Resource Script Language Reference

Click the icon above to open all folders. Click an icon below to open a folder or click the underlined text to see a specific topic.

Resource Script Language Reference

Click the icon above to open all folders. Click an icon below to open or close a folder. Click the underlined text to see a specific topic.

Resource Script Language Reference

Click the icon above to open all folders. Click an icon below to open or close a folder. Click the underlined text to see a specific topic.

Text and Data Entry provide the syntax for creating static text elements and data entry fields.

Version Information and Language Support provide the syntax for providing version and international language support.

Resource Script Language Reference

Click the icon above to open all folders. Click an icon below to open or close a folder. Click the underlined text to see a specific topic.

Resource Script Language Reference

Click the icon above to open all folders. Click an icon below to open or close a folder. Click the underlined text to see a specific topic.

Text and Data Entry provide the syntax for creating static text elements and data entry fields.

Version Information and Language Support provide the syntax for providing version and international language support.

Resource Script Language Reference

Click the icon above to open all folders. Click an icon below to open or close a folder. Click the underlined text to see a specific topic.

String and Message Tables provide the syntax for creating string tables and message tables.

Style Constants provide the syntax for specifying the styles for windows, dialog boxes, and dialog box controls.

Text and Data Entry provide the syntax for creating static text elements and data entry fields.

Version Information and Language Support provide the syntax for providing version and international language support.

Resource Script Language Reference

Click the icon above to open all folders. Click an icon below to open or close a folder. Click the underlined text to see a specific topic.

Resource Script Language Reference

Click the icon above to open all folders. Click an icon below to open or close a folder. Click the underlined text to see a specific topic.

Resource Script Language Reference

Click the icon above to open all folders. Click an icon below to open or close a folder. Click the underlined text to see a specific topic.

Window styles

Text and Data Entry provide the syntax for creating static text elements and data entry fields.

Version Information and Language Support provide the syntax for providing version and international language support.

Resource Script Language Reference

Click the icon above to open all folders. Click an icon below to open or close a folder. Click the underlined text to see a specific topic.

international language support.

Resource Script Language Reference

Click the icon above to open all folders. Click an icon below to open or close a folder. Click the underlined text to see a specific topic.

Resource Script Language Reference

Click any icon to close all folders or click the underlined text to see a specific topic.

CLASS

CONTROL

DIALOG

FONT

GROUPBOX

MENU

SCROLLBAR

STYLE

Directives provide the syntax for the preprocessor directives.

#define

#elif

#else

#endif

#error

#ifdef

#if

#ifndef

#include

#line

#pragma

#undef

RCINCLUDE

Menus and Accelerators provide the syntax for creating menus and accelerator tables.

ACCELERATOR

MENU

MENUITEM

POPUP

String and Message Tables provide the syntax for creating string tables and message tables.

MESSAGETABLE

STRINGTABLE

Style Constants provide the syntax for specifying the styles for windows, dialog boxes, and dialog box controls.

BUTTON class styles

COMBOBOX class styles

Control window styles

Dialog window styles

EDIT class styles

Extended window styles

LISTBOX class styles

SCROLLBAR class styles

STATIC class styles

Window styles

Text and Data Entry provide the syntax for creating static text elements and data entry fields.

COMBOBOX

CTEXT

EDITTEXT

LISTBOX

LTEXT

RTEXT

Version Information and Language Support provide the syntax for providing version and international language support.

- CHARACTERISTICS
- LANGUAGE
- VERSION
- VERSIONINFO

Bitmapped Resources

See Also

There are four kinds of bitmapped resources:

- bitmaps: accessed using the BITMAP statement
- cursors: accessed using the CURSOR statement
- fonts: accessed using the FONT statement
- icons: accessed using the ICON statement

Other than including these resources in your resource script file, you can only edit the hexadecimal code for these resources when you use a standard text editor, such as Resource Workhop's Script Editor.

In addition to using these resources in a resource script, you can also edit and create these types of resources using Resource Workshop's Bitmap Editor.

See Also
Resource Script Language (Overview)

Buttons and Checkboxes

See Also

There are several types of checkboxes and buttons you can use:

Checkboxes

- CHECKBOX
- AUTOCHECKBOX
- STATE3
- AUTO3STATE

Push Buttons

- PUSHBUTTON
- DEFPUSHBUTTON

Radio Buttons

- RADIOBUTTON
- AUTORADIOBUTTON

You can only use these types of controls in dialog box resources. You can edit the resource script for these controls within a dialog box definition using a standard text editor, such as Resource Workhop's Script Editor.

In addition to using these resources in a resource script, you can also edit and create these types of resources using Resource Workshop's Dialog Editor.

See Also

Dialog Boxes

Style Constants

Text and Data Entry

Resource Script Language (Overview)

Custom Resources

See Also

You can define your own custom resources and resource types using these statements:

RCDATA

User-Defined Resources

Dialog Boxes

See Also

These are the statements you can use to build dialog box resources:

CAPTION

CLASS

CONTROL

DIALOG

FONT

GROUPBOX

MENU

SCROLLBAR

STYLE

In addition to creating dialog boxes using a resource script, you can also edit and create dialog boxes using Resource Workshop's Dialog Editor.

See Also

Buttons and Checkboxes

Style Constants

Text and Data Entry

Resource Script Language (Overview)

Menus and Accelerators

See Also

You can create menu and accelerator resources using these statements:

ACCELERATOR

MENU

MENUITEM

POPUP

In addition to creating menus and accelerators using a resource script, you can also edit and create menus and accelerators using Resource Workshop's Menu Editor and Accelerator Editor.

String and Message Tables

See Also

You can store character strings and messages as resources that your Windows application can call as needed:

MESSAGETABLE

STRINGTABLE

In addition to creating string tables and message tables using a resource script, you can also edit and create string tables and message tables using Resource Workshop's String Editor.

Style Constants

See Also

There are predefined constants for class and window styles that you can specify for many of the resources you want to create:

- BUTTON class styles
- COMBOBOX class styles
- Control window styles
- Dialog window styles
- EDIT class styles
- Extended window styles
- LISTBOX class styles
- SCROLLBAR class styles
- STATIC class styles
- Window styles

Text and Data Entry

See Also

There are several resource statements that allow you to handle static text elements as well as gather information from the user of your application:

Static Text

centered text (CTEXT)

left-aligned text (LTEXT)

right-aligned text (RTEXT)

Data Entry

combo box (COMBOBOX)

editable text (EDITTEXT)

list box (LISTBOX)

You can only use these types of controls in dialog box resources. You can edit the resource script for these controls within a dialog box definition using a standard text editor, such as Resource Workhop's Script Editor.

In addition to using these resources in a resource script, you can also edit and create these types of resources using Resource Workshop's Dialog Editor.

See Also
Buttons and Checkboxes
Dialog Boxes
Style Constants
Resource Script Language (Overview)

Version Information and Language Support

See Also

You can include version information along with your resources, as well as create resources with specific language support:

CHARACTERISTICS

LANGUAGE

VERSION

VERSIONINFO

Resource Script Language (Overview)

See Also

When you create resources, Resource Workshop builds a resource script file in your project window. Most of the time, you won't need to use this language. Occasionally, however, you might want to edit a resource script with a text editor.

