Using the Resource Compiler

The Microsoft Windows Resource Compiler (RC) is a tool for the Microsoft Windows operating system. This overview describes how to create a resource-definition (script) file, and how to compile your application's resources and add them to the application.

About the Resource Compiler

To include resources in your Win32-based application with RC, do the following:

1. Create individual resource files for your cursors, icons, bitmaps, dialog boxes, and fonts. To do this, you can use the Microsoft Image Editor (IMAGEDIT.EXE), Dialog Editor (DLGEDIT.EXE), and Font Editor (FONTEDIT.EXE) included with the Win32 SDK or equivalent tools.
2. Create a resource-definition script (.RC file) that describes all the resources used by your application. For more information, see Creating a Resource-Definition File.
3. Compile the script into a resource file with RC. For more information, see Using RC (The RC Command Line).
4. Link the compiled resource file into the application's executable file with your linker.

Creating a Resource-Definition File

After creating individual resource files for your application's icon, cursor, font, bitmap, and dialog-box resources, you create a resource-definition file, or *script*. A script file is a text file with the extension .RC.

The script lists every resource in your application and describes some types of resources in great detail. For a resource that exists in a separate file, such as an icon or cursor, the script names the resource and the file that contains it. For some resources, such as a menu, the entire definition of the resource exists within the script.

A script file can contain the following types of information:

- Preprocessor directives, which instruct RC to perform actions on the script before compiling it. Directives can also assign values to names.
- Resource-definition statements, which name and describe resources. These consist of single-line statements and multiline statements.

The following sections describe directives and statements you can use in a script.

Preprocessor Directives

The following directives can be used as needed in the script to instruct RC to perform actions or to assign values to names:

Directive	Description
#define	Defines a specified name by assigning it a given value.
#elif	Marks an optional clause of a conditional-compilation block.
#else	Marks the last optional clause of a conditional-compilation block.
#endif	Marks the end of a conditional-compilation block.
#if	Conditionally compiles the script if a specified expression is true.
#ifdef	Conditionally compiles the script if a specified name is defined.
#ifndef	Conditionally compiles the script if a specified name is not defined.
#include	Copies the contents of a file into the resource-definition file.
#undef	Removes the definition of the specified name.

The syntax and semantics for the RC preprocessor are the same as for a C compiler. For more information on preprocessing in RC, see Preprocessor Directives Reference.

It's a good idea to test whether your files are being compiled by the RC compiler or a C compiler, because there are some directives that you can use in a header file that the C compiler will accept but RC will not. RC defines RC_INVOKED. Therefore, use

```
#ifndef RC_INVOKED
```

and

```
#endif
```

in header files that will be included in both .RC and .C files to surround things that RC can not compile.

Resource-Definition Statements

This section describes the statements that define the resources that the resource compiler puts in the resource (.RES) file. Once a resource is linked to the executable file, the application can load the resource as it is needed at run time. All resource statements associate an identifying name or number with a given resource.

Common Control Parameters

The general syntax for a control definition, and the meaning of each parameter is as follows:

control [*text*,] *id*, *x*, *y*, *width*, *height* [, *style* [, *extended-style*]]

Horizontal dialog units are 1/4 of the dialog base width unit. Vertical units are 1/8 of the dialog base height unit. The current dialog base units are computed from the height and width of the current system font. The **GetDialogBaseUnits** function returns the dialog base units in pixels. The coordinates are relative to the origin of the dialog box.

Parameters

control

Keyword that indicates the type of control being defined, such as **PUSHBUTTON** or **CHECKBOX**.

text

Specifies text that is displayed with the control. The text is positioned within the control's specified dimensions or adjacent to the control.

This parameter must contain zero or more characters enclosed in double quotation marks (**"**). Strings are automatically null-terminated and converted to Unicode in the resulting resource file.

By default, the characters listed between the double quotation marks are ANSI characters, and escape sequences are interpreted as byte escape sequences. If the string is preceded by the **L** prefix, the string is a wide-character string and escape sequences are interpreted as 2-byte escape sequences that specify Unicode characters. If a double quotation mark is required in the text, you must include the double quotation mark twice.

An ampersand (**&**) character in the text indicates that the following character is used as a mnemonic character for the control. When the control is displayed, the ampersand is not shown, but the mnemonic character is underlined. The user can choose the control by pressing the key corresponding to the underlined mnemonic character. To use the ampersand as a character in a string, insert two ampersands (**&&**).

id

Specifies the control identifier. This value must be a 16-bit unsigned integer in the range 0 through 65,535 or a simple arithmetic expression that evaluates to a value in that range.

x

Specifies the x-coordinate of the left side of the control relative to the left side of the dialog box. This value must be a 16-bit unsigned integer in the range 0 through 65,535. The coordinate is in dialog units and is relative to the origin of the dialog box, window, or control containing the specified control.

y

Specifies the y-coordinate of the top side of the control relative to the top of the dialog box. This value must be a 16-bit unsigned integer in the range 0 through 65,535. The coordinate is in dialog units relative to the origin of the dialog box, window, or control containing the specified control.

width

Specifies the width of the control. This value must be a 16-bit unsigned integer in the range 1 through 65,535. The width is in 1/4-character units.

height

Specifies the height of the control. This value must be a 16-bit unsigned integer in the range 1 through 65,535. The height is in 1/8-character units.

style
Specifies the control styles. Use the bitwise OR (|) operator to combine styles.

extended-style
Specifies extended (WS_EX_*xxx*) styles. You must specify a *style* to specify an *extended-style*. See also **EXSTYLE**.

Occasionally, a statement will use a parameter differently, or may ignore a parameter. The statement-specific variation is described with the statement in the reference section.

Common Resource Attributes

All resource-definition statements include a *load-mem* option that specifies the loading and memory characteristics of the resource. These attributes are divided into two groups: load attributes and memory attributes. The only attribute that is used by a Win32-based application is the **DISCARDABLE** attribute. The remaining attributes are allowed in the script for backward compatibility with existing scripts, but are ignored.

Load Attributes

The load attributes specify when the resource is to be loaded. The load parameter must be one of the following:

PRELOAD
Ignored. In 16-bit Windows, the resource is loaded with the executable file.

LOADONCALL
Ignored. In 16-bit Windows, the resource is loaded when called.

Memory Attributes

The memory attributes specify whether the resource is fixed or movable, whether it is discardable, and whether it is pure. The memory parameter can be one or more of the following:

FIXED
Ignored. In 16-bit Windows, the resource remains at a fixed memory location.

MOVEABLE
Ignored. In 16-bit Windows, the resource can be moved if necessary in order to compact memory.

DISCARDABLE
Resource can be discarded if no longer needed.

PURE
Ignored. Accepted for compatibility with existing resource scripts.

IMPURE
Ignored. Accepted for compatibility with existing resource scripts.

The default for cursor, icon, and font resources is **DISCARDABLE**.

Single-Line Statements

A single-line statement can begin with any of the following keywords:

Keyword	Description
BITMAP	Defines a bitmap by naming it and specifying the name of the file that contains it. (To use a particular bitmap, the application requests it by name.)
CURSOR	Defines a cursor or animated cursor by naming it and specifying the name of the file that contains it. (To use a particular cursor, the application requests it by name.)
FONT	Specifies the name of a file that contains a font.
ICON	Defines an icon or animated icon by naming it and specifying the name of the file that contains it. (To use a particular icon, the application requests it by name.)
LANGUAGE	Sets the language for all resources up to the next **LANGUAGE** statement or to the end of the file. When the **LANGUAGE** statement appears before the **BEGIN** in an **ACCELERATORS**, **DIALOG**, **MENU**, **RCDATA**, or **STRINGTABLE** resource definition, the specified language applies only to that resource.
MESSAGETABLE	Defines a message table by naming it and specifying the name of the file that contains it. The file is a binary resource file generated by the Message Compiler.

Multiline Statements

A multiline statement can begin with any of the following keywords:

Keyword	Description
ACCELERATORS	Defines menu accelerator keys.
DIALOG	Defines a template that an application can use to create dialog boxes.
MENU	Defines the appearance and function of a menu.
RCDATA	Defines data resources. Data resources let you include binary data in the executable file.
STRINGTABLE	Defines string resources. String resources are Unicode strings that can be loaded from the executable file.

Each of these multiline statements allows optional statements before the **BEGIN** ... **END** block that defines the resource. You can specify zero or more of the following statements:

Statement	Description
CHARACTERISTICS *dword*	User-defined information about the resource that can be used by tools that read and write resource files. The value appears in the compiled resource file. However, it is not stored in the executable file and is not used by Windows.
LANGUAGE *language*, *sublanguage*	Specifies the language for the resource. The parameters are constants from WINNT.H.
VERSION *dword*	User-defined version number for the resource that can be used by tools that read and write resource files. The value appears in the compiled resource file. However, it is not stored in the executable file and is not used by Windows.

Sample Resource-Definition File

The following example shows a script file that defines the resources for an application named Shapes:

```
#include "SHAPES.H"

ShapesCursor  CURSOR  SHAPES.CUR
ShapesIcon    ICON    SHAPES.ICO

ShapesMenu    MENU
    BEGIN
        POPUP "&Shape"
            BEGIN
                MENUITEM "&Clear", ID_CLEAR
                MENUITEM "&Rectangle", ID_RECT
                MENUITEM "&Triangle", ID_TRIANGLE
                MENUITEM "&Star", ID_STAR
                MENUITEM "&Ellipse", ID_ELLIPSE
            END
    END
```

The **CURSOR** statement names the application's cursor resource ShapesCursor and specifies the cursor file SHAPES.CUR, which contains the image for that cursor.

The **ICON** statement names the application's icon resource ShapesIcon and specifies the icon file SHAPES.ICO, which contains the image for that icon.

The **MENU** statement defines an application menu named ShapesMenu, a pop-up menu with five menu items.

The menu definition, enclosed by the **BEGIN** and **END** keywords, specifies each menu item and the menu identifier that is returned when the user selects that item. For example, the first item on the menu, Clear, returns the menu identifier ID_CLEAR when the user selects it. The menu identifiers are defined in the application header file, SHAPES.H.

Using RC (The RC Command Line)

To start RC, use the **RC** command. The following line shows RC command-line syntax:

RC [*options*] *script-file*

The **RC** command's *options* parameter can include one or more of the following options:

/?
Displays a list of **RC** command-line options.

/d
Defines a symbol for the preprocessor that you can test with the **#ifdef** directive.

/fo*resname*
Uses *resname* for the name of the .RES file.

/h
Displays a list of **RC** command-line options.

/i
Searches the specified directory before searching the directories specified by the INCLUDE environment variable.

/l*codepage*
Specifies default language for compilation. For example, -l409 is equivalent to including the following statement at the top of the resource script file:

```
LANGUAGE LANG_ENGLISH,SUBLANG_ENGLISH_US
```

/r
Ignored. Provided for compatibility with existing makefiles.

/u
Undefines a symbol for the preprocessor.

/v
Displays messages that report on the progress of the compiler.

/x
Prevents RC from checking the INCLUDE environment variable when searching for header files or resource files.

Options are not case sensitive, and a hyphen (**-**) can be used in place of a slash mark (**/**). You can combine single-letter options if they do not require any additional parameters. For example, the following two commands are equivalent:

```
RC /V /X SAMPLE.RC
rc -vx sample.rc
```

The *script-file* parameter specifies the name of the resource-definition file that contains the names, types, filenames, and descriptions of the resources to be compiled.

Defining Names for the Preprocessor

You can specify conditional compilation in a script, based on whether a name is defined on the RC command line with the **/d** option, or in the file or an include file with the **#define** directive.

For example, suppose your application has a pop-up menu, the Debug menu, that should appear only with debugging versions of the application. When you compile the application for normal use, the Debug menu is not included. The following example shows the statements that can be added to the resource-definition file to define the Debug menu:

```
MainMenu MENU
BEGIN
  . . .
#ifdef DEBUG
        POPUP "&Debug"
        BEGIN
                MENUITEM "&Memory usage", ID_MEMORY
                MENUITEM "&Walk data heap", ID_WALK_HEAP
        END
#endif
END
```

When compiling resources for a debugging version of the application, you could include the Debug menu by using the following RC command:

```
rc -d DEBUG myapp.rc
```

To compile resources for a normal version of the application—one that does not include the Debug menu—you could use the following RC command:

```
rc myapp.rc
```

Renaming the Compiled Resource File

By default, when compiling resources, RC names the compiled resource (.RES) file with the base name of the .RC file and places it in the same directory as the .RC file. CVTRES must then be invoked to convert the .RES file to a binary resource (.RBJ) format which can be understood by the linker. The following example compiles MYAPP.RC and creates a compiled resource file named MYAPP.RES in the same directory as MYAPP.RC:

```
rc myapp.rc
```

The **/fo** option gives the resulting .RES file a name that differs from the name of the corresponding .RC file. For example, to name the resulting .RES file NEWFILE.RES, use the following command:

```
rc -fo newfile.res myapp.rc
```

The **/fo** option can also place the .RES file in a different directory. For example, the following command places the compiled resource file MYAPP.RES in the directory C:\SOURCE\RESOURCE:

```
rc -fo c:\source\resource\myapp.res myapp.rc
```

Searching for Files

By default, RC searches for header files and resource files (such as icon and cursor files) first in the current directory and then in the directories specified by the INCLUDE environment variable. (The PATH environment variable has no effect on which directories RC searches.)

Adding a Directory to Search

You can use the **/i** option to add a directory to the list of directories RC searches. The compiler then searches the directories in the following order:

1. The current directory
2. The directory or directories you specify by using the **/i** option, in the order in which they appear on the RC command line
3. The list of directories specified by the INCLUDE environment variable, in the order in which the variable lists them, unless you specify the **/x** option

The following example compiles the resource-definition file MYAPP.RC:

```
rc /i c:\source\stuff /i d:\resources myapp.rc
```

When compiling the script MYAPP.RC, RC searches for header files and resource files first in the current directory, then in C:\SOURCE\STUFF and D:\RESOURCES, and then in the directories specified by the INCLUDE environment variable.

Suppressing the INCLUDE Environment Variable

You can prevent RC from using the INCLUDE environment variable when determining the directories to search. To do so, use the **/x** option. The compiler then searches for files only in the current directory and in any directories you specify by using the **/i** option.

The following example compiles the script file MYAPP.RC:

```
rc /x /i c:\source\stuff myapp.rc
```

When compiling the script MYAPP.RC, RC searches for header files and resource files first in the current directory and then in C:\SOURCE\STUFF. It does not search the directories specified by the INCLUDE environment variable.

Displaying Progress Messages

By default, RC compiles quietly. It does not display messages that report on its progress. You can, however, specify that RC is to display these messages. To do so, use the **/v** option.

The following example causes RC to report on its progress as it compiles the resource-definition file SAMPLE.RC and creates the compiled resource file SAMPLE.RES:

```
rc /v sample.rc
```

Resource Compiler Reference

This section describes the resource-definition statements, preprocessor directives, and diagnostic messages used by the resource compiler.

