCopyFile and VerUpdateFile;Installing self-registering files with XCopyFile and VerUpdateFile',0,`',`')} See also

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
```

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.

Disable

Syntax

Disable (nConstant);

Description

The Disable function disables user interface objects, graphical capabilities, or functionality in the setup program. When you disable a user interface object, it is either not visible or is grayed out. You can disable a user interface object at any point in the script.

Parameters

nConstant

Use this parameter to specify the object or event you want to disable. These constants are available:

BACKBUTTON
Disables (grays out) the Back button in dialog boxes.

BACKGROUND
Disables the main background window.

BACKGROUNDCAPTION
Disables the caption bar on the background window.

BITMAP256COLORS
Disables 256-color mode.

CORECOMPONENTHANDLING
Disables core component handling.

DIALOGCACHE
Disables the dialog Cache mechanism.

FEEDBACK_FULL
Disables the information gauges.

HOURGLASS
Disables the hourglass, causing the mouse cursor to change from an hourglass back to its default shape.

INDVFILESTATUS
Does not allow individual filenames to appear on the second line of the progress indicator when InstallShield performs transfer file operations.

LOGGING
Disables the logging of uninstallation information. This is used when it is desired to have files copied or registry entries created that will not be uninstalled when unInstallShield runs.

NEXTBUTTON
Disables (grays out) the Next button in the dialog box.

SELFREGISTERBATCH
Disables queuing of self-registering files when using the batch method for installing self-registering files. For more information, click here.

STATUS
Disables the progress indicator.

Return values

0

Indicates that the function successfully disabled the feature.

< 0
Indicates that the function was unable to disable the feature.

Comments

- Other constants may be available for use with the Enable and Disable functions. These constants are available for compatibility with previous releases of InstallShield. You should use only the constants listed above in your scripts.
- Once either the Next or Back button is disabled, it must be enabled before bringing up any other dialog; otherwise, the corresponding button will not be active.
- Note that DIALOGCACHE works only with those dialogs that have a Next button. It must be disabled before using any dialog that does not have a Next button.
- InstallShield displays the information gauges by default. The information gauges are the three feedback gauges that appear to the left of the file transfer progress indicator. If you do not want them to appear, disable them with the constant FEEDBACK_FULL.

{button ,JI(`LANGREF.HLP>Examples',`Disable_example')} Example

{button ,AL(`Enable',0,`',`')} See also

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
```

Enable

Syntax

Enable (nConstant);

Description

The Enable function enables specific user interface objects, graphical capabilities, or functionality. When you enable a user interface object, it appears. You can enable a user interface object at any point in the script.

Parameters

nConstant

Use this parameter to specify the object or event you want to enable. These constants are available:

BACKBUTTON
Enables the Back button in dialog boxes.

BACKGROUND
Enables the main background window (must be enabled in DEFWINDOWMODE).

BACKGROUNDCAPTION
Enables the caption bar on the background (in FULLWINDOWMODE) or main window (in DEFWINDOWMODE).

BITMAP256COLORS
Enables 256-color mode. Transparent bitmaps do not support 256 colors. If using transparent bitmaps, use the default 16 color mode. Refer to PlaceBitmap.

CORECOMPONENTHANDLING
Causes InstallShield to install files transferred to the Windows\System directory with the SHAREDFILE option to be installed as prescribed in Corecomp.ini, a file compressed into _sys1.cab that instructs InstallShield on how to install Windows 95 and Windows NT 4.0 core components.

DEFWINDOWMODE
Causes InstallShield to startup in a window mode as opposed to full-screen mode. In this mode, you must perform an enable on the BACKGROUND to display the main window. The first statement of the script must enable the DEFWINDOWMODE.

DIALOGCACHE
Enables the dialog Cache mechanism. This mechanism eliminates the screen flash that appears between displaying of two dialogs boxes. However, this parameter can be used only if the dialog has a Back button.

FULLWINDOWMODE
Causes InstallShield to startup in full-screen mode, and also allows you to display a caption bar using the constant BACKGROUNDCAPTION in a subsequent function call.

HOURGLASS
Enables the hourglass, causing the mouse cursor to change to an hourglass during busy states.

INDVFILESTATUS
Enables individual filenames to appear on the second line of the progress indicator when InstallShield performs transfer file operations.

LOGGING
Enables the logging of uninstallation information. This value is used when it is desired to have files copied or registry entries created that *will* be uninstalled when unInstallShield runs. Since Enable(LOGGING) is the default condition, it needs to be called only when you want to reverse the effect of a previous call to Disable(LOGGING).