Each resource script statement specifies a resource to be included in your executable file. You can list the primary statements in any order you choose; there is no "program flow" in a resource script file, although you must follow the syntax given for each statement carefully.

See Also

Resource Script Language Reference (Alphabetical Listing)

Resource Script Language Reference (Functional Listing)

Resource Script Language Reference

This is an alphabetical listing of the Resource Script language elements.

For an overview of the Resource Script Language, select Resource Script Language Overview.

For a functional listing of Resource Script Language elements, select Resource Script Language Reference (Functional Listing)

#
#define

#elif

#else

#endif

#error

#ifdef

#if

#ifndef

#include

#line

#pragma

A
ACCELERATOR

AUTO3STATE

AUTOCHECKBOX

AUTORADIOBUTTON

B
Binary resource statement

BITMAP

Binary Resource Statement

See Also

Binary resource statements identify a resource and name the file that contains the resource.

For example,

```
pencil CURSOR MOVEABLE pencil.cur
```

names the resource "pencil" and identifies it as a CURSOR resource located in the file PENCIL.CUR.

The syntax of all binary resources statements looks like this:

```
resource-name resource-type [load-type] [memory-option] filename
```

Binary resource statements specify bitmaps, cursors, fonts, and icons. Each statement has its own syntax. The binary resources statements are:

BITMAP

CURSOR

FONT

ICON

See Also
Multiple-line statements

Multiple-Line Statements

See Also

Multiple-line statements specify the contents of a resource.

Here's an example of a multiple-line statement for a menu resource:

```
mainmenu MENU PRELOAD
BEGIN
  POPUP "&File"
  BEGIN
    MENUITEM "&New", 100
    MENUITEM "&Open", 101
    MENUITEM "&Close", 102, GRAYED
    MENUITEM "&Save", 103
    MENUITEM "Save &As", 104
    MENUITEM SEPARATOR
    MENUITEM "&Print", 105
    MENUITEM "&Draft Printing", 107, CHECKED
    MENUITEM SEPARATOR
    MENUITEM "E&xit", 106
  END
    MENUITEM "&Help", 200
END
```

This example specifies the main menu of an application. The main menu contains the File and Help menus.

File is a pop-up menu (commonly known as a drop-down menu) with several options, each defined with a MENUITEM substatement contained within the BEGIN and END keywords. This is characteristic of all multiple-line statements. Like other multiple-line statements, this statement permits you to include multiple substatements between the words BEGIN and END.

Multiple-line statements specify accelerators, string resources, raw data resources, dialog boxes, and menus. Each statement has its own syntax. The multiple-line statements are:

ACCELERATOR

DIALOG

MENU

RCDATA

STRINGTABLE

See Also
Binary resource statement

ACCELERATOR

See Also Examples

```
resource-name ACCELERATOR
BEGIN
  keystroke, acc-ID, [keystroke-type] [modifier key] [NOINVERT]
END
```

Parameters

resource-name	Text identifier or numeric ID for this resource. The name or number must be unique within the ACCELERATOR resource type. Numeric IDs must be positive integers.
keystroke	Value of the character code that activates this accelerator. The field can be either a character in double quotes or a numeric value. - If a character in quotes is used, it can be preceded by a carat (^) to indicate that it is a control character (for example "^a" indicates Ctrl-A). - If a numeric value is used, it is either the ASCII value of the accelerator key or the value for a virtual key, depending on the keystroke-type.
acc-ID	User-assigned integer value that identifies the string. Uniqueness is not required of IDs: Two different key combinations can be used to invoke the same command. The ID is sent in the *wParam* of a WM_COMMAND message. The high-order word of the *lParam* of the WM_COMMAND message is set to 1 to indicate the message resulted from an accelerator key.
keystroke-type	Valid only if the keystroke field contains a numeric value. In such cases, this field indicates whether the keystroke number is an ASCII value or virtual key value as follows: ASCII — Keystroke field is ASCII value. VIRTKEY — Keystroke field is virtual key value.
modifier key	Indicates if any modifier keys must be held down while typing the keystroke to activate the accelerator. If this field is missing, no modifier keys are required. Valid modifier keys are Shift, Alt, and Ctrl. The SHIFT and CTRL modifiers have no effect unless the VIRTKEY *keystroke-type* is used. SHIFT — Shift key must be held down. CONTROL — Ctrl key must be held down. ALT — Alt key must be held down.
NOINVERT	Disables highlighting of the menu title of the accelerated menu item when using accelerator keys for actions which have no menu item equivalent. If the NOINVERT field is missing, these menu titles are highlighted.

Remarks

ACCELERATOR is a multiple-line statement that defines keyboard shortcuts for menu items and other program control actions.

The accelerator resource associates one or more accelerator keys with a corresponding accelerator command ID (*acc-ID*). Although *acc-ID*s must be numeric, the #define can be used to simplify access by the program.

See Also

#define

CHARACTERISTICS

LANGUAGE

VERSION

AUTO3STATE

See Also

```
AUTO3STATE text, control-ID, x, y, width, height, [c-style]
```

Parameters

text	Text string in double quotes. It is displayed next to the button. It can be specified as a keyboard accelerator mnemonic.
control-ID	A numeric identifier for this control. This number must be a unique short integer. Windows uses the *control-ID* to indicate which control has been selected.
x, *y*	Horizontal and vertical positions, respectively, of the button relative to the dialog window in which it appears. These numbers are specified in dialog units.
width, *height*	The width and height of the control (specified in dialog units).
c-style	Specifies styles for the control, which can be a combination of the BUTTON class style constant BS_AUTO3STATE and these window style constants: WS_TABSTOP WS_DISABLED WS_GROUP The default style is BS_AUTO3STATE and WS_TABSTOP.

Remarks

A DIALOG definition that places a 3-state check box control in a dialog box. It's valid only within a DIALOG definition. A 3-state check box is a square button with descriptive text to the left or right of the button. Normally, text is displayed to the right of the button.

A 3-state check box control maintains 3 states: It is either checked (in which case, a large X fills the square), unchecked or "indeterminate" (grayed). The control sends a message to its parent when it's selected.

See Also

AUTOCHECKBOX

CHECKBOX

CONTROL

DIALOG

STATE3

AUTOCHECKBOX

See Also

```
AUTOCHECKBOX text,control-ID, x, y, width, height, [c-style]
```

Parameters

text	Text string in double quotes. It is displayed next to the button. It can be specified as a keyboard accelerator mnemonic.
control-ID	A numeric identifier for this control. This number must be a unique short integer. Windows uses the *control-ID* to indicate which control has been selected.
x, *y*	Horizontal and vertical positions, respectively, of the button relative to the dialog window in which it appears. These numbers are specified in dialog units.
width, *height*	The width and height of the control (specified in dialog units).
c-style	Specifies the styles of the control. This value can be a combination of the BUTTON class style constant BS_AUTOCHECKBOX and these window style constants: WS_TABSTOP WS_GROUP The default style is BS_AUTOCHECKBOX and WS_TABSTOP.

Remarks

A DIALOG definition that places an automatic check box control in a dialog box. It's valid only within a DIALOG definition. An AUTOCHECKBOX is a square rectangle with descriptive text to the left or right of the button. When you choose the control, the rectangle is highlighted and a message is sent to the parent window.