Resource-Definition Statements Reference

This section describes the resource and control statements.

ACCELERATORS Resource Overview

Group

The **ACCELERATORS** statement defines one or more accelerators for an application. An accelerator is a keystroke defined by the application to give the user a quick way to perform a task. The **TranslateAccelerator** function is used to translate accelerator messages from the application queue into WM_COMMAND or WM_SYSCOMMAND messages.

Syntax

acctablename **ACCELERATORS**
[*optional-statements*]
BEGIN
event, *idvalue*, [*type*] [*options*]
. . .
END

Parameters

acctablename
Specifies either a unique name or a 16-bit unsigned integer value that identifies the resource.

optional-statements
Zero or more of the following statements:

Statement	Description
CHARACTERISTICS *dword*	User-defined information about a resource that can be used by tools that read and write resource files.
LANGUAGE *language*, *sublanguage*	Specifies the language for the resource. The parameters are constants from WINNT.H.
VERSION *dword*	User-defined version number for the resource that can be used by tools that read and write resource files.

event
Specifies the keystroke to be used as an accelerator. It can be any one of the following character types:

"*char*"
A single character enclosed in double quotation marks (**"**). The character can be preceded by a caret (^), meaning that the character is a control character.

character
An integer value representing a character. The *type* parameter must be **ASCII**.

virtual-key character
An integer value representing a virtual key. The virtual key for alphanumeric keys can be specified by placing the uppercase letter or number in double quotation marks (for example, "9" or "C"). The *type* parameter must be **VIRTKEY**.

idvalue
Specifies a 16-bit unsigned integer value that identifies the accelerator.

type
Required only when the *event* parameter is a *character* or a *virtual-key character*. The *type* parameter specifies either **ASCII** or **VIRTKEY**; the integer value of *event* is interpreted accordingly.

When **VIRTKEY** is specified and *event* contains a string, *event* must be uppercase.

options

Specifies the options that define the accelerator. This parameter can be one or more of the following values:

NOINVERT

Specifies that no top-level menu item is highlighted when the accelerator is used. This is useful when defining accelerators for actions such as scrolling that do not correspond to a menu item. If **NOINVERT** is omitted, a top-level menu item will be highlighted (if possible) when the accelerator is used.

ALT

Causes the accelerator to be activated only if the ALT key is down.

SHIFT

Causes the accelerator to be activated only if the SHIFT key is down.

CONTROL

Defines the character as a control character (the accelerator is only activated if the CONTROL key is down). This has the same effect as using a caret (^) before the accelerator character in the *event* parameter.

The **ALT**, **SHIFT**, and **CONTROL** options apply only to virtual keys.

Example

The following example demonstrates the use of accelerator keys:

```
1 ACCELERATORS
BEGIN
  "^C",   IDDCLEAR         ; control C
  "K",    IDDCLEAR         ; shift K
  "k",    IDDELLIPSE, ALT  ; alt k
  98,     IDDRECT, ASCII   ; b
  66,     IDDSTAR, ASCII   ; B (shift b)
  "g",    IDDRECT          ; g
  "G",    IDDSTAR          ; G (shift G)
  VK_F1, IDDCLEAR, VIRTKEY                ; F1
  VK_F1, IDDSTAR, CONTROL, VIRTKEY        ; control F1
  VK_F1, IDDELLIPSE, SHIFT, VIRTKEY       ; shift F1
  VK_F1, IDDRECT, ALT, VIRTKEY            ; alt F1
  VK_F2, IDDCLEAR, ALT, SHIFT, VIRTKEY    ; alt shift F2
  VK_F2, IDDSTAR, CONTROL, SHIFT, VIRTKEY ; ctrl shift F2
  VK_F2, IDDRECT, ALT, CONTROL, VIRTKEY   ; alt control F2
END
```

See Also

TranslateAccelerator
CHARACTERISTICS, **DIALOG**, **LANGUAGE**, **MENU**, **RCDATA**, **STRINGTABLE**, **VERSION**

AUTO3STATE Control

The **AUTO3STATE** statement creates an automatic 3-state check box. The control is an open box with the given text positioned to the right of the box. When chosen, the box automatically advances between three states: checked, unchecked, and disabled (grayed). The control sends a message to its parent whenever the user chooses the control.

Syntax

AUTO3STATE *text*, *id*, *x*, *y*, *width*, *height* [, *style* [, *extended-style*]]

Parameters

style

Specifies styles for the control, which can be a combination of the BS_AUTO3STATE style and the following styles: WS_TABSTOP, WS_DISABLED, and WS_GROUP.

The default style for this control is BS_AUTO3STATE and WS_TABSTOP.

For more information on the *text*, *id*, *x*, *y*, *width*, *height*, *style*, and *extended-style* parameters, see Common Control Parameters.

See Also

AUTOCHECKBOX, **CHECKBOX**, **CONTROL**, **STATE3**

AUTOCHECKBOX Control

The **AUTOCHECKBOX** statement creates an automatic check box control. The control is a small rectangle (check box) that has the specified text displayed next to it (typically, to the right). When the user chooses the control, the control highlights the rectangle and sends a message to its parent window. The **AUTOCHECKBOX** statement, which can only be used in the body of a **DIALOG** statement, defines the text, identifier, dimensions, and attributes of the control.

Syntax

AUTOCHECKBOX *text*, *id*, *x*, *y*, *width*, *height* [, *style* [, *extended-style*]]

Parameters

style

Specifies the styles of the control. This value can be a combination of the button class style BS_AUTOCHECKBOX and the WS_TABSTOP and WS_GROUP styles.

If you do not specify a style, the default style is BS_AUTOCHECKBOX and WS_TABSTOP.

For more information on the *text*, *id*, *x*, *y*, *width*, *height*, *style*, and *extended-style* parameters, see Common Control Parameters.

See Also

AUTO3STATE, **CHECKBOX**, **CONTROL**, **STATE3**

AUTORADIOBUTTON Control

The **AUTORADIOBUTTON** control statement specifies an automatic radio button control. This control automatically performs mutual exclusion with the other **AUTORADIOBUTTON** controls in the same group. When the button is chosen, the application is notified with BN_CLICKED.

Syntax

AUTORADIOBUTTON *text*, *id*, *x*, *y*, *width*, *height* [, *style* [, *extended-style*]]

Parameters

text

The specified text appears next to the radio button.

style

Specifies styles for the automatic radio button, which can be a combination of BUTTON-class styles and the following styles: WS_TABSTOP, WS_DISABLED, and WS_GROUP.

The default style for **AUTORADIOBUTTON** is BS_AUTORADIOBUTTON and WS_TABSTOP.

For more information on the *text*, *id*, *x*, *y*, *width*, *height*, *style*, and *extended-style* parameters, see Common Control Parameters.

See Also

CONTROL, **RADIOBUTTON**

BITMAP Resource Overview

Group

The **BITMAP** resource-definition statement specifies a bitmap that an application uses in its screen display or as an item in a menu or control.

Syntax

nameID **BITMAP** [*load-mem*] *filename*

Parameters

nameID

Specifies either a unique name or a 16-bit unsigned integer value identifying the resource.

load-mem

Specifies loading and memory attributes for the resource. For more information, see Common Resource Attributes.

filename

Specifies the name of the file that contains the resource. The name must be a valid filename; it must be a full path if the file is not in the current working directory. The path can either be a quoted or nonquoted string.

Example

The following example specifies two bitmap resources:

```
disk1   BITMAP disk.bmp
12      BITMAP PRELOAD diskette.bmp
```

See Also

LoadBitmap

CAPTION Statement

The **CAPTION** statement defines the title for a dialog box. The title appears in the box's caption bar (if it has one).

The default caption is empty.

Syntax

CAPTION "*captiontext***"**

Parameter

captiontext

Specifies a character string enclosed in double quotation marks (**"**).

Example

The following example demonstrates the use of the **CAPTION** statement:

```
CAPTION "Error!"
```

CHARACTERISTICS Statement

The **CHARACTERISTICS** statement allows the developer to specify information about a resource that can be used by tools that read and write resource-definition files. The specified *dword* value appears with the resource in the compiled .RES file. However, the value is not stored in the executable file and has no significance to Windows.

The **CHARACTERISTICS** statement appears before the **BEGIN** in an **ACCELERATORS**, **DIALOG**, **MENU**, **RCDATA**, or **STRINGTABLE** resource definition. The specified value applies only to that resource.

Syntax

CHARACTERISTICS *dword*

Parameter

dword

A user-defined doubleword value.

See Also

ACCELERATORS, **DIALOG**, **LANGUAGE**, **MENU**, **RCDATA**, **STRINGTABLE**

CHECKBOX Control

The **CHECKBOX** statement creates a check box control. The control is a small rectangle (check box) that has the specified text displayed next to it (typically, to the right). When the user selects the control, the control highlights the rectangle and sends a message to its parent window. The **CHECKBOX** statement, which can only be used in a **DIALOG** statement, defines the text, identifier, dimensions, and attributes of the control.

Syntax

CHECKBOX *text*, *id*, *x*, *y*, *width*, *height* [, *style* [, *extended-style*]]

Parameters

text

Specifies text that is displayed to the right of the control.

style

Specifies the control styles. This value can be a combination of the button class style BS_CHECKBOX and the WS_TABSTOP and WS_GROUP styles.

If you do not specify a style, the default style is BS_CHECKBOX and WS_TABSTOP.

For more information on the *text*, *id*, *x*, *y*, *width*, *height*, *style*, and *extended-style* parameters, see Common Control Parameters.

Example

This example creates a check-box control that is labeled "Italic":

```
CHECKBOX "Italic", 3, 10, 10, 40, 10
```

See Also

AUTOCHECKBOX, **AUTO3STATE**, **GetDialogBaseUnits**, **STATE3**

CLASS Statement

The **CLASS** statement defines the class of the dialog box. The **CLASS** statement appears in the optional section before a **DIALOG** statement's **BEGIN** keyword. If no class is given, the Windows standard dialog class is used.

Syntax

CLASS *class*

Parameters

class

Specifies a 16-bit unsigned integer or a string, enclosed in double quotation marks (**"**), that identifies the class of the dialog box. If the window procedure for the class does not process a message sent to it, it must call the **DefDlgProc** function to ensure that all messages are handled properly for the dialog box. A private class can use **DefDlgProc** as the default window procedure. The class must be registered with the **cbWndExtra** member of the **WNDCLASS** structure set to DLGWINDOWEXTRA.

Remarks

The **CLASS** statement should only be used with special cases, because it overrides the normal processing of a dialog box. The **CLASS** statement converts a dialog box to a window of the specified class; depending on the class, this could give undesirable results. Do not use the redefined control-class names with this statement.

Example

The following example demonstrates the use of the **CLASS** statement:

```
CLASS "myclass"
```

See Also

DefDlgProc, **DIALOG**

COMBOBOX Control

The **COMBOBOX** statement creates a combination box control (a combo box). A combo box consists of either a static text box or an edit box combined with a list box. The list box can be displayed at all times or pulled down by the user. If the combo box contains a static text box, the text box always displays the selection (if any) in the list box portion of the combo box. If it uses an edit box, the user can type in the desired selection; the list box highlights the first item (if any) that matches what the user has entered in the edit box. The user can then select the item highlighted in the list box to complete the choice. In addition, the combo box can be owner-drawn and of fixed or variable height.

Syntax

COMBOBOX *text*, *id*, *x*, *y*, *width*, *height* [, *style* [, *extended-style*]]

Parameters

style

Specifies the control styles. This value can be a combination of the COMBOBOX class styles and any of the following styles: WS_TABSTOP, WS_GROUP, WS_VSCROLL, and WS_DISABLED.

If you do not specify a style, the default style is CBS_SIMPLE and WS_TABSTOP.

For more information on the *text*, *id*, *x*, *y*, *width*, *height*, *style*, and *extended-style* parameters, see Common Control Parameters.

Example

This example creates a combo-box control with a vertical scroll bar:

```
COMBOBOX 777, 10, 10, 50, 54, CBS_SIMPLE | WS_VSCROLL | WS_TABSTOP
```

CONTROL: General Control

This statement defines a user-defined control window.

Syntax

CONTROL *text*, *id*, *class*, *style*, *x*, *y*, *width*, *height* [, *extended-style*]

Parameters

class

Specifies a redefined name, character string, or a 16-bit unsigned integer value that defines the class. This can be any one of the control classes; for a list of the control classes, see the first list following this description. If the value is a redefined name supplied by the application, it must be a string enclosed in double quotation marks (**"**).

style

Specifies a redefined name or integer value that specifies the style of the given control. The exact meaning of *style* depends on the *class* value. The sections following this description show the control classes and corresponding styles.

For more information on the *text*, *id*, *x*, *y*, *width*, *height*, *style*, and *extended-style* parameters, see Common Control Parameters.

The six possible control classes are described in the following sections.

The Button Control Class

A button control is a small rectangular child window that represents a "button" that the user can turn on or off by clicking it with the mouse. Button controls can be used alone or in groups, and can either be labeled or appear without text. Button controls typically change appearance when the user clicks them.

A button can have only one of the following styles, with the exception of BS_LEFTTEXT, which can be combined with check boxes and radio buttons:

BS_3STATE
: Creates a button that is the same as a check box, except that the box can be grayed (dimmed) as well as checked. The grayed state is used to show that the state of the check box is not determined.

BS_AUTO3STATE
: Creates a button that is the same as a three-state check box, except that the box changes its state when the user selects it. The state cycles through checked, grayed, and normal.

BS_AUTOCHECKBOX
: Creates a button that is the same as a check box, except that an X appears in the check box when the user selects the box; the X disappears (is cleared) the next time the user selects the box.

BS_AUTORADIOBUTTON
: Creates a button that is the same as a radio button, except that when the user selects it, the button automatically highlights itself and clears (removes the selection from) any other buttons in the same group.

BS_CHECKBOX
: Creates a small square that has text displayed to its right (unless this style is combined with the BS_LEFTTEXT style).

BS_DEFPUSHBUTTON
: Creates a button that has a heavy black border. The user can select this button by pressing the ENTER key. This style is useful for enabling the user to quickly select the most likely option (the default option).

BS_GROUPBOX
: Creates a rectangle in which other controls can be grouped. Any text associated with this style is displayed in the rectangle's upper-left corner.

BS_LEFTTEXT
: Places text on the left side of the radio button or check box when combined with a radio button or check box style.

BS_OWNERDRAW
: Creates an owner-drawn button. The owner window receives a WM_MEASUREITEM message when the button is created, and it receives a WM_DRAWITEM message when a visual aspect of the button has changed. The BS_OWNERDRAW style cannot be combined with any other button styles.

BS_PUSHBUTTON
: Creates a push button that posts a WM_COMMAND message to the owner window when the user selects the button.

BS_RADIOBUTTON
: Creates a small circle that has text displayed to its right (unless this style is combined with the BS_LEFTTEXT style). Radio buttons are usually used in groups of related but mutually exclusive choices.