NEXTBUTTON
Enables the Next button in dialog boxes.

SELFREGISTERBATCH
Enables queuing of self-registering files when using the batch method to install self-registering files. For more information, click here.

STATUS
Enables the progress indicator.

STATUSDLG
Enables the dialog style progress indicator.

STATUSOLD
Enables a progress bar without the Cancel button. Use F3 button to exit while the progress bar displays.

Return values

0
Indicates that the function successfully enabled the specified feature.

< 0
Indicates that the function was unable to enable the specified feature.

CC_ERR_OUTOFMEMORY
Memory could not be allocated to core component handling. Close open applications or reboot machine.

CC_ERR_NOCOMPONENTLIST
No entries are available under Corecomp.ini's sections. Make sure that the file is valid.

CC_ERR_FILEFORMATERROR
A line in Corecomp.ini is not formatted properly. Verify any modifications made to the file.

CC_ERR_FILEREADERROR
InstallShield is unable to read Corecomp.ini. Make sure that the file is valid. It is located in InstallShield5 Professional Edition\Redistributable\Compressed Files\Language Independent\OS Independent.

Comments

- Other constants may be available for use with the Enable and Disable functions. These constants are available for compatibility with previous releases of InstallShield. You should use only the constants listed above in your scripts.
- Once either Next or Back button is disabled, it must be enabled before bringing up any other dialog. If not, the corresponding button will not be active.
- Note that DIALOGCACHE works only with those dialogs that have the Back button. This must be disabled before using any dialog that does not have the Back button.
- If you are calling Enable(CORECOMPONENTHANDLING) it is very important that you check the return values to make sure that core component handling successfully initialized. You may not want to proceed with the setup if Enable failed.

{button ,JI(`LANGREF.HLP>Examples',`Enable_example')} Example

{button ,AL(`Disable;PlaceBitmap',0,`',`')} See also

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
```

FindWindow

Syntax

FindWindow (szClassName, szWinName);

Description

The FindWindow function provides a way for advanced developers to get a handle to a window by specifying its window class and window name. If you know the class and window name of an application, you can get its handle. You can use the window handle to send messages directly to the window.

Parameters

szClassName
The name of the class to which the window belongs.

szWinName
The Window caption (title) whose handle you want to retrieve.

Return values

XXXX
The handle of the window.

NULL
FindWindow was unable to find a window with the specified name and class name.

Comments

To find the class and name of a window, run the Microsoft Spy.exe program.

{button ,JI(`LANGREF.HLP>Examples',`FindWindow_example')} Example

{button ,AL(`AppCommand;LaunchApp;SendMessage',0,`',`')} See also

FindWindow example

```
/*--------------------------------------------------------------------------*\
 *
 * This example illustrates the usage of the FindWindow and SendMessage
 * functions.
 *
 * FindWindow is first called to look for the Notepad window.  If found, the
 * window is maximized, and then minimized (after a three-second delay) using
 * system commands by calling SendMessage.  Notice that the Windows messages
 * must be defined before using them.
 *
 * NOTE: In order for this script to run properly, it is advised to set the
 *       NOTEPAD_EXE constant to the windows executable on the target system.
 *
\*--------------------------------------------------------------------------*/

// Define Windows messages to InstallShield.
#define WM_SYSCOMMAND 0x0112
#define SC_MINIMIZE   0xF020
#define SC_MAXIMIZE   0xF030

#define NOTEPAD_EXE   "EXAMPLE\\NOTEPAD.EXE"

   STRING szClassName, szWinName;
   NUMBER nMsg, nwParam, nlParam;
   HWND nHwnd;

program

   Disable(BACKGROUND);

   LaunchApp(NOTEPAD_EXE, "");

   szClassName = "NOTEPAD";
   szWinName   = "";
/*--------------------------------------------------------------------------*\
 *
 * The following retrieves the handle of the Notepad window.
 *
\*--------------------------------------------------------------------------*/
   nHwnd = FindWindow(szClassName, szWinName);

   if (nHwnd = NULL) then

      MessageBox("FindWindow failed.", SEVERE);
   else

      SprintfBox(INFORMATION, "FindWindow & SendMessage", "Handle: %i",
                 nHwnd);

      nMsg    = WM_SYSCOMMAND;
      nwParam = SC_MAXIMIZE;
      nlParam = 0;
/*--------------------------------------------------------------------------*\
 *
 * The following sends maximize and minimize system commands to the window.
 *
\*--------------------------------------------------------------------------*/
      SendMessage(nHwnd, nMsg, nwParam, nlParam);
      Delay(3);
      nwParam = SC_MINIMIZE;
      SendMessage(nHwnd, nMsg, nwParam, nlParam);
   endif;