See Also

AUTO3STATE

CHECKBOX

CONTROL

DIALOG

STATE3

AUTORADIOBUTTON

See Also

```
AUTORADIOBUTTON text, control-ID, x, y, width, height [c-style]
```

Parameters

text	Text string in double quotes. It is displayed next to the button. It can be specified as a keyboard accelerator mnemonic.
control-ID	A numeric identifier for this control. This number must be a unique short integer. Windows uses the *control-ID* to indicate which control has been selected.
x, *y*	Horizontal and vertical positions, respectively, of the button relative to the dialog window in which it appears. These numbers are specified in dialog units.
width, *height*	The width and height of the control (specified in dialog units).
c-style	Specifies the styles of the control. This value can be a combination of the BUTTON class style constant BS_AUTORADIOBUTTON and these window style constants: WS_DISABLED WS_TABSTOP WS_GROUP The default style is BS_AUTORADIOBUTTON and WS_TABSTOP.

Remarks

A DIALOG definition that places an automatic radio button control in a dialog box. It's valid only within a DIALOG definition. An automatic radio button is automatically mutually exclusive with other AUTORADIOBUTTON controls in the same group. When you choose an automatic radio button, a BN_CLICKED message is sent to the application.

See Also

CONTROL

DIALOG

RADIOBUTTON

BITMAP

See Also Examples

```
resource-name BITMAP [load-type] [memory-option] filename
```

Parameters

resource-name	Text identifier or numeric ID for this resource. The identifier or number must be unique within the BITMAP resource type. Numeric IDs must be positive integers.
load-type	Specifies when the resource is loaded into memory.
memory-option	Specifies how the resource is loaded into memory.
filename	The name of the DOS file containing the bitmap data. A relative or full path name can be used to specify files which are not in the current working directory. The data in the specified file is included in the current project.

Remarks

BITMAP is a binary resource statement that associates a file containing bitmap resource data with a resource name and causes the bitmap data to be included in the current project.

You can write a BITMAP statement with the same syntax as RCDATA.

See Also
RCDATA

CHARACTERISTICS

See Also

```
CHARACTERISTICS dword
```

Parameter

dword	A user-defined doubleword value.

Remarks

You use the CHARACTERISTICS statement to specify a resource's characteristics. The value appears with the resource in the compiled .RES file and can be used by tools that read and write resource-definition files. It is not stored in the executable file and has no significance to Windows.

Use the CHARACTERISTICS statement before the BEGIN statement in these resource definitions:

- ACCELERATOR
- DIALOG
- MENU
- RCDATA
- STRINGTABLE

The characteristic applies only to the specific resource.

See Also

LANGUAGE

VERSION

CHECKBOX

See Also Examples

```
CHECKBOX text, control-ID, x, y, width, height, [c-style]
```

Parameters

text	Text string in double quotes. It is displayed next to the button. It can be specified as a keyboard accelerator mnemonic.
control-ID	A numeric identifier for this control. This number must be a unique short integer. Windows uses the *control-ID* to indicate which control has been selected.
x, *y*	Horizontal and vertical positions, respectively, of the button relative to the dialog window in which it appears. These numbers are specified in dialog units.
width, *height*	The width and height of the control (specified in dialog units).
c-style	The control style constant (an unsigned long integer value that is interpreted as a series of bit flags). Constants can be combined with a bitwise OR to apply multiple styles to the control. The field can contain these window style constants: WS_DISABLED WS_TABSTOP WS_GROUP and any of the BUTTON class style constants. The default styles are WS_TABSTOP and BS_CHECKBOX.

Remarks

A DIALOG definition that places a check box control in a dialog box. It's valid only within a DIALOG definition.

A check box is a square button with descriptive text to the left or right of the button. Normally, text is displayed to the right of the button. You can change this to the left side with the appropriate *c-style*.

A check box control maintains a state: It is either checked (in which case, a large X fills the square) or unchecked. Check box controls do not change state automatically: The application programmer must change their state in response to BN_CLICKED notification messages. For that reason, it's a good idea to use the auto-check box control.

A check box is a member of the button class. When the user presses the mouse button in the control's area, the button is highlighted. When the mouse button is released, it is returned to normal, and a message is sent to the parent window indicating that the button has been pressed.

See Also
AUTO3STATE
AUTOCHECKBOX
DIALOG
STATE3

COMBOBOX

Examples

```
COMBOBOX ID, x, y, width, height, [c-style]
```

Parameters

ID	Numeric identifier for this control. This number must be a unique short integer. Windows uses the control ID to indicate which control has been selected.
x, *y*	Integer values that specify the x- and y-coordinates of the upper left corner of the COMBOBOX control. These values are specified in dialog units.
width, *height*	Integer values that specify the size of the control. These values are specified in dialog units.
c-style	The control style constant (an unsigned long integer value that is interpreted as a series of bit flags). Constants can be combined with a bitwise OR to apply multiple styles to the control. The field can contain these window style constants: WS_DISABLED WS_GROUP WS_TABSTOP WS_VSCROLL and any of the COMBOBOX class style constants. The default styles are WS_TABSTOP and CBS_SIMPLE.

Remarks

A DIALOG definition that creates a combo box, a combination of either a static text field or an edit field with a list box.

If a static text field is used, this field displays the current selection in the list box. If an edit field is used, the selection is typed in and the list box highlights the first item that matches the typed entry.

CONTROL

See Also Examples

```
CONTROL text, control-ID, c-class, c-style, x, y, width, height
```

Parameters

text	Text string in double quotes. This text is displayed, if appropriate, for the type of control specified. It can be specified as a keyboard accelerator mnemonic.
control-ID	A numeric identifier. This number can be a unique short integer, although unreferenced static controls are usually given a control ID of -1 to document that they are truly static. Windows uses the control ID to indicate which control has been selected.
c-class	A byte or a string indicating the control's class. The standard classes have predefined constants that can be used to identify them: c-class must be one of these contants: AUTO3STATE AUTOCHECKBOX AUTORADIOBUTTON BUTTON COMBOBOX EDIT LISTBOX SCROLLBAR STATE3 STATIC
c-style	The control style constant (an unsigned long integer value that is interpreted as a series of bit flags). Constants can be combined with a bitwise OR to apply multiple styles to the control. The options available to you depend on the *c-class* value. There is no default style. The high-order word of this field is reserved for Windows. The low-order word is available for user-defined control styles.
x, *y*	Horizontal and vertical positions of the control relative to the dialog window in which it appears. These numbers are specified in dialog units.
width, *height*	The width and height of the control. These numbers are specified in dialog units.