The Combobox Control Class

Combo-box controls consist of a selection field similar to an edit control plus a list box. The list box may be displayed at all times or may be dropped down when the user selects a "pop box" next to the selection field.

Depending on the style of the combo box, the user can or cannot edit the contents of the selection field. If the list box is visible, typing characters into the selection box will cause the first list-box entry that matches the characters typed to be highlighted. Conversely, selecting an item in the list box displays the selected text in the selection field. Combo-box control styles are described below.

CBS_SIMPLE
: Displays the list box at all times. The current selection in the list box is displayed in the edit control.

CBS_DROPDOWN
: Similar to CBS_SIMPLE except that the list box is not displayed unless the user selects an icon next to the selection field.

CBS_DROPDOWNLIST
: Similar to CBS_DROPDOWN except that the edit control is replaced by a static text item which displays the current selection in the list box.

CBS_OWNERDRAWFIXED
: Specifies a fixed-height owner-draw combo box. The owner of the list box is responsible for drawing its contents; the items in the list box are all the same height.

CBS_OWNERDRAWVARIABLE
: Specifies a variable-height owner-draw combo box. The owner of the list box is responsible for drawing its contents; the items in the list box can have different heights.

CBS_AUTOHSCROLL
: Scrolls the text in the edit control to the right when the user types a character at the end of the line. If this style is not set, only text that fits within the rectangular boundary is allowed.

CBS_SORT
: Sorts strings entered into the list box.

CBS_HASSTRINGS
: Specifies an owner-draw combo box that contains items consisting of strings. The combo box maintains the memory and pointers for the strings so that the application can use the LB_GETTEXT message to retrieve the text for a particular item.

CBS_OEMCONVERT
: Converts text entered in the combo box edit control from the ANSI character set to the OEM character set and then back to ANSI. This ensures proper character conversion when the application calls the **CharToOem** function to convert an ANSI string in the combo box to OEM characters. This style is most useful for combo boxes that contain filenames and applies only to combo boxes created with the CBS_SIMPLE or CBS_DROPDOWN styles.

The Edit Control Class

An edit control is a rectangular child window in which the user can enter text from the keyboard. The user selects the control, and gives it the input focus, by clicking the mouse inside it or pressing the TAB key. The user can enter text when the control displays a flashing insertion point. The mouse can be used to move the cursor and select characters to be replaced, or to position the cursor for inserting characters. The BACKSPACE key can be used to delete characters.

Edit controls use the fixed-pitch font and display Unicode characters. They expand tab characters into as many space characters as are required to move the cursor to the next tab stop. Tab stops are assumed to be at every eighth character position. Edit control styles are described below.

ES_LEFT
Justifies the text to the left.

ES_CENTER
Centers the text. This style is valid in multiline edit controls only.

ES_RIGHT
Justifies the text to the right. This style is valid in multiline edit controls only.

ES_LOWERCASE
Converts all characters to lowercase as they are typed into the edit control.

ES_UPPERCASE
Converts all characters to uppercase as they are typed into the edit control.

ES_PASSWORD
Displays all characters as an asterisk (*) as they are typed into the edit control. An application can use the EM_SETPASSWORDCHAR message to change the character that is displayed.

ES_MULTILINE
Multiple-line edit control. (The default is single-line.) Shows as many lines of text as possible. The following four styles specify options for horizontal and vertical scrolling.

ES_AUTOVSCROLL specified
Shows as many lines as possible and scrolls vertically when the user presses ENTER. (This is actually the carriage-return character, which the edit control expands to a carriage-return - linefeed combination.)

ES_AUTOVSCROLL not specified
Shows as many lines as possible and beeps if the user presses ENTER when no more lines can be displayed.

ES_AUTOHSCROLL specified
Scrolls horizontally when the insertion point goes past the right edge of the control. To start a new line, press ENTER.

ES_AUTOHSCROLL not specified
Wraps words to the beginning of the next line when necessary. A new line is also started if the user presses ENTER. If the window size changes, the word-wrap position changes, and the text is redisplayed.

Multiple-line edit controls can have scroll bars. An edit control with scroll bars processes its own scroll-bar messages. Edit controls without scroll bars scroll as described above and process any scroll messages sent by the parent window.

ES_AUTOVSCROLL
Scrolls text automatically up one page when the user presses ENTER on the last line.

ES_AUTOHSCROLL
Scrolls text automatically to the right by 10 characters when the user types a character at the end of the line. When the user presses ENTER, the control scrolls all text back to position 0.

ES_NOHIDESEL
Overrides the default action, in which an edit control hides the selection when the control loses the

input focus. Inverts the selection instead.

ES_OEMCONVERT

Converts text entered in the edit control from the ANSI character set to the OEM character set and then back to ANSI. This ensures proper character conversion when the application calls the **CharToOem** function to convert an ANSI string in the edit control to OEM characters. This style is most useful for edit controls that contain filenames.

The Listbox Control Class

Listbox controls consist of a list of character strings. The control is used whenever an application needs to present a list of names, such as filenames, that the user can view and select. The user can select a string by pointing to the string with the mouse and clicking a mouse button. When a string is selected, it is highlighted and a notification message is passed to the parent window. A scroll bar can be used with a listbox control to scroll lists that are too long or too wide for the control window. Listbox control styles are described below.

LBS_STANDARD
: Strings in the list box are sorted alphabetically and the parent window receives an input message whenever the user clicks or double-clicks a string. The list box contains borders on all sides.

LBS_DISABLENOSCROLL
: Shows a disabled vertical scroll bar for the list box when the box does not contain enough items to scroll. If this style is not specified, the scroll bar is hidden when the list box does not contain enough items.

LBS_EXTENDEDSEL
: The user can select multiple items using the mouse with the SHIFT and/or the CONTROL key or special key combinations.

LBS_HASSTRINGS
: An owner-draw list box contains items consisting of strings. The list box maintains the memory and pointers for the strings so the application can use the LB_GETTEXT message to retrieve the text for a particular item.

LBS_NOTIFY
: The parent receives an input message whenever the user clicks or double-clicks a string.

LBS_MULTIPLESEL
: The string selection is toggled each time the user clicks or double-clicks the string. Any number of strings can be selected.

LBS_MULTICOLUMN
: The list box contains multiple columns. The list box can be scrolled horizontally. The LB_SETCOLUMNWIDTH message sets the width of the columns.

LBS_NOINTEGRALHEIGHT
: The size of the list box is exactly the size specified by the application when it created the list box. Normally, Windows sizes a list box so that the list box does not display partial items.

LBS_SORT
: The strings in the list box are sorted alphabetically.

LBS_NOREDRAW
: The list-box display is not updated when changes are made. This style can be changed at any time by sending a WM_SETREDRAW message.

LBS_OWNERDRAWFIXED
: The owner of the list box is responsible for drawing its contents; the items in the list box are all the same height.

LBS_OWNERDRAWVARIABLE
: The owner of the list box is responsible for drawing its contents; the items in the list box are variable in height.

LBS_USETABSTOPS
: The list box is able to recognize and expand tab characters when drawing its strings. The default tab positions are set at every 32 dialog units. (A dialog unit is a horizontal or vertical distance. One horizontal dialog unit is equal to 1/4 of the current dialog base width unit. The dialog base units are computed from the height and width of the current system font. The **GetDialogBaseUnits** function returns the size of the dialog base units in pixels.)

LBS_WANTKEYBOARDINPUT

The owner of the list box receives WM_VKEYTOITEM or WM_CHARTOITEM messages whenever the user presses a key while the list box has input focus. This allows an application to perform special processing on the keyboard input.

The Scroll-Bar Control Class

A scroll-bar control is a rectangle that contains a scroll thumb and has direction arrows at both ends. The scroll bar sends a notification message to its parent whenever the user clicks the mouse in the control. The parent is responsible for updating the thumb position, if necessary. Scroll-bar controls have the same appearance and function as the scroll bars used in ordinary windows. But unlike scroll bars, scroll-bar controls can be positioned anywhere within a window and used whenever needed to provide scrolling input for a window. Scroll-bar control styles are described below.

SBS_VERT
: Vertical scroll bar. If neither SBS_RIGHTALIGN nor SBS_LEFTALIGN is specified, the scroll bar has the height, width, and position given in the **CreateWindow** function.

SBS_RIGHTALIGN
: Used with SBS_VERT. The right edge of the scroll bar is aligned with the right edge of the rectangle specified by the *x*, *y*, *width*, and *height* values given in the **CreateWindow** function. The scroll bar has the default width for system scroll bars.

SBS_LEFTALIGN
: Used with SBS_VERT. The left edge of the scroll bar is aligned with the left edge of the rectangle specified by the *x*, *y*, *width*, and *height* values given in the **CreateWindow** function. The scroll bar has the default width for system scroll bars.

SBS_HORZ
: Horizontal scroll bar. If neither SBS_BOTTOMALIGN nor SBS_TOPALIGN is specified, the scroll bar has the height, width, and position given in the **CreateWindow** function.

SBS_TOPALIGN
: Used with SBS_HORZ. The top edge of the scroll bar is aligned with the top edge of the rectangle specified by the *x*, *y*, *width*, and *height* values given in the **CreateWindow** function. The scroll bar has the default height for system scroll bars.

SBS_BOTTOMALIGN
: Used with SBS_HORZ. The bottom edge of the scroll bar is aligned with the bottom edge of the rectangle specified by the *x*, *y*, *width*, and *height* values given in the **CreateWindow** function. The scroll bar has the default height for system scroll bars.

SBS_SIZEBOX
: Size box. If neither SBS_SIZEBOXBOTTOMRIGHTALIGN nor SBS_SIZEBOXTOPLEFTALIGN is specified, the size box has the height, width, and position given in the **CreateWindow** function.

SBS_SIZEBOXTOPLEFTALIGN
: Used with SBS_SIZEBOX. The top-left corner of the size box is aligned with the top-left corner of the rectangle specified by the *x*, *y*, *width*, and *height* values given in the **CreateWindow** function. The size box has the default size for system size boxes.

SBS_SIZEBOXBOTTOMRIGHTALIGN
: Used with SBS_SIZEBOX. The bottom-right corner of the size box is aligned with the bottom-right corner of the rectangle specified by the *x*, *y*, *width*, and *height* values given in the **CreateWindow** function. The size box has the default size for system size boxes.

The Static Control Class

Static controls are simple text fields, boxes, and rectangles that can be used to label, box, or separate other controls. Static controls take no input and provide no output. Static control styles are described below.

SS_LEFT
: A simple rectangle displaying the given text flush left. The text is formatted before it is displayed. Words that would extend past the end of a line are automatically wrapped to the beginning of the next line.

SS_CENTER
: A simple rectangle displaying the given text centered. The text is formatted before it is displayed. Words that would extend past the end of a line are automatically wrapped to the beginning of the next line.

SS_RIGHT
: A simple rectangle displaying the given text flush right. The text is formatted before it is displayed. Words that would extend past the end of a line are automatically wrapped to the beginning of the next line.

SS_LEFTNOWORDWRAP
: A simple rectangle displaying the given text flush left. Tabs are expanded, but words are not wrapped. Text that extends past the end of a line is clipped.

SS_SIMPLE
: A simple rectangle with a single line of text flush left. The line of text cannot be shortened or altered in any way. (The control's parent window or dialog box must not process the WM_CTLCOLOR message.)

SS_NOPREFIX
: Removes any ampersand (**&**) characters and underlines the next character in the string. Unless this style is specified, Windows will interpret any ampersand characters in the control's text to be accelerator prefix characters. If a static control is to contain text where this feature is not wanted, SS_NOPREFIX may be added. This static-control style may be included with any of the defined static controls.

  You can combine SS_NOPREFIX with other styles by using the bitwise OR (**|**) operator. This is most often used when filenames or other strings that may contain an ampersand need to be displayed in a static control in a dialog box.

SS_ICON
: An icon displayed in the dialog box. The given text is the name of an icon (not a filename) defined elsewhere in the resource file. For the **ICON** statement, the *width* and *height* parameters in the **CreateWindow** function are ignored; the icon automatically sizes itself.

SS_BLACKRECT
: A rectangle filled with the color used to draw window frames. This color is black in the default Windows color scheme.

SS_GRAYRECT
: A rectangle filled with the color used to fill the screen background. This color is gray in the default Windows color scheme.

SS_WHITERECT
: A rectangle filled with the color used to fill window backgrounds. This color is white in the default Windows color scheme.

SS_BLACKFRAME
: Box with a frame drawn with the same color as window frames. This color is black in the default Windows color scheme.

SS_GRAYFRAME
: Box with a frame drawn with the same color as the screen background (desktop). This color is gray

in the default Windows color scheme.

SS_WHITEFRAME
Box with a frame drawn with the same color as window backgrounds. This color is white in the default Windows color scheme.

SS_USERITEM
User-defined item.

CTEXT Control

The **CTEXT** statement creates a centered-text control. The control is a simple rectangle displaying the given text centered in the rectangle. The text is formatted before it is displayed. Words that would extend past the end of a line are automatically wrapped to the beginning of the next line. The **CTEXT** statement, which you can use only in a **DIALOG** statement, defines the text, identifier, dimensions, and attributes of the control.

Syntax

CTEXT *text*, *id*, *x*, *y*, *width*, *height* [, *style* [, *extended-style*]]

Parameters

text

Specifies text that is centered in the rectangular area of the control.

style

Specifies the control styles. This value can be any combination of the following styles: SS_CENTER, WS_TABSTOP, and WS_GROUP.

If you do not specify a style, the default style is SS_CENTER and WS_GROUP.

For more information on the *text*, *id*, *x*, *y*, *width*, *height*, *style*, and *extended-style* parameters, see Common Control Parameters.

Example

This example creates a centered-text control that is labeled "filename":

```
CTEXT "filename", 101, 10, 10, 100, 100
```

See Also

CONTROL, **DIALOG**, **LTEXT**, **RTEXT**

CURSOR Resource Overview

Group

The **CURSOR** statement specifies a bitmap that defines the shape of the cursor on the display screen or an animated cursor.

Syntax

nameID **CURSOR** [*load-mem*] *filename*

Parameters

nameID

Specifies either a unique name or a 16-bit unsigned integer identifying the resource.

load-mem

Specifies loading and memory attributes for the resource. For more information, see Common Resource Attributes.

filename

Specifies the name of the file that contains the resource. The name must be a valid filename; it must be a full path if the file is not in the current working directory. The path can either be a quoted or nonquoted string.

Remarks

Icon and cursor resources can contain more than one image. If the resource is marked with the **PRELOAD** option, Windows loads all images in the resource when the application executes.

Example

The following example specifies two cursor resources; one by name (cursor1) and the other by number (2):

```
cursor1 CURSOR bullseye.cur
2       CURSOR "d:\\cursor\\arrow.cur"
```

DEFPUSHBUTTON Control

The **DEFPUSHBUTTON** statement creates a default push-button control. The control is a small rectangle with a bold outline that represents the default response for the user. The given text is displayed inside the button. The control highlights the button in the usual way when the user clicks the mouse in it and sends a message to its parent window.