endprogram

// Source file: Is5fn078.rul
```

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:

- By passing one of the location constants in the parameter nDrawOp.
- 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

Enter 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

If the bitmap resides in a .dll, enter its resource ID. Otherwise, if the bitmap source is a metafile or bitmap file, set nID_BITMAP to 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

Enter 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

Enter 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

Use the following constants in this parameter to specify the bitmap placement location, set placement options, or remove a previously placed bitmap:

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

- Call SetDisplayEffect to set special effects for non-tiled, full-screen, transparent bitmaps; you can also set limited special effects for metafiles.
- 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.
- 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.
- 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.
- 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.
- 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',0,`',`')} See also

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
```

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

Use this parameter to specify the object whose position you want to change. These constants are available:

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

Enter 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

Enter 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

Use this parameter to specify from which corner to measure the distances, expressed in nDx and nDy. These constants are available:

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

- 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.
- 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.
- To center the information gauge, call PlaceWindow as shown below.

```
PlaceWindow(FEEDBACK, CENTERED, CENTERED, CENTERED);
```

- 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',0,`',`')} See also

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
```

PlayMMedia

Syntax

PlayMMedia (nType, szFileName, nOperation, nReserved);

Description

The PlayMMedia function plays a sound or AVI file. Sound files of type MIDI or WAVE and AVI files can be played either in synchronous mode or in asynchronous mode. This function can also be used to play a file continuously. To do so, a flag can be bitwise ORed, which indicates that the file is to be run in continuous mode.

Because AVI files are already tightly compressed and are often larger than 1.4 MB diskettes, video presentations are recommended only for CD-ROM setups.

Parameters

nType

Use one of the following predefined constants to identify the file type.

MMEDIA_WAVE

The file is of WAVE sound format.

MMEDIA_MIDI

The file is of MIDI sound format.

MMEDIA_AVI

The file is an AVI file.

szFileName

The fully-qualified name of the sound/AVI file to be played.

nOperation

Use of the following constants to indicate the play mode:

MMEDIA_PLAYSYNCH

Play synchronously.

MMEDIA_PLAYASYNCH

Play asynchronously.

MMEDIA_PLAYCONTINUOUS

Play in a continuous loop. This value cannot be used when playing a sound/AVI file in synchronous mode. It can be used only with files being played in asynchronous mode.

MMEDIA_STOP

Stop playing.

nReserved:

Enter 0 for this parameter, which is reserved for future use.

Return values

0

Indicates that the function successfully played the file.

< 0

Indicates that the function was unable to play the file.

{button ,JI(`LANGREF.HLP>Examples',`PlayMMedia_example')}   Example

PlayMMedia example

```
/*--------------------------------------------------------------------------*\
 *
 *  This example illustrates the use of PlayMMedia.
 *
 *  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. Then add an AVI file to disk 1
 *             in the Setup Files pane in the IDE. Change the filename in
 *             #define SOURCE line below to specify your AVI file.
 *
 *             IMPORTANT: Since this example does not include uninstallation
 *             functionality, use this example only with projects that do
 *             not overwrite important files, install shared files, or update
 *             the registry.
 *
\*--------------------------------------------------------------------------*/

#define SOURCE  SRCDIR + "windy7(1).avi"
#define DESTDIR "c:\\temp"
#define TITLE1  "Playing AVI synchronously..."
#define TITLE2  "Playing AVI asynchronously and continuously..."

NUMBER nvDisk;

program

    // First, play the AVI synchronously to demonstrate how this causes
    // it to play by itself, with no other events taking place.
    SetTitle( TITLE1, 16, YELLOW );
    PlaceWindow( MMEDIA_AVI, 10, 10, UPPER_RIGHT );

    if( PlayMMedia( MMEDIA_AVI, SOURCE,
                    MMEDIA_PLAYSYNCH, 0 ) < 0 ) then
        MessageBox( "Unable to play AVI file.", WARNING );
    endif;

    // Now play the AVI asynchronously and continously, meaning that
    // it will play at the same time as the following events (file
    // transfer in this case) until you stop it or the setup ends.
    SetTitle( TITLE2, 16, YELLOW );
    PlaceWindow( MMEDIA_AVI, 10, 10, LOWER_RIGHT );

    if( PlayMMedia( MMEDIA_AVI, SOURCE,
                    MMEDIA_PLAYASYNCH | MMEDIA_PLAYCONTINUOUS, 0 ) < 0 ) then
        MessageBox( "Unable to play AVI file.", WARNING );
    endif;