Remarks

CONTROL is a DIALOG definition that places any type of control in a dialog box. It is valid only within a DIALOG definition and can duplicate the function of these definitions:

CHECKBOX
COMBOBOX
CTEXT
DEFPUSHBUTTON
EDITTEXT
GROUPBOX
ICON
LISTBOX
LTEXT

PUSHBUTTON
RADIOBUTTON
RTEXT
SCROLLBAR

See Also
DIALOG

CTEXT

See Also Examples

```
CTEXT text, control-ID, x, y, width, height, [c-style]
```

Parameters

text	A text string, enclosed in quotes, that appears in the dialog window. It can be specified as a keyboard accelerator mnemonic. A common Windows programming technique is to label an edit control by preceding it with a static text control. This label contains a mnemonic for the edit field, such as "&Name". If the CTEXT control doesn't use the WS_TABSTOP control style, Windows will set the focus in the next control.
control-ID	A numeric identifier. The control ID is used by Windows and an application to identify the control within the dialog. If there is no need to refer to the control at run time (as is most often the case with static controls), usually this field is set to -1 to document that it is truly static.
x, *y*	Horizontal and vertical positions of the text relative to the dialog window in which the text appears. These numbers are specified in dialog units.
width, *height*	The size of the control. These numbers are specified in dialog units. The static text appears centered within the specified area. If the text is too wide to fit on a single line, Windows automatically word wraps the text to multiple lines.
c-style	The control style constant (an unsigned long integer value that is interpreted as a series of bit flags). More than one control style can be combined using a bitwise OR. Defaults are the STATIC class style constant SS_CENTER and the window style constant WS_GROUP.

Remarks

CTEXT is a DIALOG definition that specifies a text string, its attributes, and where it's located in the dialog box. CTEXT defines centered static text in a dialog window. The text is centered within the specified rectangle. If the text is too wide to fit within the rectangle, it is automatically wrapped.

CTEXT is valid only in a DIALOG definition.

See Also

DIALOG

LTEXT

RTEXT

CURSOR

Examples

```
resource-name CURSOR [load-type] [memory-option] filename
```

Parameters

resource-name	Text identifier or numeric ID for this resource. The identifier or number must be unique within the CURSOR resource type. Numeric IDs must be positive integers.
load-type	Specifies when the resource is loaded into memory.
memory-option	Specifies how the resource is loaded into memory.

Remarks

CURSOR is a binary resource statement that causes cursor data to be included in the current project.

DEFPUSHBUTTON

See Also Examples

```
DEFPUSHBUTTON text, control-ID, x, y, width, height, [c-style]
```

Parameters

text	A text string in double quotes. This text is displayed inside the button area. It can be specified as a keyboard accelerator mnemonic.
control-ID	A numeric identifier for this control. It must be a unique integer. The control ID is used by Windows to indicate which control has been selected. By convention, only one DEFPUSHBUTTON is included in a dialog. It is given the control ID IDOK (1). When a user presses the Enter key in a modal dialog box, Windows sends a WM_COMMAND message with *wParam* set to IDOK. Also, when the user presses the Esc key, Windows sends a WM_COMMAND with *wParam* set to IDCANCEL (2).
x, *y*	Horizontal and vertical positions of the button relative to the dialog window in which it appears. These numbers are specified in dialog units.
width, *height*	The width and height of the control. These numbers are specified in dialog units.
c-style	The control style constant (an unsigned long integer value that is interpreted as a series of bit flags). Constants can be combined with a bitwise OR to apply multiple styles to the control. The field can contain these window style constants: WS_DISABLED WS_GROUP WS_TABSTOP and any of the BUTTON class style constants. The default styles are WS_TABSTOP and BS_DEFPUSHBUTTON.

Remarks

DEFPUSHBUTTON is a DIALOG definition that places a default push button control in a dialog box.

A DEFPUSHBUTTON is a square button containing text describing its action. This control is almost identical to a normal push button, except it has a heavy border that indicates to the user it is the default action for this dialog window.

DEFPUSHBUTTON is valid only within a DIALOG definition. A DEFPUSHBUTTON is a member of the BUTTON class.

See Also
DIALOG

DIALOG

See Also Examples

```
resource-name DIALOG [load-type] [memory-option] x, y, width, height
[STYLE w-style] [CAPTION w-cap]
[MENU res-name] [CLASS w-class]
[FONT f-spec]
BEGIN
  dialog-controls
END
```

Parameters

resource-name	Text identifier or numeric ID for this resource. The identifier or number must be unique within the DIALOG resource type. Numeric IDs must be positive integers.
load-type	Specifies when the resource is loaded into memory.
memory-option	Specifies how the resource is loaded into memory.
x, *y*	Horizontal and vertical positions of the upper left corner of the dialog window's client area. These coordinates can either be relative to the window's parent or owner window, or relative to the origin of the screen. This is determined by the window's style setting. Dialog windows are positioned relative to their parent or owner window unless the dialog window style constant DS_ABSALIGN is used.
width, *height*	The size of the client area of the window in dialog units.
STYLE	The window style of the dialog box. The window style specifies whether the box is a pop-up or a child window.
CAPTION	The title of the dialog box. The title appears in the box's caption bar (if it has one). The default caption is empty.
MENU	The dialog box's menu. If no statement is given, the dialog box has no menu.
CLASS	The class of the dialog box. If no statement is given, the Windows standard dialog class will be used as the default.
FONT	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.
dialog-controls	Definition(s) that specify the content of the dialog windows. This includes static and editable text, various boxes and buttons, controls, and icons. Here are the dialog control statements:

AUTO3STATE

AUTOCHECKBOX

AUTORADIOBUTTON

CHECKBOX

COMBOBOX

CONTROL

CTEXT

DEFPUSHBUTTON

EDITTEXT

GROUPBOX

ICON

LISTBOX

LTEXT
PUSHBUTTON
RADIOBUTTON
RTEXT
SCROLLBAR
STATE3

Remarks

DIALOG is a multiple-line statement that specifies a dialog window.

A dialog window includes the window style, class, size, location, and the controls which will appear in the window. Dialog windows can contain text, check boxes, various buttons, icons, controls, list boxes, and so on.

See Also

CHARACTERISTICS

LANGUAGE

VERSION

STYLE (DIALOG statement)

Statement in the DIALOG definition that defines the window style of the dialog box. The window style specifies whether the box is a pop-up or a child window. If you don't specify a style, the default values WS_POPUP, WS_BORDER, and WS_SYSMENU are used.

Parameter

w-style	The window style constant or dialog window style constant. w-style is an integer value or a predefined name.

CAPTION (DIALOG statement)

Statement in the DIALOG definition that defines the dialog box title. The title appears in the box's caption bar (if it has one). The default caption is empty.

If CAPTION is not present in the DIALOG definition, the caption bar is empty.

Parameter

w-cap	An ASCII character string enclosed in double quotes.

MENU (DIALOG statement)

Statement in the DIALOG definition that associates a menu resource with the dialog.

If MENU is not present in the DIALOG definition, no menu is associated with the dialog box.

Parameter

res-name	The resource identifier or numeric ID of the associated menu.

CLASS (DIALOG statement)

Statement in the DIALOG definition that overrides the normal processing of a dialog box. It converts a dialog box to a window of the specified class; and depending on the class, could give undesirable results. Do not use the predefined control-class names with this statement.

Using a custom dialog class provides additional control over the behavior of the dialog window. In order to create a custom dialog class, you must set the *cbWndExtra* field of the WNDCLASS structure to at least as many bytes as used by DLGWINDOWEXTRA, the default dialog class. However, the additional control provided by the custom class is illusory because much of the Windows dialog manager is implemented in the *IsDialogMessage* function, not the dialog window procedure.

If no CLASS statement is given, the Windows standard dialog class will be used as the default.

Parameter

w-class	An integer or text string that specifies the desired window class.