Syntax

DEFPUSHBUTTON *text*, *id*, *x*, *y*, *width*, *height* [, *style* [, *extended-style*]]

Parameters

text

Specifies text that is centered in the rectangular area of the control.

style

Specifies the control styles. This value can be a combination of the following styles: BS_DEFPUSHBUTTON, WS_TABSTOP, WS_GROUP, and WS_DISABLED.

If you do not specify a style, the default style is BS_DEFPUSHBUTTON and WS_TABSTOP.

For more information on the *text*, *id*, *x*, *y*, *width*, *height*, *style*, and *extended-style* parameters, see Common Control Parameters.

Example

This example creates a default push-button control that is labeled "Cancel":

```
DEFPUSHBUTTON "Cancel", 101, 10, 10, 24, 50
```

See Also

PUSHBUTTON, **RADIOBUTTON**

DIALOG Resource Overview

Group

The **DIALOG** statement defines a window that an application can use to create dialog boxes. The statement defines the position and dimensions of the dialog box on the screen as well as the dialog box style.

Syntax

nameID **DIALOG** [*load-mem*]] *x*, *y*, *width*, *height*
[[*optional-statements*]]
BEGIN
control-statement
. . .
END

Parameters

nameID
Identifies the dialog box. This is either a unique name or a unique 16-bit unsigned integer value in the range 1 to 65,535.

load-mem
Specifies loading and memory attributes for the resource. For more information, see Common Resource Attributes.

option-statements
Specifies options for the dialog box. This can be zero or more of the following statements:

CAPTION "*text***"**
Specifies the caption of the dialog box if it has a title bar. See **CAPTION** for more information.

CHARACTERISTICS *dword*
Specifies a user-defined double-word value for use by resource tools. This value is not used by Windows. For more information, see **CHARACTERISTICS**.

CLASS *class*
Specifies a 16-bit unsigned integer or a string, enclosed in double quotation marks (**"**), that identifies the class of the dialog box. See **CLASS** for more information.

LANGUAGE *language,sublanguage*
Specifies the language of the dialog box. See **LANGUAGE** for more information.

STYLE *styles*
Specifies the styles of the dialog box. See **STYLE** for more information.

EXSTYLE=*extended-styles*
Specifies the extended styles of the dialog box. See **EXSTYLE** for more information.

VERSION *dword*
Specifies a user-defined doubleword value. This statement is intended for use by additional resource tools and is not used by Windows. For more information, see **VERSION**.

For more information on the *x*, *y*, *width*, and *height* parameters, see Common Control Parameters.

Remarks

The **GetDialogBaseUnits** function returns the dialog base units in pixels. The exact meaning of the coordinates depends on the style defined by the **STYLE** option statement. For child-style dialog boxes, the coordinates are relative to the origin of the parent window, unless the dialog box has the style DS_ABSALIGN; in that case, the coordinates are relative to the origin of the display screen.

Do not use the WS_CHILD style with a modal dialog box. The **DialogBox** function always disables the parent/owner of the newly created dialog box. When a parent window is disabled, its child windows are

implicitly disabled. Since the parent window of the child-style dialog box is disabled, the child-style dialog box is too.

If a dialog box has the DS_ABSALIGN style, the dialog coordinates for its upper-left corner are relative to the screen origin instead of to the upper-left corner of the parent window. You would typically use this style when you wanted the dialog box to start in a specific part of the display no matter where the parent window may be on the screen.

The name **DIALOG** can also be used as the class-name parameter to the **CreateWindow** function to create a window with dialog box attributes.

Example

The following demonstrates the usage of the **DIALOG** statement:

```
#include <windows.h>

ErrorDialog DIALOG  10, 10, 300, 110
STYLE WS_POPUP|WS_BORDER
CAPTION "Error!"
BEGIN
    CTEXT "Select One:", 1, 10, 10, 280, 12
    PUSHBUTTON "&Retry", 2, 75, 30, 60, 12
    PUSHBUTTON "&Abort", 3, 75, 50, 60, 12
    PUSHBUTTON "&Ignore", 4, 75, 80, 60, 12
END
```

See Also

CONTROL, **CreateDialog**, **CreateWindow**, **DialogBox**, **DIALOGEX**, **GetDialogBaseUnits**, **ACCELERATORS**, **CHARACTERISTICS**, **LANGUAGE**, **MENU**, **RCDATA**, **STRINGTABLE**, **VERSION**

DIALOGEX Resource

The **DIALOGEX** resource is an extension of the **DIALOG** resource. In addition to the functionality offered by **DIALOG**, **DIALOGEX** allows for the following:

- Help IDs on the dialog itself as well as on controls within the dialog.
- Use of the **EXSTYLE** statement for the dialog itself as well as on controls within the dialog.
- Font weight and italic settings for the font to be used in the dialog.
- Control-specific data for controls within the dialog.
- Use of the BEDIT, IEDIT, and HEDIT predefined system class names.

Syntax

nameID **DIALOGEX** [[*load-mem*]] *x*, *y*, *width*, *height* [[, *helpID*]]
[[*optional-statements*]]
BEGIN
control-statement
. . .
END

Parameters

nameID
Identifies the dialog box. This is either a unique name or a unique 16-bit unsigned integer value in the range 1 to 65,535.

load-mem
Specifies loading and memory attributes for the resource. For more information, see Common Resource Attributes.

x
Specifies the location on the screen of the left side of the dialog, in dialog units.

y
Specifies the location on the screen of the top of the dialog, in dialog units.

width
Specifies the width of the dialog, in dialog units.

height
Specifies the height of the dialog, in dialog units.

helpID
Specifies a numeric expression indicating the ID used to identify the dialog during WM_HELP processing.

optional-statements
Specifies options for the dialog box. This can be zero or more of the following statements:

CAPTION *"text"*
Specifies the caption of the dialog box if it has a title bar. See **CAPTION** for more information.

CHARACTERISTICS *DWORD*
Specifies a user-defined DWORD value for use by resource tools. This value is not used by Windows. For more information see **CHARACTERISTICS**.

CLASS *class*
Specifies a 16-bit unsigned integer or a string, enclosed in double quotation marks (**"**), that identifies the class of the dialog box. See **CLASS** for more information.

EXSTYLE=*extended-styles*
Specifies the extended styles of the dialog box. See **EXSTYLE** for more information.

FONT *pointsize, typeface, weight, italic*

pointsize
Specifies the size, in points, of the font.

typeface
Specifies the name of the typeface. This name must be identical to the name defined in the [FONTS] section of WIN.INI. This parameter must be enclosed in double quotation marks (**"**).

weight
Specifies a numeric expression for the font weight (explicit FW_* values defined in WINGDI.H can be used by adding an include to the RC file: **#include** "WINGDI.H")

italic
Indicates whether the font should be italic or not. Specify either TRUE or FALSE for the *italic* value.

LANGUAGE *language, sublanguage*
Specifies the language of the dialog box. See **LANGUAGE** for more information.

MENU *menuname*
Specifies the menu to use. This value is either the name of the menu or the integer identifier.

STYLE *styles*
Specifies the styles of the dialog box. See **STYLE** for more information.

VERSION *DWORD*
Specifies a user-defined **DWORD** value. This statement is intended for use by additional resource tools and is not used by Windows. For more information, see **VERSION**.

The **DIALOGEX** body is marked with a **BEGIN** statement at the beginning of the body and an **END** statement at the end of the body. The body is made up of any number of control statements. There are four families of control statements: generic, static, button, and edit. While each of these families uses a different syntax for defining specific features of its controls, they all share a common syntax for defining the position, size, extended styles, help identification number, and control-specific data. That common syntax is:

(controlType), *x*, *y*, *cx*, *cy* **[**, **[***exStyle***]** **[**, *helpID***]**
[BEGIN
data-element-1 **[**,
data-element-2 **[**,
...]]
END]

(controlType)
Family-specific definition – described below.

x
Specifies the location in the dialog of the left side of the control, in dialog units.

y
Specifies the location in the dialog of the top of the control, in dialog units.

cx
Specifies the width of the control, in dialog units.

cy
Specifies the height of the control, in dialog units.

exStyle
Specifies any number of extended window styles (explicit WS_EX_* style values defined in winuser.h can be used by adding an include to the RC file: #include "winuser.h")

helpID
Specifies a numeric expression indicating the ID used to identify the control during WM_HELP processing.

controlData
Marked by a nested **BEGIN** and **END**, specifies the control-specific data for the control. When a

dialog is created, and a control in that dialog which has control-specific data is created, a pointer to that data is passed into the control's window procedure via the *lParam* to the WM_CREATE message for that control.

Following is the family-specific syntax of the (*controlType*):

Generic control:

CONTROL *controlText*, *id*, *className*, *style*

controlText
Specifies the window text for the control. For more information, see Common Control Parameters.

id
Specifies the control identifier. For more information, see Common Control Parameters.

className
Specifies the name of the class. This may be either a string enclosed in double quotation marks (**"**) or one of the following predefined system classes: BUTTON, STATIC, EDIT, LISTBOX, SCROLLBAR, or COMBOBOX.

style
Specifies the window styles (explicit WS_*, BS_*, SS_*, ES_*, LBS_*, SBS_*, and CBS_* style values defined in WINUSER.H can be used by adding an include to the RC file: **#include** "WINUSER.H")

Static control:

staticClass *controlText*, *id*

staticClass {LTEXT | RTEXT | CTEXT}

controlText
Specifies the window text for the control. For more information, see Common Control Parameters.

id
Specifies the control identifier. For more information, see Common Control Parameters.

Button control:

buttonClass *controlText*, *id*

buttonClass {AUTO3STATE | AUTOCHECKBOX | AUTORADIOBUTTON | CHECKBOX | PUSHBOX | PUSHBUTTON | RADIOBUTTON | STATE3 | USERBUTTON}

controlText
Specifies the window text for the control. For more information, see Common Control Parameters.

id
Specifies the control identifier. For more information, see Common Control Parameters.

Edit control:

editClass *id*

editClass {EDITTEXT | BEDIT | HEDIT | IEDIT}

id
Specifies the control identifier. For more information, see Common Control Parameters.

Arithmetic and Boolean Operations

Valid operations that may be contained in any of the numeric expressions in the statements of **DIALOGEX** are:

- Add ('+')
- Subtract ('-')

- Unary minus ('-')
- Unary NOT ('~')
- AND ('&')
- OR ('|')

See Also

CONTROL, **CreateDialog**, **CreateWindow**, **DialogBox**, **DIALOGEX**, **GetDialogBaseUnits**, **ACCELERATORS**, **CHARACTERISTICS**, **LANGUAGE**, **MENU**, **RCDATA**, **STRINGTABLE**, **VERSION**

EDITTEXT Control

The **EDITTEXT** statement defines an EDIT control belonging to the EDIT class. It creates a rectangular region in which the user can type and edit text. The control displays a cursor when the user clicks the mouse in it. The user can then use the keyboard to enter text or edit the existing text. Editing keys include the BACKSPACE and DELETE keys. The user can also use the mouse to select characters to be deleted or to select the place to insert new characters.

Syntax

EDITTEXT *text*, *id*, *x*, *y*, *width*, *height* [[, *style* [[, *extended-style*]]]]

Parameters

style

Specifies the control styles. This value can be a combination of the edit class styles and the following styles: WS_TABSTOP, WS_GROUP, WS_VSCROLL, WS_HSCROLL, and WS_DISABLED.

If you do not specify a style, the default style is ES_LEFT, WS_BORDER, and WS_TABSTOP.

For more information on the *text*, *id*, *x*, *y*, *width*, *height*, *style*, and *extended-style* parameters, see Common Control Parameters.

Example

The following example demonstrates the use of the **EDITTEXT** statement:

```
EDITTEXT  3, 10, 10, 100, 10
```

EXSTYLE Statement

The EXSTYLE command allows you to designate a dialog box with the WS_EX_*style* styles. In a resource definition, the **EXSTYLE** statement is placed with the optional statements before the **BEGIN** keyword.

Syntax

EXSTYLE *extended-style*

Parameter

extended-style

The WS_EX_*style* style for the dialog box or control.

Remarks

It can also be placed with the *load-mem* parameters in a **DIALOG** statement as follows:

name **DIALOG** [[*load-mem*]] **EXSTYLE=***extended-style x*, *y*, ...

For controls, extended styles are specified after the *style* option in any control definition, as follows:

control text, *id*, *x*, *y*, *width*, *height*, [[*style*[[, *extended-style*]]]]

See also

ACCELERATORS, **CONTROL**, **DIALOG**, **MENU**, **POPUP**, **RCDATA**, **STRINGTABLE**, User-Defined Resource

FONT Resource Overview

Group

The **FONT** resource-definition statement specifies a file that contains a font.

For a font resource, *nameID* must be a number; it cannot be a name.

Syntax

nameID **FONT** [[*load-mem*]] *filename*

Parameters

nameID

Specifies either a unique name or a 16-bit unsigned integer value identifying the resource.

load-mem

Specifies loading and memory attributes for the resource. For more information, see Common Resource Attributes.

filename

Specifies the name of the file that contains the resource. The name must be a valid filename; it must be a full path if the file is not in the current working directory. The path can either be a quoted or nonquoted string.

Example

The following example specifies a single font resource:

```
5 FONT   CMROMAN.FNT
```

FONT Statement

The **FONT** statement defines the font with which Windows will draw text in the dialog box. The font must have been previously loaded, either from the WIN.INI file or by calling the **LoadResource** function.

Syntax

FONT *pointsize*, *typeface*

Parameters

pointsize

Specifies the size, in points, of the font.

typeface

Specifies the name of the typeface. This name must be identical to the name defined in the [FONTS] section of WIN.INI. This parameter must be enclosed in double quotes (**"**).

Example

The following example demonstrates the use of the **FONT** statement:

```
FONT 12, "MS Shell Dlg"
```

See Also

DIALOG, **LoadResource**

GROUPBOX Control

The **GROUPBOX** statement creates a group box control. The control is a rectangle that groups other controls together. The controls are grouped by drawing a border around them and displaying the given text in the upper-left corner. The **GROUPBOX** statement, which you can use only in a **DIALOG** statement, defines the text, identifier, dimensions, and attributes of a control window.

When the style contains WS_TABSTOP or the text specifies an accelerator, tabbing or pressing the accelerator key moves the focus to the first control within the group.

Syntax

GROUPBOX *text*, *id*, *x*, *y*, *width*, *height* [[, *style* [[, *extended-style*]]]]

Parameters

style

Specifies the control styles. This value can be a combination of the button class style BS_GROUPBOX and the WS_TABSTOP and WS_DISABLED styles.

If you do not specify a style, the default style is BS_GROUPBOX.

For more information on the *text*, *id*, *x*, *y*, *width*, *height*, *style*, and *extended-style* parameters, see Common Control Parameters.