    Enable( STATUSDLG );
    Enable( INDVFILESTATUS );
    StatusUpdate( ON, 99 );
    TARGETDIR = DESTDIR;
     ComponentMoveData( MEDIA, nvDisk, 0 );
    Disable( INDVFILESTATUS );
    Disable( STATUSDLG );

    // Stop the AVI playing. End of setup will also stop it. But if
    // other setup events occur, you may wish to stop it thus.
    PlayMMedia( MMEDIA_AVI, SOURCE, MMEDIA_STOP, 0 );
```

```
endprogram

// Source file: Is5fn606.rul
```

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
A numeric constant, ranging in value from 0..255, indicates the amount of red in the custom color.

constGreen
A numeric constant, ranging in value from 0..255, indicates the amount of green in the custom color.

constBlue
A numeric constant, ranging in value from 0..255, indicates the amount of green 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

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
```

SetColor

Syntax

SetColor (nObject, nColor);

Description

The SetColor function sets the color of two user interface objects.

Parameters

nObject

Use this parameter to specify which user interface object you want to change. These objects are available:

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

Use this parameter to specify the color you want to apply to the user interface objects. You can specify the color by using the constants described below or by passing the function RGB in this parameter.

The following constants are available for the background color:

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 smoothing effect (gradient) when the background is painted. It should be noted that the smoothing effect will be better with 256 colors enabled than disabled.

The following constants are available for the progress bar:

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(`Enable',0,`',`')} See also

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
```

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

Use this parameter to identify the built-in dialog box whose title you are changing. These constants are available:

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

Enter the text you want as the new title. You must enclose the text in double quotation marks.

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

- You must call SetDialogTitle separately for each dialog box title you wish to change.
- 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;AskText;AskYesNo;MessageBox;EnterDisk',0,`',`')} See also

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
```

SetDisplayEffect

Syntax

SetDisplayEffect (nEffect);

Description

The SetDisplayEffect function sets the display effect for default bitmaps in Sd dialogs. Once the effect is set, it remains in effect until changed by another call to SetDisplayEffect.

Parameters

nEffect

Use one of the constants below to specify an effect. Note that these constants are exclusive; they cannot be bitwise ORed with each other. Moreover, this parameter has no effect if BITMAPICON, BITMAPFULLSCREEN, or BITMAPTILE styles are used with PlaceBitmap when the bitmap is displayed.

EFF_FADE

The bitmap or billboard slowly fades in and out.

EFF_REVEAL

The bitmap or billboard gradually fills in from the center toward all sides.

EFF_HORZREVEAL

The bitmap or billboard gradually scrolls out horizontally from its center.

EFF_HORZSTRIPE

A section of the bitmap or billboard fills in horizontally from the outside in; then the remainder fills in from the center out.

EFF_VERTSTRIPE

A section of the bitmap or billboard fills in vertically from the outside in; then the remainder fills in from the center out.

EFF_BOXSTRIPE

A section of the bitmap or billboard fills in from all sides; then the remainder fills in toward all sides.

EFF_NONE

This option is the default setting. Use it to clear the display effects after calling one of the other options.

Only EFF_REVEAL and EFF_HORZREVEAL work with metafiles.

Return values

0

Indicates that the function successfully set the display effect.

< 0

Indicates that the function was unable to set the display effect.

Comments

These effects are ignored for full screen, tile or transparent bitmaps.

{button ,JI(`LANGREF.HLP>Examples',`SetDisplayEffect_example')} Example {button ,AL(`PlaceBitmap',0,`',`')}   See also

SetDisplayEffect example

```
/*--------------------------------------------------------------------------*\
 * This example illustrates the uses of the SetDisplayEffect function.
 *
 * PlaceBitmap is called three times to display a bitmap on the screen.  The
 * SetDisplayEffect function sets the display effect for bitmap or metafile
 *  images.
 *
 * 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:\\windows\\clouds.BMP"

   STRING szBitmap;
   NUMBER nID_BITMAP, nDx, nDy, nCorner;

program

   szBitmap   = BMP_PATH;
   nID_BITMAP = 12;
   nDx        = 10;
   nDy        = 10;
   nCorner    = UPPER_LEFT;

   // Display image using the fading effect
   SetDisplayEffect(EFF_FADE);
   // 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;


   // Display image using the box stripe display effect
   SetDisplayEffect(EFF_BOXSTRIPE);

   // 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);


   // Display image using the horizontal revealing effect
   SetDisplayEffect(EFF_HORZREVEAL);
   // 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: Is5fn607.rul
```