FONT (DIALOG statement)

Statement in the DIALOG definition that 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.

Parameter

f-spec	The font specification. It consists of a point size (in points) followed by a font typeface string in double quotation marks (for example, 12, "Helv"). Windows uses the bold (approximately 700) weight for the font when this field is used. To use a lighter-weight attribute, use the WM_SETFONT message at run time to set the font.

EDITTEXT

See Also Examples

```
EDITTEXT control-ID, x, y, width, height, [c-style]
```

Parameters

control-ID	Numeric identifier for this control. This number must be a unique integer. The control ID is used by Windows to indicate which control has been selected.
x, *y*	Horizontal and vertical positions of the edit text control relative to the dialog window in which it appears. These numbers are specified in dialog units.
width, *height*	The width and height of the edit text control. These numbers are specified in dialog units.
c-style	The control style constant (an unsigned long integer value that is interpreted as a series of bit flags). Constants can be combined with a bitwise OR to apply multiple styles to the control. The field may contain the window style constants: WS_DISABLED WS_GROUP WS_HSCROLL WS_TABSTOP WS_VSCROLL and any of the EDIT class style constants. The default styles are WS_BORDER, WS_TABSTOP, and ES_LEFT.

Remarks

EDITTEXT is a DIALOG definition that places an editable text field in a dialog box.

Text can be edited using the mouse, cursor keys, Backspace, and Del. An edit text control belongs to the EDIT class.

EDITTEXT is valid only within a definition.

See Also
DIALOG

FONT

Examples

```
resource-num FONT [load-type] [memory-option] filename
```

Parameters

resource-num	The resource number. It can be either a unique name (text) or an integer value that is unique within the font resource type. This number identifies the font; you can't use resource names for fonts.
load-type	Specifies when the resource is loaded into memory.
memory-option	Specifies how the resource is loaded into memory.
filename	The name of the DOS file containing the font data. A full path name can be used to specify files which are not in the current working directory. The data in the specified file is included in the current project.

Remarks

FONT is a binary resource statement definition that associates a file containing font resource data with a resource number. It also causes the font data to be included in the current project.

GROUPBOX

See Also Examples

```
GROUPBOX text, control-ID, x, y, width, height, [c-style]
```

Parameters

text	A text string in double quotes. This text is displayed at the top of the group box. It can be specified as a keyboard accelerator mnemonic.
control-ID	A numeric identifier for this control. This number can be a unique integer, but is often -1. The control ID is used by Windows to indicate which control has been selected.
x, *y*	Horizontal and vertical positions of the group box relative to the dialog window in which it appears. These numbers are specified in dialog units.
width, *height*	The width and height of the group box. These numbers are specified in dialog units.
c-style	The control style constants (an unsigned long integer value that is interpreted as a series of bit flags). Constants can be combined with a bitwise OR to apply multiple styles to the control. The field may contain these window style constants: WS_DISABLED WS_TABSTOP and any of the BUTTON class style constants. The default styles are WS_TABSTOP and BS_GROUPBOX.

Remarks

GROUPBOX is a DIALOG definition that places a static group box in a dialog box.

You usually use a group box to visually group a number of related controls. The user can then easily see which controls are related. The text field of the group box definition usually describes the group. A group box consists of a rectangle of the specified size, with a text title overwriting the left part of the top line of the rectangle.

GROUPBOX is valid only within a DIALOG definition.

See Also
DIALOG

ICON

Examples

For a type 1 icon definition:

```
resource-name ICON [load-type] [memory-option] filename
```

For a type 2 icon definition:

```
ICON resource-name, control-ID, x, y, width, height, [c-style]
```

Parameters

For a type 1 icon definition:

resource-name	Text identifier or numeric ID for this resource. The identifier or number must be unique within the ICON resource type. Numeric IDs must be integers.
load-type	Specifies when the resource is loaded into memory.
memory-option	Specifies how the resource is loaded into memory.
filename	The name of the DOS file containing the icon data. A full path name can be used to specify files which are not in the current working directory. The data in the specified file is included in the current project.

For a type 2 icon definition:

resource-name	The identifier of the icon resource to be included in the dialog window. This resource identifier is assigned to the icon using a type 1 icon definition. Note that this is not the icon's file name. If the resource name is a text identifier, it must be enclosed in quotes.
control-ID	A numeric identifier for this control. This number must be a unique integer. The control ID is used by Windows to indicate which control has been selected.
x, *y*	Horizontal and vertical positions of the icon relative to the dialog window in which the icon appears. These numbers are specified in dialog units.
width, *height*	Ignored. The icon is automatically sized.
c-style	The control style constant (an unsigned long integer value that is interpreted as a series of bit flags). Constants can be combined with a bitwise OR to apply multiple styles to the control. The only permissible style is the STATIC class style constant, SS_ICON.

Remarks

There are two types of icon definitions:

type 1 icon	A binary resource statement that associates a file containing icon resource data with a resource name.
type 2 icon	A DIALOG definition that specifies a static icon control in a dialog box.

The type 1 icon definition tells Resource Workshop in which file an icon's data is contained, and gives this icon the specified *resource-name*. Data from the indicated file is included in the executable file when the application is built.

The type 2 icon definition refers to an icon that was specified with a type 1 definition (or defined in a free-form resource). Type 2 icon definitions place static icon controls within a dialog window. This type of ICON definition is only valid within a DIALOG definition.

LANGUAGE

See Also

```
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.

Remarks

You use the LANGUAGE statement to assign the language and the sub-language to resources. The definition is active until you use another LANGUAGE statement to change it.

The specified language applies only to an individual resource when you use the LANGUAGE statement before the BEGIN statement in these resource definitions:

ACCELERATOR

DIALOG

MENU

RCDATA

STRINGTABLE

See Also

CHARACTERISTICS

VERSION

LISTBOX

See Also Examples

```
LISTBOX control-ID, x, y, width, height, [c-style]
```

Parameters

control-ID	Numeric identifier for this control. This number must be a unique integer. The control ID is used by Windows to indicate which control has been selected.
x, *y*	Horizontal and vertical positions of the list box relative to the dialog window in which it appears. These numbers are specified in dialog units.
width, *height*	The width and height of the control. These numbers are specified in dialog units.
c-style	The control style constant (an unsigned long integer value that is interpreted as a series of bit flags). Constants can be combined with a bitwise OR to apply multiple styles to the control. The field may contain these window style constants: WS_BORDER WS_DISABLED WS_VSCROLL and any of the LISTBOX class style constants. Default values are WS_VSCROLL, WS_BORDER, and LBS_NOTIFY.

Remarks

LISTBOX is a DIALOG definition that places a list box control in a dialog box.

A list box is an area where multiple text strings are displayed. A list box is a member of the list box class.

LISTBOX is valid only within a DIALOG definition.

See Also
DIALOG

LTEXT

See Also Examples