Example

This example creates a group-box control that is labeled "Options":

```
GROUPBOX "Options", 101, 10, 10, 100, 100
```

See Also

DIALOG

ICON Resource Overview

Group

The **ICON** resource-definition statement specifies a bitmap that defines the shape of the icon to be used for a given application or an animated icon.

Syntax

nameID **ICON** [[*load-mem*]] *filename*

Parameters

nameID

Specifies either a unique name or a 16-bit unsigned integer value identifying the resource.

load-mem

Specifies loading and memory attributes for the resource. For more information, see Common Resource Attributes.

filename

Specifies the name of the file that contains the resource. The name must be a valid filename; it must be a full path if the file is not in the current working directory. The path can either be a quoted or nonquoted string.

Remarks

Icon and cursor resources can contain more than one image. If the resource is marked as **PRELOAD**, Windows loads all images in the resource when the application executes.

Example

The following example specifies two icon resources:

```
desk1   ICON desk.ico
11      ICON DISCARDABLE custom.ico
```

ICON Control

The **ICON** statement creates an icon control. This control is an icon displayed in a dialog box. The **ICON** control statement, which you can use only in a **DIALOG** statement, defines the icon-resource identifier, icon-control identifier, position, and attributes of a control.

Syntax

ICON *text*, *id*, *x*, *y*, [[*width*, *height*, *style* [[, *extended-style*]]]]

Parameters

text
Specifies the name of an icon (not a filename) defined elsewhere in the resource file.

width
This value is ignored and should be set to zero.

height
This value is ignored and should be set to zero.

style
Specifies the control style. This parameter is optional. The only value that can be specified is the SS_ICON style. This is the default style whether this parameter is specified or not.

For more information on the *text*, *id*, *x*, *y*, *width*, *height*, *style*, and *extended-style* parameters, see Common Control Parameters.

Example

This example creates an icon control whose icon identifier is 901 and whose name is "myicon":

```
ICON "myicon" 901, 30, 30
```

See Also

DIALOG, **ICON**

LANGUAGE Statement Overview

Group

The **LANGUAGE** statement sets the language for all resources up to the next **LANGUAGE** statement or to the end of the file. When the **LANGUAGE** statement appears before the **BEGIN** in an **ACCELERATORS**, **DIALOG**, **MENU**, **RCDATA**, or **STRINGTABLE** resource definition, the specified language applies only to that resource.

Syntax

LANGUAGE *language*, *sublanguage*

Parameters

language

Language identifier. Must be one of the constants from WINNT.H

sublanguage

Sublanguage identifier. Must be one of the constants from WINNT.H

See Also

ACCELERATORS, **CHARACTERISTICS**, **DIALOG**, **MENU**, **RCDATA**, **STRINGTABLE**, **VERSION**

LISTBOX Control

The **LISTBOX** statement creates commonly used controls for a dialog box or window. The control is a rectangle containing a list of strings (such as filenames) from which the user can select. The **LISTBOX** statement, which can only be used in a **DIALOG** or **WINDOW** statement, defines the identifier, dimensions, and attributes of a control window.

Syntax

LISTBOX *id*, *x*, *y*, *width*, *height* [[, *style* [[, *extended-style*]]]]

Parameters

style

Specifies the control styles. This value can be a combination of the list-box class styles and any of the following styles: WS_BORDER and WS_VSCROLL.

If you do not specify a style, the default style is LBS_NOTIFY and WS_BORDER.

For more information on the *id*, *x*, *y*, *width*, *height*, *style*, and *extended-style* parameters, see Common Control Parameters.

Example

This example creates a list-box control whose identifier is 101:

```
LISTBOX 101, 10, 10, 100, 100
```

See Also

COMBOBOX, **DIALOG**

LTEXT Control

The **LTEXT** statement creates a left-aligned text control. The control is a simple rectangle displaying the given text left-aligned in the rectangle. The text is formatted before it is displayed. Words that would extend past the end of a line are automatically wrapped to the beginning of the next line. The **LTEXT** statement, which can be used only in a **DIALOG** statement, defines the text, identifier, dimensions, and attributes of the control.

Syntax

LTEXT *text*, *id*, *x*, *y*, *width*, *height* [[, *style* [[, *extended-style*]]]]

Parameters

style

Specifies the control styles. This value can be any combination of the BS_RADIOBUTTON style and the following styles: SS_LEFT, WS_TABSTOP, and WS_GROUP.

If you do not specify a style, the default style is SS_LEFT and WS_GROUP.

For more information on the *text*, *id*, *x*, *y*, *width*, *height*, *style*, and *extended-style* parameters, see Common Control Parameters.

Example

This example creates a left-aligned text control that is labeled "Filename":

```
LTEXT "Filename", 101, 10, 10, 100, 100
```

See Also

CONTROL, **CTEXT**, **DIALOG**, **RTEXT**

MENU Resource Overview

Group

The **MENU** statement defines the contents of a menu resource. A menu resource is a collection of information that defines the appearance and function of an application menu. A menu is a special input tool that lets a user select commands and open submenus from a list of menu items.

Syntax

menuID **MENU** [[*load-mem*]]
[[*optional-statements*]]
BEGIN
item-definitions
. . .
END

Parameters

menuID

A number that identifies the menu. This value is either a unique string or a unique 16-bit unsigned integer value in the range of 1 to 65,535.

load-mem

An attribute that specifies loading and memory attributes for the resource. For more information, see Common Resource Attributes.

optional-statements

This parameter can be zero of more of the following statements:

Statement	Description
CHARACTERISTICS *dword*	User-defined information about a resource that can be used by tools that read and write resource files.
LANGUAGE *language*, *sublanguage*	Specifies the language for the resource. The parameters are constants from WINNT.H.
VERSION *dword*	User-defined version number for the resource that can be used by tools that read and write resource files.

Example

Following is an example of a complete **MENU** statement:

```
sample MENU
BEGIN
     MENUITEM "&Soup", 100
     MENUITEM "S&alad", 101
     POPUP "&Entree"
     BEGIN
          MENUITEM "&Fish", 200
          MENUITEM "&Chicken", 201, CHECKED
          POPUP "&Beef"
          BEGIN
               MENUITEM "&Steak", 301
               MENUITEM "&Prime Rib", 302
          END
     END
     MENUITEM "&Dessert", 103
END
```

See Also

MENUEX, **MENUITEM**, **POPUP**, **ACCELERATORS**, **CHARACTERISTICS**, **DIALOG**, **LANGUAGE**, **RCDATA**, **STRINGTABLE**, **VERSION**

MENU Statement

The **MENU** statement defines the dialog box's menu. If no statement is given, the dialog box has no menu.

Syntax

MENU *menuname*

Parameter

menuname

Specifies the menu to use. This value is either the name of the menu or the integer identifier of the menu.

Example

The following example demonstrates the use of the **MENU** dialog statement:

```
MENU errmenu
```

MENUEX Resource

The **MENUEX** resource is an extension of the **MENU** resource. In addition to the functionality offered by **MENU**, **MENUEX** allows for the following:

- Help identifiers on menus.
- Identifiers on menus.
- Use of the MFT_* type flags and MFS_* state flags. For more information on these flags, see the **MENUITEMINFO** structure.

Syntax

menuID **MENUEX**
  BEGIN
    [{[**MENUITEM** *itemText* [, [*id*] [, [*type*] [, *state*]]]] |
    [**POPUP** *itemText* [, [*id*] [, [*type*] [, [*state*] [, *helpID*]]]]
    BEGIN
      popupBody
    END]} ...]
  END

Parameters

MENUITEM statement
  Defines a menu item.

  itemText
    A string containing the text for the menu item. For more information, see **MENUITEM**.

  id
    A numeric expression indicating the identifier of the menu item.

  type
    A numeric expression indicating the type of the menu item To use the predefined MFT_* type values, include the following statement in your .RC file:

```
#include "WINUSER.H"
```

  state
    A numeric expression indicating the state of the menu item To use the predefined MFS_* state values, include the following statement in your .RC file:

```
#include "WINUSER.H"
```

POPUP statement
  Defines a menu item that has a submenu associated with it.

  itemText
    A string containing the text for the menu item.

  id
    A numeric expression indicating the identifier of the menu item.

  type
    A numeric expression indicating the type of the menu item To use the predefined MFT_* type values, include the following statement in your .RC file:

```
#include "WINUSER.H"
```

  state
    A numeric expression indicating the state of the menu item To use the predefined MFS_* state values, include the following statement in your .RC file:

```
#include "WINUSER.H"
```

helpID
A numeric expression indicating the identifier used to identify the menu during WM_HELP processing.

popupBody
Contains any combination of the **MENUITEM** and **POPUP** statements.

Remarks

Valid arithmetic and Boolean operations that can be contained in any of the numeric expressions in the statements of **MENUEX** are:

- Add ('+')
- Subtract ('-')
- Unary minus ('-')
- Unary NOT ('~')
- AND ('&')
- OR ('|')

See Also

MENU, **MENUITEM**, **POPUP**, **ACCELERATORS**, **CHARACTERISTICS**, **DIALOG**, **LANGUAGE**, **RCDATA**, **STRINGTABLE**, **VERSION**

MENUITEM Statement

The **MENUITEM** statement defines a menu item.

Syntax

MENUITEM *text*, *result*, [[*optionlist*]]
MENUITEM SEPARATOR

Parameters

text
A string that indicates the name of the menu item.

The string can contain the escape characters **\t** and **\a**. The **\t** character inserts a tab in the string and is used to align text in columns. Tab characters should be used only in menus, not in menu bars. (For information on menus, see **POPUP Resource**.) The **\a** character aligns all text that follows it flush right to the menu bar or pop-up menu.

result
A number that specifies the result generated when the user selects the menu item. This parameter takes an integer value. Menu-item results are always integers; when the user clicks the menu-item name, the result is sent to the window that owns the menu.

optionlist
This parameter specifies the appearance of the menu item. This optional parameter takes one or more redefined menu options, separated by commas or spaces. The menu options are as follows:

CHECKED
Menu item has a check mark next to it.

GRAYED
Menu item is initially inactive and appears on the menu in gray or a lightened shade of the menu-text color.

HELP
Identifies a help item.

INACTIVE
Menu item is displayed but it cannot be selected.

MENUBARBREAK
Same as **MENUBREAK** except that for pop-up menus, it separates the new column from the old column with a vertical line.

MENUBREAK
Places the menu item on a new line for static menu-bar items. For menus, it places the menu item in a new column with no dividing line between the columns.

The **INACTIVE** and **GRAYED** options cannot be used together.

SEPARATOR
The **MENUITEM SEPARATOR** form of the **MENUITEM** statement creates an inactive menu item that serves as a dividing bar between two active menu items on a menu.

Example

The following example demonstrates the use of the **MENUITEM** and **MENUITEM SEPARATOR** statements:

```
MENUITEM "&Roman", 206, CHECKED, GRAYED
MENUITEM SEPARATOR
MENUITEM "&Blackletter", 301
```

See Also

MENU, **POPUP**

MESSAGETABLE Resource Overview

Group

The **MESSAGETABLE** resource-definition statement defines the ID and file of an application's message table resource. Message tables are special string resources used in event logging and with the **FormatMessage** function. The file contains a binary message table generated by the Message Compiler. The Message Compiler also generates a resource script file that contains the **MESSAGETABLE** statements you need to include the message table resources in the compiled resource file. Use the **#include** directive to include this resource script into your main resource script.

Syntax

nameID **MESSAGETABLE** *filename*

Parameters

nameID

Specifies either a unique name or a 16-bit unsigned integer value identifying the resource.

filename

Specifies the name of the file that contains the resource. The name must be a valid filename; it must be a full path if the file is not in the current working directory. The path can either be a quoted or nonquoted string.

See Also

Using the Message Compiler, **STRINGTABLE**

POPUP Resource

The **POPUP** statement defines a menu item that can contain menu items and submenus.

Syntax

POPUP text, [[*optionlist*]]

BEGIN
item-definitions
.
.
.
END

Parameters

text
A string that contains the name of the menu. This string must be enclosed in double quotation marks (**"**).

optionlist
This parameter specifies redefined menu options that specify the appearance of the menu item. This optional parameter can be one or more of the following.

CHECKED
Menu item has a check mark next to it. This option is not valid for a top-level menu.

GRAYED
Menu item is initially inactive and appears on the menu in gray or a lightened shade of the menu-text color.

INACTIVE
Menu item is displayed but it cannot be selected.

MENUBARBREAK
Same as **MENUBREAK** except that for menus, it separates the new column from the old column with a vertical line.

MENUBREAK
Places the menu item on a new line for static menu-bar items. For menus, it places the menu item in a new column with no dividing line between the columns.

The **INACTIVE** and **GRAYED** options cannot be used together.

Example

The following example demonstrates the use of the **POPUP** statement:

```
chem MENU
BEGIN
    POPUP "&Elements"
    BEGIN
         MENUITEM "&Oxygen", 200
         MENUITEM "&Carbon", 201, CHECKED
         MENUITEM "&Hydrogen", 202
         MENUITEM SEPARATOR
         MENUITEM "&Sulfur", 203
         MENUITEM "Ch&lorine", 204
    END
```

```
    POPUP "&Compounds"
    BEGIN
         POPUP "&Sugars"
         BEGIN
            MENUITEM "&Glucose", 301
            MENUITEM "&Sucrose", 302, CHECKED
            MENUITEM "&Lactose", 303, MENUBREAK
            MENUITEM "&Fructose", 304
         END
         POPUP "&Acids"
         BEGIN
              "&Hydrochloric", 401
              "&Sulfuric", 402
         END
    END
END
```

See Also

MENU, **MENUITEM**

PUSHBOX Control

The **PUSHBOX** statement creates a push-box control, which is identical to a **PUSHBUTTON**, except that it does not display a button face or frame; only the text appears.

Syntax

PUSHBOX *text*, *id*, *x*, *y*, *width*, *height* [[, *style* [[, *extended-style*]]]]

Parameters

style

Specifies styles for the pushbox, which can be a combination of the BS_PUSHBOX style and the following styles: WS_TABSTOP, WS_DISABLED, and WS_GROUP.

The default style is BS_PUSHBOX and WS_TABSTOP.

For more information on the *text*, *id*, *x*, *y*, *width*, *height*, *style*, and *extended-style* parameters, see Common Control Parameters.

See Also

PUSHBUTTON

PUSHBUTTON Control

The **PUSHBUTTON** statement creates a push-button control. The control is a round-cornered rectangle containing the given text. The text is centered in the control. The control sends a message to its parent whenever the user chooses the control.

Syntax

PUSHBUTTON *text*, *id*, *x*, *y*, *width*, *height* [[, *style* [[, *extended-style*]]]]

Parameters

style

Specifies styles for the pushbutton, which can be a combination of the BS_PUSHBUTTON style and the following styles: WS_TABSTOP, WS_DISABLED, and WS_GROUP.