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

Use this parameter to specify which error message you want to replace. These constants are available:

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

Enter the error message you want to appear 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

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
```

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

Use this parameter to specify the error message box whose title you want to replace. These constants are available:

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

Enter the error message you want to appear 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

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
```

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

Use this parameter to specify the item whose font and font style you want to set. This constant is available:

FONT_TITLE

Specifies the main title of the setup procedure, which is displayed in the top left corner of the setup window.

nFontStyle

Specify font styles with the following predefined constants. 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

Enter the name of an available Windows font. Valid font names include Courier, Helv, Helvetica, Modern, Roman, Script, Terminal, Times, and TmsRmn. Consult the Windows SDK documentation for current available fonts. If the specified font is not found with the styles specified, Arial will be 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

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
```

SetStatusWindow

Syntax

SetStatusWindow (nPercent, szString);

Description

The SetStatusWindow function enables the progress indicator and allows you to set an initial percentage number for the progress bar. You can also use SetStatusWindow to change the top message string in the progress indicator. SetStatusWindow uses the value passed as the parameter szString to update the top line of text in the progress indicator; however, It does not change the text of any other line.

Call SetStatusWindow whenever you want to display a progress indicator for a setup event, for example before starting a of file transfer with ComponentMoveData or XCopyFile. Be sure to call StatusUpdate once after the call to SetStatusWindow and again after each call to a function that is a part of the event.

If you call Enable with the constant INDVFILESTATUS before using this function, the StatusUpdate function will automatically display the name of the file being copied or decompressed by InstallShield. The filename appears in the Feedback line. The default setting for the Feedback line is the bottom line in the progress indicator. InstallShield uses the Feedback line in the copy, compress, and decompress operations.

During data transfers with ComponentMoveData, the component property Status Text that is stored in the media library will change the message displayed in the upper left-hand corner of the progress indicator (the szString value).

Parameters

nPercent

Enter a number that reflects the percentage of the setup process that is complete at that point in the script. This number will be the initial progress bar setting for the particular stage of the setup. The number must be an integer in the range 0 and 100. For example, if the setup is half finished, enter 50 in this parameter. When using SetStatusWindow to change the top message string in the progress indicator, enter -1 in this parameter position.

szString

Enter a brief message that explains which file is being copied or what the setup program is doing. The string is displayed in the upper left-hand corner of the progress indicator.

Return values

There are no return values for this function.

{button ,JI(`LANGREF.HLP>Examples',`SetStatusWindow_example')}   Example

{button ,AL(`Enable;StatusUpdate',0,`',`')}   See also

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
```

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

Enter the string you want to use as the title. The title must be a single line of text, no more than 80 characters in length.

nPointSize

Enter the size, in points, of the text you want. The recommended size is 28 points. Experiment to see which size you like best.

nColor

Use this parameter to set the color of the text. These colors are available: 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);
Enable(BACKGROUNDCAPTION);
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.

{button ,JI(`LANGREF.HLP>Examples',`SetTitle_example')} Example

{button ,AL(`Enable',0,`',`')} See also

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
```

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

Enter the object you want to size. These constants are available:

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

Enter the horizontal size of the object in pixels.

nDy

Enter 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',0,`',`')} See also

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
```

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

This parameter specifies whether to enable or disable the link between file transfer operations and the progress indicator of the status bar. You can use one of two constants as follows:

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

When bLink is ON, enter the final percentage value that the progress indicator of the status bar should reach at the end of the next file transfer operation. If the value you enter 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.

- 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',0,`',`')}   See also

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
```

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.

InstallShield Help found the Dialog Sampler but could not run it.

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.

```
IfThen(IsMark("SamplerWasOpened"), "Back()");     ExecFile(`../Dialog Sampler/Sampler.exe', SdWelcome,
SW_SHOW,cantrun); IfThen(not(IsMark("SamplerWasOpened")),SaveMark("SamplerWasOpened"))
```

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.

_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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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_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_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_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_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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

DLG_CDIR

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

Click here for a list of functions that use this constant.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

EFF_HORZREVEAL

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

Click here for a list of functions that use this constant.

EFF_HORZSTRIPE

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

Click here for a list of functions that use this constant.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

FS_GENERROR

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

Click here for a list of functions that use this constant.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

{

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

IISLANG_SLOVAK_STANDARD

ISLANG_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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

IISLANG_UKRAINIAN_STANDARD

ISLANG_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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.