```
LTEXT text, control-ID, x, y, width, height, [c-style]
```

Parameters

text	Text string, enclosed in quotes, that appears in the dialog window. It can be specified as a keyboard accelerator mnemonic. A common Windows programming technique is to label an edit control by preceding it with a static text control. The static label contains a mnemonic for the edit field, such as "&Name". If the LTEXT control does not use the window style constant WS_TABSTOP, Windows sets the focus in the next control.
control-ID	A numeric identifier for this control. This number must be a unique integer. The control ID is used by Windows to indicate which control has been selected. If there is no need to refer to the control at run time (as is most often the case with static controls), this field is by convention set to -1 to document that it is truly static.
x, *y*	Horizontal and vertical positions of the text relative to the dialog window in which the text appears. These numbers are specified in dialog units.
width, *height*	The size of the control in dialog units. The static text appears centered within the specified area.
c-style	The control style constant (an unsigned long integer value that is interpreted as a series of bit flags). More than one control style can be combined using a bitwise OR. The default style is the STATIC class style constants SS_LEFT and the window style WS_GROUP. The WS_GROUP style is always set for static text.

Remarks

LTEXT is a DIALOG definition that defines left-aligned static text in a dialog box.

LTEXT specifies a text string, its attributes, and where it is located in the dialog window. The text is aligned flush left within the specified rectangle. If the text is too wide to fit within the rectangle, it automatically wraps.

This definition can only appear within a DIALOG definition.

See Also

CTEXT

DIALOG

RTEXT

MENU

See Also Examples

```
resource-name MENU [load-type] [memory-option]
BEGIN
  item-definitions
END
```

Parameters

resource-name	Text identifier or numeric ID for this resource. The identifier or number must be unique within the MENU resource type. Numeric IDs must be integers.
load-type	Specifies when the resource is loaded into memory.
memory-option	Specifies how the resource is loaded into memory.
item-definitions	Contain one or more MENUITEM or POPUP definitions. The MENUITEM definition specifies individual menu items, and the POPUP definition describes a pop-up menu (also known as a drop-down menu).

Remarks

MENU is a multiple-line statement that defines a menu resource and specifies which menu items appear on this menu.

A MENU definition contains the item definitions, MENUITEM and POPUP.

See Also

CHARACTERISTICS

LANGUAGE

VERSION

MENUITEM

See Also Examples

```
MENUITEM [item-text] [item-ID] [item-attributes]
```

or

```
MENUITEM SEPARATOR
```

Parameters

item-text	Describes the menu selection. It is a character string enclosed in double quotes. The item-text also indicates which character in the menu item serves as the menu mnemonic. The item text string can also include the following escape codes: \a and \t. The \a escape code aligns the following text flush right. The \t escape code inserts a tab in the item text aligning the text. It should only be used within a POPUP definition and not for items in the main menu bar.
item-ID	An integer value that identifies the menu item. Windows uses the value to indicate which menu item has been selected.
item-attributes	Accepts one or more keywords that describe the menu item's state (active, inactive, grayed, checked, etc.) The keywords GRAYED and INACTIVE are mutually exclusive. Other keywords can be used together by combining their values with a bitwise OR.

GRAYED	Item is unavailable for selection.
INACTIVE	Item is never available for selection.
CHECKED	Checkmark appears next to item. Not supported for top-level menu items.
HELP	Appears on right side of menu bar.
MENUBREAK	Item begins a new menu column.
MENUBARBREAK	Item begins a new menu row.

SEPARATOR	Indicates that this menu item is a separator line, rather than a selectable item. Separator lines are horizontal lines separating menu items when used in pop-up menus, and vertical lines separating menu titles when used in the menu bar.

Remarks

A MENU definition that defines a single menu item or a menu separator.

The MENUITEM definition associates an item ID with a specific menu element. MENUITEM can be used only within a MENU or POPUP definition.

See Also
MENU

MESSAGETABLE

See Also

```
MESSAGETABLE [load-type] [memory-option]
BEGIN
  message-ID, message
END
```

Parameters

load-type	Specifies when the resource is loaded into memory.
memory-option	Specifies how the resource is loaded into memory.
string-ID	A user-assigned integer value that identifies the string. Each string ID must be unique. The string ID is used at run time by the *LoadString* function to determine which string is being requested by the program.
message	An ASCII string in standard C language format.

Remarks

MESSAGETABLE is a multiple-line statement that specifies null-terminated ASCII strings that can be accessed by the program.

Each string is assigned a unique unsigned short integer string ID. Strings are read in for access at run time by calling the *LoadString* function with the desired string ID.

The message table mechanism is a convenient method to keep text strings separate from code for easy update and possible translation into foreign languages.

Although string IDs must be numeric, the #define preprocessor directive can be used to simplify access by the program.

See Also
CHARACTERISTICS
LANGUAGE
STRINGTABLE
VERSION

POPUP

See Also Examples

```
POPUP [popup-name] [popup-attributes]
BEGIN
  item-definitions
END
```

Parameters

popup-name	The name of the pop-up menu. This name is typically shown in the menu bar. It is a character string in double quotes. The pop-up name also indicates which character in the menu item is to serve as the menu mnemonic. The item text string can also include the following escape codes: \a and \t. The \a escape code aligns the following text flush right. The \t escape code inserts a tab in the item text aligning the text. It should be used only within a POPUP definition and not for items in the main menu bar.
popup-attributes	One or more keywords that describe the pop-up menu's state (active, inactive, grayed, checked, etc.).The keywords GRAYED and INACTIVE are mutually exclusive. Other keywords can be used together by combining their values with a bitwise OR.

GRAYED	Item is unavailable for selection.
INACTIVE	Item is never available for selection.
CHECKED	Checkmark appears next to item. Not supported for top-level menu items.
HELP	Appears on right side of menu bar.
MENUBREAK	Item begins a new menu column.
MENUBARBREAK	Item begins a new menu row.

item-definitions	One or more MENUITEM or POPUP definitions. The MENUITEM definition specifies individual menu items, and the POPUP definition describes a pop-up menu.

Remarks

POPUP is a MENU definition that defines a pop-up menu (also known as a drop-down menu).

POPUP contains a number of MENUITEM definitions, which specify the individual items in the pop-up menu. POPUP can be used only within a MENU definition.

See Also
MENU

PUSHBUTTON

See Also Examples

```
PUSHBUTTON text, control-ID, x, y, width, height, [c-style]
```

Parameters

text	Text string in double quotes. This text is displayed inside the button area. It can be specified as a keyboard accelerator mnemonic.
control-ID	A numeric identifier for this control. This number must be a unique integer. The control ID is used by Windows to indicate which control has been selected.
x, *y*	Horizontal and vertical positions of the button relative to the dialog window in which it appears. These numbers are specified in dialog units.
width, *height*	The width and height of the control. These numbers are specified in dialog units.
c-style	The control style constant (an unsigned long integer value that is interpreted as a series of bit flags). Constants can be combined with a bitwise OR to apply multiple styles to the control. The field can contain any of the window style constants. WS_TABSTOP WS_DISABLED WS_GROUP and any of the BUTTON class style constants. The default styles are WS_TABSTOP and BS_PUSHBUTTON.

Remarks

PUSHBUTTON is a DIALOG definition that places a push button control in a dialog box.

A push button is a square button containing text describing its action. It is a member of the button class. PUSHBUTTON is valid only within a DIALOG definition.

See Also
DIALOG

RADIOBUTTON

See Also Examples