The default style is BS_PUSHBUTTON and WS_TABSTOP.

For more information on the *text*, *id*, *x*, *y*, *width*, *height*, *style*, and *extended-style* parameters, see Common Control Parameters.

Example

The following example demonstrates the use of the **PUSHBUTTON** statement:

```
PUSHBUTTON "ON", 7, 10, 10, 20, 10
```

See Also

GetDialogBaseUnits

RADIOBUTTON Control

The **RADIOBUTTON** statement creates a radio-button control. The control is a small circle that has the given text displayed next to it, typically to its right. The control highlights the circle and sends a message to its parent window when the user selects the button. The control removes the highlight and sends a message when the button is next selected.

Syntax

RADIOBUTTON *text*, *id*, *x*, *y*, *width*, *height* [[, *style* [[, *extended-style*]]]]

Parameters

style

Specifies styles for the radio button, which can be a combination of BUTTON-class styles and the following styles: WS_TABSTOP, WS_DISABLED, and WS_GROUP.

The default style for **RADIOBUTTON** is BS_RADIOBUTTON and WS_TABSTOP.

For more information on the *text*, *id*, *x*, *y*, *width*, *height*, *style*, and *extended-style* parameters, see Common Control Parameters.

Example

The following example demonstrates the use of the **RADIOBUTTON** statement:

```
RADIOBUTTON "Italic", 100, 10, 10, 40, 10
```

See Also

GetDialogBaseUnits

RCDATA Resource Overview

Group

The **RCDATA** statement defines a raw data resource for an application. Raw data resources permit the inclusion of binary data directly in the executable file.

Syntax

nameID **RCDATA** [[*load-mem*]]
[[*optional-statements*]]
BEGIN
raw-data
. . .
END

Parameters

nameID

Specifies either a unique name or a 16-bit unsigned integer value that identifies the resource.

load-mem

Specifies loading and memory attributes for the resource. For more information, see Common Resource Attributes.

optional-statements

Zero or more of the following statements:

Statement	Description
CHARACTERISTICS *dword*	User-defined information about a resource that can be used by tools that read and write resource files.
LANGUAGE *language*, *sublanguage*	Specifies the language for the resource. The parameters are constants from WINNT.H.
VERSION *dword*	User-defined version number for the resource that can be used by tools that read and write resource files.

raw-data

Specifies raw data consisting of one or more integers or strings of characters. Integers can be specified in decimal, octal, or hexadecimal format. To be compatible with Windows 3.1, integers are stored as WORD values. You can store an integer as a DWORD value by qualifying the integer with the **L** suffix.

Strings are enclosed in quotation marks. RC does not automatically append a terminating null character to a string. Each string is a sequence of the specified ANSI characters, unless you qualify it as a wide-character string with the **L** prefix.

The block of data begins on a **DWORD** boundary and RC performs no padding or alignment of data within the *raw-data* block. It is the programmer's responsibility to ensure the proper alignment of data within the block.

Example

The following example demonstrates the use of the **RCDATA** statement:

```
resname RCDATA
BEGIN
   "Here is an ANSI string\0",    /* explicitly null-terminated */
   L"Here is a Unicode string\0", /* explicitly null-terminated */
   1024,                          /* integer, stored as WORD */
   7L,                            /* integer, stored as DWORD */
   0x029a,                        /* hex integer    */
   0o733,                         /* octal integer  */
END
```

See Also

ACCELERATORS, CHARACTERISTICS, DIALOG, LANGUAGE, MENU, STRINGTABLE, VERSION

RTEXT Control

The **RTEXT** statement creates a right-aligned text control. The control is a simple rectangle displaying the given text right-aligned in the rectangle. The text is formatted before it is displayed. Words that would extend past the end of a line are automatically wrapped to the beginning of the next line.

Syntax

RTEXT *text*, *id*, *x*, *y*, *width*, *height* [[, *style* [[, *extended-style*]]]]

Parameters

style

Specifies styles for the text control, which can be any combination of the following: WS_TABSTOP and WS_GROUP.

The default style for **RTEXT** is SS_RIGHT and WS_GROUP.

For more information on the *text*, *id*, *x*, *y*, *width*, *height*, *style*, and *extended-style* parameters, see Common Control Parameters.

Example

The following example demonstrates the use of the **RTEXT** statement:

```
RTEXT "Number of Messages", 4, 30, 50, 100, 10
```

See Also

CONTROL, **CTEXT**, **DIALOG**, **LTEXT**

SCROLLBAR Control

The **SCROLLBAR** statement creates a scroll-bar control. The control is a rectangle that contains a scroll box and has direction arrows at both ends. The scroll-bar control sends a notification message to its parent whenever the user clicks the mouse in the control. The parent is responsible for updating the scroll-box position. Scroll-bar controls can be positioned anywhere in a window and used whenever needed to provide scrolling input.

Syntax

SCROLLBAR *text*, *id*, *x*, *y*, *width*, *height* [[, *style* [[, *extended-style*]]]]

Parameters

style

Specifies a combination (or none) of the following styles: WS_TABSTOP, WS_GROUP, and WS_DISABLED.

In addition to these styles, the *style* parameter may contain a combination (or none) of the SCROLLBAR-class styles. The default style for **SCROLLBAR** is SBS_HORZ.

For more information on the *text*, *id*, *x*, *y*, *width*, *height*, *style*, and *extended-style* parameters, see Common Control Parameters.

Example

The following example demonstrates the use of the **SCROLLBAR** statement:

```
SCROLLBAR 999, 25, 30, 10, 100
```

STATE3 Control

The **STATE3** statement defines a three-state check box control. The control is identical to a **CHECKBOX**, except that it has three states: checked, unchecked, and disabled (grayed).

Syntax

STATE3 *text*, *id*, *x*, *y*, *width*, *height* [[, *style* [[, *extended-style*]]]]

Parameters

text

Specifies text that is displayed to the right of the control.

style

Specifies the control styles. This value can be a combination of the button class style BS_3STATE and the WS_TABSTOP and WS_GROUP styles.

If you do not specify a style, the default style is BS_3STATE and WS_TABSTOP.

For more information on the *text*, *id*, *x*, *y*, *width*, *height*, *style*, and *extended-style* parameters, see Common Control Parameters.

See Also

AUTOCHECKBOX, **CHECKBOX**, **CONTROL**

STRINGTABLE Resource Overview

Group

The **STRINGTABLE** statement defines one or more string resources for an application. String resources are simply null-terminated Unicode strings that can be loaded when needed from the executable file, using the **LoadString** function.

Syntax

STRINGTABLE [[*load-mem*]]
[[*optional-statements*]]
BEGIN
stringID string
. . .
END

Parameter

load-mem
Specifies loading and memory attributes for the resource. For more information, see Common Resource Attributes.

optional-statements
Zero or more of the following statements:

Statement	Description
CHARACTERISTICS *dword*	User-defined information about a resource that can be used by tools that read and write resource files.
LANGUAGE *language*, *sublanguage*	Specifies the language for the resource. The parameters are constants from WINNT.H.
VERSION *dword*	User-defined version number for the resource that can be used by tools that read and write resource files.

stringID
Specifies an unsigned 16-bit integer that identifies the resource.

string
Specifies one or more strings, enclosed in double quotation marks. The string must be no longer than 4097 characters and must occupy a single line in the source file. To add a carriage return to the string, use this character sequence: **\012**. For example, "Line one\012Line two" would define a string that would be displayed as follows:

```
Line one
Line two
```

Remarks

Grouping strings in separate sections allows all related strings to be read in at one time and discarded together. When possible, an application should make the table movable and discardable. RC allocates 16 strings per section and uses the identifier value to determine which section is to contain the string. Strings with the same upper-12 bits in their identifiers are placed in the same section.

Example

The following example demonstrates the use of the **STRINGTABLE** statement:

```
#define IDS_HELLO    1
#define IDS_GOODBYE  2

STRINGTABLE
BEGIN
    IDS_HELLO,   "Hello"
    IDS_GOODBYE, "Goodbye"
END
```

See Also

LoadString, **ACCELERATORS**, **CHARACTERISTICS**, **DIALOG**, **LANGUAGE**, **MENU**, **RCDATA**, **VERSION**

STYLE Statement

The **STYLE** statement defines the window style of the dialog box. The window style specifies whether the box is a pop-up or a child window. The default style has the following attributes: WS_POPUP, WS_BORDER, and WS_SYSMENU.

Syntax

STYLE *style*

Parameter

style

Specifies the window style. This parameter takes an integer value or redefined name. The following lists the redefined styles:

DS_LOCALEDIT
: Specifies that edit controls in the dialog box will use memory in the application's data section. By default, all edit controls in dialog boxes use memory outside the application's data section. This feature can be suppressed by adding the DS_LOCALEDIT flag to the **STYLE** command for the dialog box. If this flag is not used, EM_GETHANDLE and EM_SETHANDLE messages must not be used since the storage for the control is not in the application's data section. This feature does not affect edit controls created outside of dialog boxes.

DS_MODALFRAME
: Creates a dialog box with a modal dialog box frame that can be combined with a title bar and System menu by specifying the WS_CAPTION and WS_SYSMENU styles.

DS_NOIDLEMSG
: Suppresses WM_ENTERIDLE messages that Windows would otherwise send to the owner of the dialog box while the dialog box is displayed.

DS_SYSMODAL
: Creates a system-modal dialog box.

WS_BORDER
: Creates a window that has a border.

WS_CAPTION
: Creates a window that has a title bar (implies the WS_BORDER style).

WS_CHILD
: Creates a child window. It cannot be used with the WS_POPUP style.

WS_CHILDWINDOW
: Creates a child window that has the WS_CHILD style.

WS_CLIPCHILDREN
: Excludes the area occupied by child windows when drawing within the parent window. Used when creating the parent window.

WS_CLIPSIBLINGS
: Clips child windows relative to each other; that is, when a particular child window receives a WM_PAINT message, this style clips all other top-level child windows out of the region of the child window to be updated. (If the WS_CLIPSIBLINGS style is not given and child windows overlap, it is possible, when drawing in the client area of a child window, to draw in the client area of a neighboring child window.) For use with the WS_CHILD style only.

WS_DISABLED
: Creates a window that is initially disabled.

WS_DLGFRAME
: Creates a window with a modal dialog box frame but no title.

WS_GROUP
: Specifies the first control of a group of controls in which the user can move from one control to the

next by using the arrow keys. All controls defined with the WS_GROUP style after the first control belong to the same group. The next control with the WS_GROUP style ends the style group and starts the next group (that is, one group ends where the next begins). This style is valid only for controls.

WS_HSCROLL
Creates a window that has a horizontal scroll bar.

WS_ICONIC
Creates a window that is initially iconic. For use with the WS_OVERLAPPED style only.

WS_MAXIMIZE
Creates a window of maximum size.

WS_MAXIMIZEBOX
Creates a window that has a Maximize box.

WS_MINIMIZE
Creates a window of minimum size.

WS_MINIMIZEBOX
Creates a window that has a Minimize box.

WS_OVERLAPPED
Creates an overlapped window. An overlapped window has a caption and a border.

WS_OVERLAPPEDWINDOW
Creates an overlapped window having the WS_OVERLAPPED, WS_CAPTION, WS_SYSMENU, WS_THICKFRAME, WS_MINIMIZEBOX, and WS_MAXIMIZEBOX styles.

WS_POPUP
Creates a pop-up window. It cannot be used with the WS_CHILD style.

WS_POPUPWINDOW
Creates a pop-up window that has the WS_POPUP, WS_BORDER, and WS_SYSMENU styles. The WS_CAPTION style must be combined with the WS_POPUPWINDOW style to make the System menu visible.

WS_SIZEBOX
Creates a window that has a size box. Used only for windows with a title bar or with vertical and horizontal scroll bars.

WS_SYSMENU
Creates a window that has a System-menu box in its title bar. Used only for windows with title bars. If used with a child window, this style creates a Close box instead of a System-menu box.

WS_TABSTOP
Specifies one of any number of controls through which the user can move by using the TAB key. The TAB key moves the user to the next control specified by the WS_TABSTOP style. This style is valid only for controls.

WS_THICKFRAME
Creates a window with a thick frame that can be used to size the window.

WS_VISIBLE
Creates a window that is initially visible. This applies to overlapping and pop-up windows. For overlapping windows, the *y* parameter is used as a parameter for the **ShowWindow** function.

WS_VSCROLL
Creates a window that has a vertical scroll bar.

Remarks

If the redefined names are used, you must include WINDOWS.H.

User-Defined Resource

A user-defined resource statement specifies a resource that contains application-specific data. The data can have any format and can be defined either as the content of a given file (if the *filename* parameter is given) or as a series of numbers and strings (if the *raw-data* block is given).

Syntax

nameID typeID [[*load-mem*]] *filename*

The *filename* specifies the name of a file containing the binary data of the resource. The contents of the file are included as the resource. RC does not interpret the binary data in any way. It is the programmer's responsibility to ensure that the data is properly aligned for the target computer architecture.

A user-defined resource can also be defined completely in the resource script using the syntax:

nameID typeID [[*load-mem*]]
BEGIN
raw-data
END

Parameters

nameID
Specifies either a unique name or a 16-bit unsigned integer that identifies the resource.

typeID
Specifies either a unique name or a 16-bit unsigned integer that identifies the resource type. If a number is given, it must be greater than 255. The numbers 1 through 255 are reserved for existing and future redefined resource types.

load-mem
Specifies loading and memory attributes for the resource. For more information, see Common Resource Attributes.

filename
Specifies the name of the file that contains the resource data. The parameter must be a valid filename; it must be a full path if the file is not in the current working directory.

raw-data
Specifies raw data consisting of one or more integers or strings of characters. Integers can be specified in decimal, octal, or hexadecimal format. To be compatible with Windows 3.1, integers are stored as WORD values. You can store an integer as a DWORD value by qualifying the integer with the **L** suffix.

Strings are enclosed in quotation marks. RC does not automatically append a terminating null character to a string. Each string is a sequence of the specified ANSI characters, unless you qualify it as a wide-character string with the **L** prefix.

The block of data begins on a **DWORD** boundary and RC performs no padding or alignment of data within the *raw-data* block. It is the programmer's responsibility to ensure the proper alignment of data within the block.

Example

The following example shows several user-defined statements:

```
array   MYRES   data.res
14      300     custom.res
18 MYRES2
BEGIN
   "Here is an ANSI string\0",    /* explicitly null-terminated */
```

```
   L"Here is a Unicode string\0", /* explicitly null-terminated */
   1024,                          /* integer, stored as WORD */
   7L,                            /* integer, stored as DWORD */
   0x029a,                        /* hex integer    */
   0o733,                         /* octal integer  */
END
```

VERSION Statement

The **VERSION** statement allows the developer to specify version information about a resource that can be used by tools that read and write resource files. The specified *dword* value appears with the resource in the compiled .RES file. However, the value is not stored in the executable file and has no significance to Windows.

The **VERSION** statement appears before the **BEGIN** in an **ACCELERATORS**, **DIALOG**, **MENU**, **RCDATA**, or **STRINGTABLE** resource definition. The specified value applies only to that resource.

Syntax

VERSION *dword*

Parameter

dword

A user-defined doubleword value.

See Also

ACCELERATORS, **CHARACTERISTICS**, **DIALOG**, **LANGUAGE**, **MENU**, **RCDATA**, **STRINGTABLE**

VERSIONINFO Resource

The **VERSIONINFO** statement creates a version-information resource. The resource contains such information about the file as its version number, its intended operating system, and its original filename. The resource is intended to be used with the File Installation library functions.

Syntax

versionID **VERSIONINFO** *fixed-info*
BEGIN
block-statement
. . .
END

Parameters

versionID
Specifies the version-information resource identifier. This value must be 1.

fixed-info
Specifies the version information, such as the file version and the intended operating system. This parameter consists of the following statements:

FILEVERSION *version*
Specifies the binary version number for the file. The *version* consists of two 32-bit integers, defined by four 16-bit integers. For example, "FILEVERSION 3,10,0,61" is translated into two doublewords: 0x0003000a and 0x0000003d, in that order. Therefore, if *version* is defined by the doublewords dw1 and dw2, they need to appear in the **FILEVERSION** statement as follows: **HIWORD**(dw1), **LOWORD**(dw1), **HIWORD**(dw2), **LOWORD**(dw2).

PRODUCTVERSION *version*
Specifies the binary version number for the product with which the file is distributed. The *version* parameter is two 32-bit integers, defined by four 16-bit integers. For more information about *version*, see the **FILEVERSION** description.

FILEFLAGSMASK *fileflagsmask*
Specifies which bits in the **FILEFLAGS** statement are valid. If a bit is set, the corresponding bit in **FILEFLAGS** is valid.

FILEFLAGS *fileflags*
Specifies the Boolean attributes of the file. The *fileflags* parameter must be the combination of all the file flags that are valid at compile time. For Windows 3.1, this value is 0x3f.

FILEOS *fileos*
Specifies the operating system for which this file was designed. The *fileos* parameter can be one of the operating system values given in the Comments section.

FILETYPE *filetype*
Specifies the general type of file. The *filetype* parameter can be one of the file type values listed in the Comments section.

FILESUBTYPE *subtype*
Specifies the function of the file. The *subtype* parameter is zero unless the *type* parameter in the **FILETYPE** statement is **VFT_DRV**, **VFT_FONT**, or **VFT_VXD**. For a list of file subtype values, see the Comments section.

block-statement
Specifies one or more version-information blocks. A block can contain string information or variable information.

Remarks

To use the constants specified with the **VERSIONINFO** statement, the WINVER.H file (included in WINDOWS.H) must be included in the resource-definition file.

The following list describes the parameters used in the **VERSIONINFO** statement:

fileflags

Specifies a combination of the following values:

VS_FF_DEBUG
File contains debugging information or is compiled with debugging features enabled.

VS_FF_INFOINFERRED
File contains a dynamically created version-information resource. Some of the blocks for the resource may be empty or incorrect. This value is not intended to be used in version-information resources created by using the **VERSIONINFO** statement.

VS_FF_PATCHED
File has been modified and is not identical to the original shipping file of the same version number.

VS_FF_PRERELEASE
File is a development version, not a commercially released product.

VS_FF_PRIVATEBUILD
File was not built using standard release procedures. If this value is given, the **StringFileInfo** block must contain a **PrivateBuild** string.

VS_FF_SPECIALBUILD
File was built by the original company using standard release procedures but is a variation of the standard file of the same version number. If this value is given, the **StringFileInfo** block must contain a **SpecialBuild** string.

fileos

Specifies one of the following values:

Value	Description
VOS_UNKNOWN	Operating system for which the file was designed is unknown to Windows.
VOS_DOS	File was designed for MS-DOS.
VOS_NT	File was designed for Windows NT.
VOS_WINDOWS16	File was designed for Windows version 3.0 or later.
VOS_WINDOWS32	File was designed for 32-bit Windows.
VOS_DOS_WINDOWS16	File was designed for Windows version 3.0 or later running with MS-DOS.
VOS_DOS_WINDOWS32	File was designed for 32-bit Windows running with MS-DOS.
VOS_NT_WINDOWS32	File was designed for 32-bit Windows running with Windows NT.

The values 0x00002L, 0x00003L, 0x20000L and 0x30000L are reserved.

filetype

Specifies one of the following values:

Value	Description
VFT_UNKNOWN	File type is unknown to Windows.
VFT_APP	File contains an application.
VFT_DLL	File contains a dynamic-link library (DLL).
VFT_DRV	File contains a device driver. If the

	dwFileType member is **VFT_DRV**, the **dwFileSubtype** member contains a more specific description of the driver.
VFT_FONT	File contains a font. If the **dwFileType** member is **VFT_FONT**, the **dwFileSubtype** member contains a more specific description of the font.
VFT_VXD	File contains a virtual device.
VFT_STATIC_LIB	File contains a static-link library.

All other values are reserved for use by Microsoft.

subtype

Specifies additional information about the file type.

If the **FILETYPE** statement specifies **VFT_DRV**, this parameter can be one of the following values:

Value	Description
VFT2_UNKNOWN	Driver type is unknown to Windows.
VFT2_DRV_COMM	File contains a communications driver.
VFT2_DRV_PRINTER	File contains a printer driver.
VFT2_DRV_KEYBOARD	File contains a keyboard driver.
VFT2_DRV_LANGUAGE	File contains a language driver.
VFT2_DRV_DISPLAY	File contains a display driver.
VFT2_DRV_MOUSE	File contains a mouse driver.
VFT2_DRV_NETWORK	File contains a network driver.
VFT2_DRV_SYSTEM	File contains a system driver.
VFT2_DRV_INSTALLABLE	File contains an installable driver.
VFT2_DRV_SOUND	File contains a sound driver.

If the **FILETYPE** statement specifies **VFT_FONT**, this parameter can be one of the following values:

Value	Description
VFT2_UNKNOWN	Font type is unknown to Windows.
VFT2_FONT_RASTER	File contains a raster font.
VFT2_FONT_VECTOR	File contains a vector font.
VFT2_FONT_TRUETYPE	File contains a TrueType font.

If the **FILETYPE** statement specifies **VFT_VXD**, this parameter must be the virtual-device identifier included in the virtual-device control block.

All *subtype* values not listed here are reserved for use by Microsoft.

langID

Specifies one of the following language codes:

Code	Language	Code	Language

0x0401	Arabic	0x0415	Polish
0x0402	Bulgarian	0x0416	Brazilian Portuguese
0x0403	Catalan	0x0417	Rhaeto-Romanic
0x0404	Traditional Chinese	0x0418	Romanian
0x0405	Czech	0x0419	Russian
0x0406	Danish	0x041A	Croato-Serbian (Latin)
0x0407	German	0x041B	Slovak
0x0408	Greek	0x041C	Albanian
0x0409	U.S. English	0x041D	Swedish
0x040A	Castilian Spanish	0x041E	Thai
0x040B	Finnish	0x041F	Turkish
0x040C	French	0x0420	Urdu
0x040D	Hebrew	0x0421	Bahasa
0x040E	Hungarian	0x0804	Simplified Chinese
0x040F	Icelandic	0x0807	Swiss German
0x0410	Italian	0x0809	U.K. English
0x0411	Japanese	0x080A	Mexican Spanish
0x0412	Korean	0x080C	Belgian French
0x0413	Dutch	0x0C0C	Canadian French
0x0414	Norwegian - Bokml	0x100C	Swiss French
0x0810	Swiss Italian	0x0816	Portuguese
0x0813	Belgian Dutch	0x081A	Serbo-Croatian (Cyrillic)
0x0814	Norwegian - Nynorsk		

charsetID

Specifies one of the following character-set identifiers:

Identifier	**Character Set**
0	7-bit ASCII
932	Windows, Japan (Shift - JIS X-0208)
949	Windows, Korea (Shift - KSC 5601)
950	Windows, Taiwan (GB5)
1200	Unicode
1250	Windows, Latin-2 (Eastern European)
1251	Windows, Cyrillic
1252	Windows, Multilingual
1253	Windows, Greek
1254	Windows, Turkish
1255	Windows, Hebrew
1256	Windows, Arabic

string-name

Specifies one of the following redefined names:

Comments

Specifies additional information that should be displayed for diagnostic purposes.

CompanyName

Specifies the company that produced the file—for example, "Microsoft Corporation" or "Standard Microsystems Corporation, Inc." This string is required.

FileDescription

Specifies a file description to be presented to users. This string may be displayed in a list box when the user is choosing files to install—for example, "Keyboard Driver for AT-Style Keyboards" or "Microsoft Word for Windows". This string is required.

FileVersion

Specifies the version number of the file—for example, "3.10" or "5.00.RC2". This string is required.

InternalName

Specifies the internal name of the file, if one exists—for example, a module name if the file is a dynamic-link library. If the file has no internal name, this string should be the original filename, without extension. This string is required.

LegalCopyright

Specifies all copyright notices that apply to the file. This should include the full text of all notices, legal symbols, copyright dates, and so on—for example, "Copyright© Microsoft Corporation 1990-1992". This string is optional.

LegalTrademarks

Specifies all trademarks and registered trademarks that apply to the file. This should include the full text of all notices, legal symbols, trademark numbers, and so on—for example, "Windows™ is a trademark of Microsoft® Corporation". This string is optional.

OriginalFilename

Specifies the original name of the file, not including a path. This information enables an application to determine whether a file has been renamed by a user. The format of the name depends on the file system for which the file was created. This string is required.

PrivateBuild

Specifies information about a private version of the file—for example, "Built by TESTER1 on \TESTBED". This string should be present only if the **VS_FF_PRIVATEBUILD** flag is set in the **dwFileFlags** member of the **VS_FIXEDFILEINFO** structure of the root block.

ProductName

Specifies the name of the product with which the file is distributed—for example, "Microsoft Windows". This string is required.

ProductVersion

Specifies the version of the product with which the file is distributed—for example, "3.10" or "5.00.RC2". This string is required.

SpecialBuild

Specifies how this version of the file differs from the standard version—for example, "Private build for TESTER1 solving mouse problems on M250 and M250E computers". This string should be present only if the **VS_FF_SPECIALBUILD** flag is set in the **dwFileFlags** member of the **VS_FIXEDFILEINFO** structure in the root block.

A string information block has the following form:

```
BLOCK "StringFileInfo"
  BEGIN
    BLOCK "lang-charset"
    BEGIN
```

```
        VALUE "string-name", "value"
            . . .
    END
END
```

Following are the parameters in the **StringFileInfo** block:

lang-charset
Specifies a language and character-set identifier pair. It is a hexadecimal string consisting of the concatenation of the language and character-set identifiers listed earlier in this section.

string-name
Specifies the name of a value in the block and can be one of the redefined names listed earlier in this section.

value
Specifies, as a character string, the value of the corresponding string name. More than one **VALUE** statement can be given.

A variable information block has the following form:

```
BLOCK "VarFileInfo"
    BEGIN
        VALUE "Translation",
            langID, charsetID
                . . .
    END
```

Following are the parameters in the variable information block:

langID
Specifies one of the language identifiers listed earlier in this section.

charsetID
Specifies one of the character-set identifiers listed earlier in this section. More than one identifier pair can be given, but each pair must be separated from the preceding pair with a comma.

Preprocessor Directives Reference

The syntax and semantics for the RC preprocessor are the same as for the preprocessor in a C compiler. Single-line comments begin with two forward slashes (**//**) and run to the end of the line. Block comments begin with an opening delimiter (**/***) and run to a closing delimiter (***/**). Comments do not nest.

You use the **#define** directive to define symbols for your resource identifiers in a header file. You include this header both in the resource script and your application source code. Similarly, the values for attributes and styles are defined by including the C header files for Windows.

You can also define macros. The standard C preprocessing operators (#, ##) can be used in a macro definition. For detailed information on preprocessing and defining macros, see your C compiler documentation.

The rest of this section describes the preprocessing directives.

#define Overview

Group

The **#define** directive assigns the given value to the specified name. All subsequent occurrences of the name are replaced by the value.

Syntax

#define *name value*

Parameters

name

Specifies the name to be defined. This value is any combination of letters, digits, and punctuation.

value

Specifies any integer, character string, or line of text.

Example

This example assigns values to the names "NONZERO" and "USERCLASS":

```
#define     NONZERO     1
#define     USERCLASS   "MyControlClass"
```

See Also

#ifdef, **#ifndef**, **#undef**

#elif Overview

Group

The **#elif** directive marks an optional clause of a conditional-compilation block defined by a **#ifdef**, **#ifndef**, or **#if** directive. The directive controls conditional compilation of the resource file by checking the specified constant expression. If the constant expression is nonzero, **#elif** directs the compiler to continue processing statements up to the next **#endif**, **#else**, or **#elif** directive and then skip to the statement after **#endif**. If the constant expression is zero, **#elif** directs the compiler to skip to the next **#endif**, **#else**, or **#elif** directive. You can use any number of **#elif** directives in a conditional block.

Syntax

#elif *constant-expression*

Parameter

constant-expression

Specifies the expression to be checked. This value is a defined name, an integer constant, or an expression consisting of names, integers, and arithmetic and relational operators.

Example

In this example, **#elif** directs the compiler to process the second **BITMAP** statement only if the value assigned to the name "Version" is less than 7. The **#elif** directive itself is processed only if Version is greater than or equal to 3.

```
#if Version < 3
BITMAP 1 errbox.bmp
#elif Version < 7
BITMAP 1 userbox.bmp
#endif
```

See Also

#else, **#endif**, **#if**, **#ifdef**, **#ifndef**

#else Overview

Group

The **#else** directive marks an optional clause of a conditional-compilation block defined by a **#ifdef**, **#ifndef**, or **#if** directive. The **#else** directive must be the last directive before the **#endif** directive.

This directive has no arguments.

Syntax

#else

Example

This example compiles the second **BITMAP** statement only if the name "DEBUG" is not defined:

```
#ifdef DEBUG
    BITMAP 1 errbox.bmp
#else
    BITMAP 1 userbox.bmp
#endif
```

See Also

#elif, **#endif**, **#if**, **#ifdef**, **#ifndef**

#endif Overview

Group

The **#endif** directive marks the end of a conditional-compilation block defined by a **#ifdef** directive. One **#endif** is required for each **#if**, **#ifdef**, or **#ifndef** directive.

Syntax

#endif

This directive has no arguments.

See Also

#elif, **#else**, **#if**, **#ifdef**, **#ifndef**

#if Overview

Group

The **#if** directive controls conditional compilation of the resource file by checking the specified constant expression. If the constant expression is nonzero, **#if** directs the compiler to continue processing statements up to the next **#endif**, **#else**, or **#elif** directive and then skip to the statement after the **#endif** directive. If the constant expression is zero, **#if** directs the compiler to skip to the next **#endif**, **#else**, or **#elif** directive.