```
RADIOBUTTON text, control-ID, x, y, width, height, [c-style]
```

Parameters

text	Text string in double quotes. This text is displayed next to the button. It can be specified as a keyboard accelerator mnemonic.
control-ID	A numeric identifier for this control. This number must be a unique integer. The control ID is used by Windows to indicate which control has been selected.
x, *y*	Horizontal and vertical positions of the button relative to the dialog window in which it appears. These numbers are specified in dialog units.
width, *height*	The width and height of the control. These numbers are specified in dialog units.
c-style	The control style constant (an unsigned long integer value that is interpreted as a series of bit flags). Constants can be combined with a bitwise OR to apply multiple styles to the control. The field may contain the window style constants: WS_DISABLED WS_GROUP WS_TABSTOP and any of the BUTTON class style constants. The default styles are WS_TABSTOP and BS_RADIOBUTTON.

Remarks

RADIOBUTTON is a DIALOG definition that places a radio button control in a dialog box.

A radio button is a round button with text to the left or right of the button describing the button. Normally, text is displayed to the right of the button. You can change this to the left side with the appropriate *c-style*.

Radio buttons maintain a state: they are either checked or unchecked. A checked radio button has a small black circle within the outer circle to show that it is checked. Radio buttons require application program intervention to maintain their state. To simplify programming, use the auto-radio button which is defined with the CONTROL definition.

A radio button is a member of the button class. When the user presses the mouse button in the control's area, the button is highlighted. When the mouse button is released, the radio button is returns to normal, and a message is sent to the parent window indicating that the button was pressed.

Never perform any action that changes the keyboard focus (or for that matter any action which toggles the state of the application) in response to the BN_CLICKED message, since a radio button sends this notification each time it receives the keyboard focus.

RADIOBUTTON is valid only within a DIALOG definition.

See Also

AUTORADIOBUTTON

DIALOG

RCDATA

See Also Examples

```
resource-name RCDATA [load-type] [memory-option]
BEGIN
  resource-data
END
```

Parameters

resource-name	The text identifier or numeric ID for this resource. The identifier or number must be unique. Numeric IDs must be integers.
load-type	Specifies when the resource is loaded into memory.
memory-option	Specifies how the resource is loaded into memory.
resource-data	One or more lines of data in standard C language format. Data can consist of any mix of numeric values and strings. Numeric values can be represented in hex, octal, or decimal. Strings are placed inside double quotation marks. String values are not automatically null terminated. To terminate a string with a null character, simply include a \0 at the end of the string.

Remarks

RCDATA is a multiple-line statement that lets the user include any type of data directly in an .RC file.

The resource data is associated with the specified resource name, and is included in the executable file for access at run time.

See Also

CHARACTERISTICS

LANGUAGE

VERSION

RCINCLUDE

Examples

```
rcinclude filename
```

Parameters

filename	Can be absolute or relative to the current directory. The INCLUDE environment path is not searched for files which are rcincluded.

Remarks

RCINCLUDE is a directive that causes Resource Workshop to open the named source file and process directives and resource definitions contained in that file.

RCINCLUDE directives can be nested up to 19 levels. Note, however, that the Microsoft Resource Compiler does not support nested RCINCLUDE statements.

The rcinclude keyword is not case sensitive: rcinclude, RCINCLUDE, and RcInClUde are all processed identically by Resource Workshop.

RTEXT

See Also Examples

```
RTEXT text, control-ID, x, y, width, height, [c-style]
```

Parameters

text	A text string, enclosed in quotes, that appears in the dialog window. It can be specified as a keyboard accelerator mnemonic. A common Windows programming technique is to label an edit control by preceding it with a static text control. The label contains a mnemonic for the edit field, such as "&Name". If the CTEXT control does not use the window style constant WS_TABSTOP, Windows sets the focus in the next control.
control-ID	A numeric identifier for this control. This number can be a unique integer. The control ID is used by Windows to indicate which control has been selected. If there is no need to refer to the control at run time (as is most often the case with static controls), this field is by convention set to -1 to document that it is truly static.
x, *y*	Horizontal and vertical positions of the text relative to the dialog window in which the text appears. These numbers are specified in dialog units.
width, *height*	The size of the control in dialog units. The static text appears centered within the specified area.
c-style	The control style constant (an unsigned long integer value that is interpreted as a series of bit flags). More than one control style can be combined using a bitwise OR. The window style WS_GROUP is always set for static text. The default style is the STATIC style constant SS_RIGHT and the window style WS_GROUP.

Remarks

RTEXT is a DIALOG definition that defines right-aligned static text in a dialog box.

RTEXT specifies a text string, its attributes, and where it is located in the window. The text is aligned flush right within the specified rectangle. If the text is too wide to fit within the rectangle, it automatically wraps.

This definition can only appear within a DIALOG definition.

See Also

CTEXT

DIALOG

LTEXT

SCROLLBAR

Examples

```
SCROLLBAR control-ID, x, y, width, height, [c-style]
```

Parameters

control-ID	A numeric identifier for this control. This number must be a unique identifier. The control-ID is used by Windows to indicate which control has been selected.
x, *y*	Horizontal and vertical position of the scroll bar control relative to the dialog window in which it appears. These numbers are specified in dialog units.
width, height	The size of the control in dialog units.
c-style	The control style constant (an unsigned long integer value that is interpreted as a series of bit flags). Constants may be combined with a bitwise OR to apply multiple styles to the control. The field can combine any of the window style constants: WS_DISABLED WS_GROUP WS_TABSTOP and any of the SCROLLBAR class style constants. The default style is SBS_HORZ.

Remarks

SCROLLBAR is a DIALOG definition that defines a scroll bar, a rectangular control with direction arrows on both ends and a movable scroll box.

A scroll bar can appear anywhere in a window. When the user clicks the control with a mouse, a message is sent to the parent window.

STATE3

See Also

```
STATE3 text, control-ID, x, y, width, height [c-style]
```

Parameters

text	Text string in double quotes. It is displayed next to the button. It can be specified as a keyboard accelerator mnemonic.
control-ID	A numeric identifier for this control. This number must be a unique short integer. Windows uses the *control-ID* to indicate which control has been selected.
x, *y*	Horizontal and vertical positions, respectively, of the button relative to the dialog window in which it appears. These numbers are specified in dialog units.
width, *height*	The width and height of the control (specified in dialog units).
c-style	Specifies the styles of the control. This value can be a combination of the BUTTON class style constant BS_3STATE and these window style constants: WS_TABSTOP WS_GROUP The default style is BS_3STATE and WS_TABSTOP.

Remarks

A DIALOG definition that places a 3-state check box control in a dialog box. It's valid only within a DIALOG definition. A 3-state check box is identical to a CHECKBOX control, except that it has three states: checked, unchecked, and "indeterminate" (grayed).

See Also
AUTOCHECKBOX
CONTROL
DIALOG

STRINGTABLE

See Also Examples

```
STRINGTABLE [load-type] [memory-option]
BEGIN
  string-ID, string
END
```

Parameters

load-type	Specifies when the resource is loaded into memory.
memory-option	Specifies how the resource is loaded into memory.
string-ID	A user-assigned integer value that identifies the string. Each string ID must be unique. The string ID is used at run time by the *LoadString* function to determine which string is being requested by the program.
string	An ASCII string in standard C language format.

Remarks

STRINGTABLE is a multiple-line statement that specifies null-terminated ASCII strings that can be accessed by the program.

Each string is assigned a unique unsigned short integer string ID. Strings are read in for access at run time by calling the *LoadString* function with the desired string ID.

The string table mechanism is a convenient method to keep text strings separate from code for easy update and possible translation into foreign languages.

Although string IDs must be numeric, the #define preprocessor directive can be used to simplify access by the program. (See the second example.)

See Also

CHARACTERISTICS

LANGUAGE

MESSAGETABLE

VERSION

User-Defined Resources

Examples