Syntax

#if *constant-expression*

Parameter

constant-expression

Specifies the expression to be checked. This value is a defined name, an integer constant, or an expression consisting of names, integers, and arithmetic and relational operators.

Example

This example compiles the **BITMAP** statement only if the value assigned to the name "Version" is less than 3:

```
#if Version < 3
BITMAP 1 errbox.bmp
#endif
```

See Also

#elif, **#else**, **#endif**, **#ifdef**, **#ifndef**

#ifdef Overview

Group

The **#ifdef** directive controls conditional compilation of the resource file by checking the specified name. If the name has been defined by using a **#define** directive or by using the **/d** command-line option with the resource compiler, **#ifdef** directs the compiler to continue with the statement immediately after the **#ifdef** directive. If the name has not been defined, **#ifdef** directs the compiler to skip all statements up to the next **#endif** directive.

Syntax

#ifdef *name*

Parameter

name

Specifies the name to be checked by the directive.

Example

This example compiles the **BITMAP** statement only if the name "Debug" is defined:

```
#ifdef Debug
BITMAP 1 errbox.bmp
#endif
```

See Also

#define, **#endif**, **#if**, **#ifndef**, **#undef**

#ifndef Overview

Group

The **#ifndef** directive controls conditional compilation of the resource file by checking the specified name. If the name has not been defined or if its definition has been removed by using the **#undef** directive, **#ifndef** directs the compiler to continue processing statements up to the next **#endif**, **#else**, or **#elif** directive and then skip to the statement after the **#endif** directive. If the name is defined, **#ifndef** directs the compiler to skip to the next **#endif**, **#else**, or **#elif** directive.

Syntax

#ifndef *name*

Parameter

name

Specifies the name to be checked by the directive.

Example

This example compiles the **BITMAP** statement only if the name "Optimize" is not defined:

```
#ifndef Optimize
BITMAP 1 errbox.bmp
#endif
```

See Also

#elif, **#else**, **#endif**, **#if**, **#ifdef**, **#undef**

#include Overview

Group

The **#include** directive causes resource compiler to process the file specified in the *filename* parameter. This file should be a header file that defines the constants used in the resource-definition file. Use the statement

```
#ifndef RC_INVOKED
```

in your header file to surround statements that can be compiled by a C compiler but not RC. This way, you can use the same include files in your .C and .RC files.

Syntax

#include *filename*

Parameter

filename

Specifies the name of the file to be included. If the file is in the current directory, the string must be enclosed in double quotation marks; if the file is in the directory specified by the INCLUDE environment variable, the string must be enclosed in less-than and greater-than characters (**<>**). You must give a full path enclosed in double quotation marks (**"**) if the file is not in the current directory or in the directory specified by the INCLUDE environment variable.

Example

This example processes the header files WINDOWS.H and HEADERS\MYDEFS.H while compiling the resource-definition file:

```
#include <windows.h>
#include "headers\mydefs.h"
```

See Also

#define

#undef Overview

Group

The **#undef** directive removes the current definition of the specified name. All subsequent occurrences of the name are processed without replacement.

Syntax

#undef *name*

Parameter

name

Specifies the name to be removed. This value is any combination of letters, digits, and punctuation.

Example

This example removes the definitions for the names "nonzero" and "USERCLASS":

```
#undef      nonzero
#undef      USERCLASS
```

See Also

#define

RC Diagnostic Messages Reference

This section describes of diagnostic messages produced by RC. Many of these messages appear when RC is not able to compile resources properly. The descriptions clarify the cause of each message. The messages are listed in alphabetic order.

A capital V in parentheses (V) at the beginning of a message description indicates that the message is displayed only if RC is run with the **/v** (verbose) option. These messages are generally informative and do not necessarily indicate errors.

A

Accelerator Type required (ASCII or VIRTKEY)
The *type* parameter in the **ACCELERATORS** statement must contain either the **ASCII** or **VIRTKEY** value.

B

BEGIN expected in Accelerator Table
An **ACCELERATORS** statement was not followed by the **BEGIN** keyword.

BEGIN expected in Dialog
A **DIALOG** statement was not followed by the **BEGIN** keyword.

BEGIN expected in menu
A **MENU** statement was not followed by the **BEGIN** keyword.

BEGIN expected in RCData
An **RCDATA** statement was not followed by the **BEGIN** keyword.

BEGIN expected in String Table
A **STRINGTABLE** statement was not followed by the **BEGIN** keyword.

BEGIN expected in VERSIONINFO resource
A **VERSIONINFO** statement was not followed by the **BEGIN** keyword.

Bitmap file *resource-file* **is not in** *version-number* **format.**
Use Microsoft Image Editor (IMAGEDIT.EXE) to convert old resource files to the current format.

C

Cannot Re-use String Constants
You are using the same value twice in a **STRINGTABLE** statement. Make sure that you have not mixed overlapping decimal and hexadecimal values.

Control Character out of range [A - Z]
A control character in the **ACCELERATORS** statement is invalid. The character following the caret (^) must be in the range A through Z.

Could not open *in-file-name*
Microsoft Windows Resource Compiler (RC) could not open the specified file. Make sure that the file exists and that you typed the filename correctly.

Couldn't open *resource-name*
Microsoft Windows Resource Compiler (RC) could not open the specified file. Make sure that the file exists and that you typed the filename correctly.

Creating *resource-name*
(V) Microsoft Windows Resource Compiler (RC) is creating a new binary resource (.RES) file.

E

Empty menus not allowed
An **END** keyword appears before any menu items are defined in the **MENU** statement. Empty menus are not permitted by Microsoft Windows Resource Compiler (RC). Make sure that you do not have any opening quotation marks within the **MENU** statement.

END expected in Dialog
The **END** keyword must appear at the end of a **DIALOG** statement. Make sure that there are no opening quotation marks left from the preceding statement.

END expected in menu
The **END** keyword must appear at the end of a **MENU** statement. Make sure that there are no mismatched **BEGIN** and **END** statements.

Error Creating *resource-name*
Microsoft Windows Resource Compiler (RC) could not create the specified binary resource (.RES) file. Make sure that it is not being created on a read-only drive. Use the **/v**option to find out whether the file is being created.

Expected Comma in Accelerator Table
Microsoft Windows Resource Compiler (RC) requires a comma between the *event* and *idvalue* parameters in the **ACCELERATORS** statement.

Expected control class name
The *class* parameter of a **CONTROL** statement in the **DIALOG** statement must be one of the following control types: **BUTTON**, **COMBOBOX**, **EDIT**, **LISTBOX**, **SCROLLBAR**, **STATIC**, or user-defined. Make sure that the class is spelled correctly.

Expected font face name
The *typeface* parameter of the **FONT** statement in the **DIALOG** statement must be a character string enclosed in double quotation marks (**"**). This parameter specifies the name of a font.

Expected ID value for Menuitem
The **MENU** statement must contain a **MENUITEM** statement, which has either an integer or a symbolic constant in the *MenuID* parameter.

Expected Menu String
Each **MENUITEM** and **POPUP** statement must contain a *text* parameter. This parameter is a string enclosed in double quotation marks (**"**) that specifies the name of the menu item or pop-up menu. A **MENUITEM SEPARATOR** statement requires no quoted string.

Expected numeric command value
Microsoft Windows Resource Compiler (RC) was expecting a numeric *idvalue* parameter in the **ACCELERATORS** statement. Make sure that you have used a **#define** constant to specify the value and that the constant used is spelled correctly.

Expected numeric constant in string table
A numeric constant, defined in a **#define** statement, must immediately follow the **BEGIN** keyword in a **STRINGTABLE** statement.

Expected numeric point size
The *pointsize* parameter of the **FONT** statement in the **DIALOG** statement must be an integer point-size value.

Expected Numerical Dialog constant
A **DIALOG** statement requires integer values for the *x, y, width*, and *height* parameters. Make sure that these values, which are included after the **DIALOG** keyword, are not negative.

Expected String in STRINGTABLE
A string is expected after each numeric *stringid* parameter in a **STRINGTABLE** statement.

Expected String or Constant Accelerator command
Microsoft Windows Resource Compiler (RC) was not able to determine which key was being set up for the accelerator. The *event* parameter in the **ACCELERATORS** statement might be invalid.

Expected VALUE, BLOCK, or END keyword.
The **VERSIONINFO** structure requires a **VALUE**, **BLOCK**, or **END** keyword.

Expecting number for ID
A number is expected for the *id* parameter of a **CONTROL** statement in the **DIALOG** statement. Make sure that you have a number or a **#define** statement for the control identifier.

Expecting quoted string for key
The key string following the **BLOCK** or **VALUE** keyword should be enclosed in double quotation marks.

Expecting quoted string in dialog class
The *class* parameter of the **CLASS** statement in the **DIALOG** statement must be an integer or a string enclosed in double quotation marks (**"**).

Expecting quoted string in dialog title
The *captiontext* parameter of the **CAPTION** statement in the **DIALOG** statement must be a character string, enclosed in double quotation marks (**"**).

F

File not found: *filename*
The file specified in the **rc** command was not found. Make sure that the file has not been moved to another directory and that the filename or path is typed correctly.

Font names must be ordinals
The *pointsize* parameter in the **FONT** statement must be an integer, not a string.

I

Invalid Accelerator
An *event* parameter in the **ACCELERATORS** statement was not recognized or was more than two characters long.

Invalid Accelerator Type (ASCII or VIRTKEY)
The *type* parameter in the **ACCELERATORS** statement must contain either the **ASCII** or **VIRTKEY** value.

Invalid control character
A control character in the **ACCELERATORS** statement is invalid. A valid control character consists of a caret (^) followed by a single letter.

Invalid Control type
The **CONTROL** statement in a **DIALOG** statement must be one of the following: **AUTO3STATE**, **AUTOCHECKBOX**, **AUTORADIOBUTTON**, **CHECKBOX**, **COMBOBOX**, **CONTROL**, **CTEXT**, **DEFPUSHBUTTON**, **EDITTEXT**, **GROUPBOX**, **ICON**, **LISTBOX**, **LTEXT**, **PUSHBOX**, **PUSHBUTTON**, **RADIOBUTTON**, **RTEXT**, **SCROLLBAR**, **STATE3**.

Invalid directive in preprocessed RC file
The specified filename has an embedded newline character.

Invalid .EXE file
The executable (.EXE) file is invalid. Make sure that the linker created it correctly and that the file exists.

Invalid switch, *option*
An option used was invalid. For a list of the command-line options, use the **rc /?** command.

Invalid switch, too many -D switches
Too many **-d** options were specified. To define a large number of symbols, use the **#include** directive to include a file of definitions.

Invalid type
The resource type was not among the types defined in the include file.

Invalid usage. Use RC -? for Help
Make sure that you have at least one filename to work with. For a list of the command-line options, use the **rc -?** command.

I/O error reading file
Read failed. Since this is a generic routine, no specific filename is supplied.

I/O error seeking in file
Seeking in file failed. Since this is a generic routine, no specific filename is supplied.

I/O error writing file
Write failed. Since this is a generic routine, no specific filename is supplied.

N

No resource binary filename specified.
The **/fo** option was used, but no binary resource (.RES) file was specified.

O

Old DIB in *resource-name*. Pass it through IMAGEDIT.
The resource file specified is not compatible with Win32. Make sure you have read and saved this file using the latest version of Microsoft Image Editor (IMAGEDIT.EXE).

Out of heap memory
There was not enough memory.

Out of memory, needed *n* bytes
Microsoft Windows Resource Compiler (RC) was not able to allocate the specified amount of memory.

R

Redefinition of Button Type
The specified button styles conflict with or modify the style specified by the control keyword. This is a warning only and may be ignored.

The following common method of declaring an automatic radio button generates this warning:

```
RADIOBUTTON "&Italic", MYRB_ITALIC, 10, 10, 30, 12, BS_AUTORADIOBUTTON
```

To avoid the warning, use the control statement appropriate for the type of button that you want to define. In this example, the corrected statement is:

```
AUTORADIOBUTTON "&Italic", MYRB_ITALIC, 10, 10, 30, 12
```

RC: Invalid switch: *option*
An option used was invalid. For a list of the command-line options, use the **rc /?** command.

RC: RCPP.ERR not found
The RCPP.ERR file must be in the current directory or in a directory specified in the PATH environment variable.

RC terminated by user
A CTRL+C key combination was pressed, exiting Microsoft Windows Resource Compiler (RC).

RC terminating after preprocessor errors
For information about preprocessor errors, see the documentation for the preprocessor.

Resource file *resource-name* **is not in** *version-number* **format.**
Make sure your icons and cursors have been read and saved using the latest version of Microsoft Image Editor (IMAGEDIT.EXE).

T

Text string or ordinal expected in Control
The *text* parameter of a **CONTROL** statement in the **DIALOG** statement must be either a text string or an ordinal reference to the type of control that is expected. If using an ordinal, make sure that you have a **#define** statement for the control.

U

Unable to create *destination*
Microsoft Windows Resource Compiler (RC) was not able to create the destination file. Make sure that there is enough disk space.

Unbalanced Parentheses
Make sure that you have closed every opening parenthesis in the **DIALOG** statement.

Unexpected value in RCData
The values for the *raw-data* parameter in the **RCDATA** statement must be integers or strings, separated by commas. Make sure that you did not leave out a comma or a quotation mark around a string.

Unexpected value in value data
A statement contained information with a format or size different from the expected value for that parameter.

Unknown DIB header format
The device-independent bitmap (DIB) header is not a **BITMAPCOREHEADER** or **BITMAPINFOHEADER** structure.

Unknown Menu SubType
The *item-definitions* parameter of the **MENU** statement can contain only **MENUITEM** and **POPUP** statements.

Unrecognized VERSIONINFO field; BEGIN or comma expected
The format of the information following a **VERSIONINFO** statement is incorrect.

V

Version WORDs separated by commas expected
Values in an information block for a **VERSIONINFO** statement should be separated by commas.

W

Warning: ASCII character not equivalent to virtual key code
An invalid virtual-key code exists in the **ACCELERATORS** statement. The ASCII values for some characters such as *****, **^**, or **&** are not equivalent to the virtual-key codes for the corresponding keys. In the case of the asterisk (*****), the virtual-key code is equivalent to the ASCII value for 8, the numeric character on the same key. Therefore, the statement **VIRTKEY '*'** is invalid.

Warning: SHIFT or CONTROL used without VIRTKEY
The **ALT**, **SHIFT**, and **CONTROL** options apply only to virtual keys in the **ACCELERATORS** statement. Make sure that the **VIRTKEY** option is used with one of these other options.

Writing resource *type image* **lang:***language* **size:***size*
(V) Microsoft Windows Resource Compiler (RC) is writing the specified resource. The *type* is the resource type. It may be a number or a string. The *image* can be a number or a string. The *language* is the national language of the resource. The *size* is the decimal size of the resource in bytes.