```
resource-name type-ID [load-type] [memory-option] filename
```

Parameters

resource-name	Text identifier or numeric ID for this resource. The identifier or number must be unique within each user-defined resource type. Numeric IDs must be integers.
type-ID	The type-identifier for the custom resource type. This type identifier should be a text identifier or an integer number. User-defined type identifier numbers must be greater than 255. (Numbers 1 to 255 are reserved by Resource Workshop for predefined resource types and future expansion.)
load-type	Specifies when the resource is loaded into memory.
memory-option	Specifies how the resource is loaded into memory.
filename	The name of the DOS file containing the user data. A full path name can be used to specify files which are not in the current working directory. The data in the specified file is included in the current project.

Remarks

A user-defined resource definition associates a file containing user-defined resource data with a resource name and a type ID.

This definition includes data from the specified file in the current project. This definition can also use the syntax of RCDATA.

VERSION

See Also

```
VERSION dword
```

Parameter

dword	A user-defined doubleword value.

Remarks

You use the VERSION statement to specify version information about a resource. The value appears with the resource in the compiled .RES file and can be used by tools that read and write resource-definition files. It is not stored in the executable file and has no significance to Windows.

Use the VERSION statement before the BEGIN statement in these resource definitions:

ACCELERATOR

DIALOG

MENU

RCDATA

STRINGTABLE

The version information applies only to the specified resource.

See Also

CHARACTERISTICS

LANGUAGE

VERSIONINFO

```
versionID VERSIONINFO fixed-info
BEGIN
  block-statements
END
```

The VERSIONINFO resource is a version stamper for Windows 3.1 .EXE files. It is a collection of data used by several Windows API functions (i.e. GetFileVersionInfo, VerInstallFile, and so on) that are typically used by Windows-based installation programs.

Parameters

versionID	Version information resource identifier. Must be set to 1.
fixed-info	Fixed version information.
block-statements	One or more version information blocks. Blocks are either string information or variable information.

String information block:

```
BLOCK "StringFileInfo"
BEGIN
    BLOCK "lang-charset"
    BEGIN
        VALUE "string-name", "value"
            .
            .
            .
    END
END
```

where:

lang-charset	Hex string for language and character set.
string-name	Name of a value in the block.
VALUE	Character string specifying value of corresponding string-name. There can be more than one value statement.

Variable information block:

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

where:

langID/charsetID	Language and character set identifier.There can be more than one identifier pair. Each pair must be separated from the preceding pair with a comma.

Remarks

VERSIONINFO creates a version information resource. The resource contains the:

file version number

product version number

operating system

file type

file function

This resource is intended for use with File Installation functions.

fixed-info

Parameter in the VERSIONINFO statement. Can be set to one of the following:

Statement	Description
FILEVERSION *version*	Binary version number for file. A 64-bit, integer number.
PRODUCTVERSION *version*	Binary version number for product that file is distributed with. A 64-bit, integer number.
FILEFLAGSMASK fileflags	Specifies valid bits in FILEFLAGS statement. If a bit is set, the corresponding bit in FILEFLAGS is valid.
FILEFLAGS fileflags	Boolean attributes of file.
FILEOS fileos	Operating system file designed for.
FILETYPE filetype	File type.
FILESUBTYPE subtype	File function. subtype is 0 unless type parameter in FILETYPE VFT_DRV, VFT_FONT, or VFT_VXD.

fileflags

Parameter used in the VERSIONINFO statement. To use these constants, VER.H must be included in your resource-definition file.

Value	Meaning
VS_FF_DEBUG	Debugging - file contains debugging information or was compiled with debugging features enabled.
VS_FF_INFOINFERRED	Incorrect version information - file contains a dynamically-created version information resource with possible empty or incorrect blocks. Do not use this value in version information resources created with the VERSIONINFO statement.
VS_FF_PATCHED	Patch - file has been modified; is not identical to the original shipping file of the same version number.
VS_FF_PRERELEASE	Pre-release - file is a development version, not a commercially released product.
VS_FF_PRIVATEBUILD	Private build - file was not built using standard release procedures. When you use this value, include the PrivateBuild string in the string-name parameter.
VS_FF_SPECIALBUILD	Special build - file was built by the original company using standard release procedures, but is a variation of the standard file of the same version number. When you use this value, include the SpecialBuild string in the string-name parameter.

fileos

Parameter used in the VERSIONINFO statement. To use these constants, VER.H must be included in your resource-definition file.

Value	Meaning
VOS_UNKNOWN	File designed for unknown operating system.
VOS_DOS	File designed for MS-DOS.
VOS_NT	File designed for Windows NT.
VOS_WINDOWS16	File designed for Windows 3.0 or later.
VOS_WINDOWS32	File designed for Windows 32-bit.
VOS_DOS_WINDOWS16	File designed for Windows 3.0 or later running on MS-DOS.
VOS_DOS_WINDOWS32	File designed for Windows 32-bit running on MS-DOS.
VOS_NT_WINDOWS32	File designed for Windows 32-bit running on Windows NT.

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

filetype

Parameter used in the VERSIONINFO statement. To use these constants, VER.H must be included in your resource-definition file.

Value	Meaning
VFT_UNKNOWN	File type is unknown to Windows.
VFT_APP	File contains an application.
VFT_DLL	File contains a dynamic-link library.
VFT_DRV	File contains a device driver. If you use this constant, include a more specific description of the driver in FILESUBTYPE.
VFT_FONT	File contains a font. If you use this constant, include a more specific description of the font file in FILESUBTYPE.
VFT_VXD	File contains a virtual device. If you use this constant, include a more specific description of the device in FILESUBTYPE.
VFT_STATIC_LIB	File contains a static-link library.

All other values are reserved for use by Microsoft.

subtype

Parameter used in the VERSIONINFO statement. To use these constants, VER.H must be included in your resource-definition file.

subtype can be one of the following values if FILETYPE specifies VFT_DRV:

Value	Meaning
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.

subtype can be one of the following values if FILETYPE specifies VFT_FONT:

Value	Meaning
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.

subtype must be the virtual-device identifier included in the virtual device control block if FILETYPE specifies VFT_VXD.

All values not listed are reserved.

langID

Parameter used in the VERSIONINFO statement. To use these constants, VER.H must be included in your resource-definition file.

langID can be one of the following values:

Value	Language
0x0401	Arabic
0x0402	Bulgarian
0x0403	Catalan
0x0404	Traditional Chinese
0x0405	Czech
0x0406	Danish
0x0407	German
0x0408	Greek
0x0409	U.S. English
0x040A	Castilian Spanish
0x040B	Finnish
0x040C	French
0x040D	Hebrew
0x040E	Hungarian
0x040F	Icelandic
0x0410	Italian
0x0411	Japanese
0x0412	Korean
0x0413	Dutch
0x0414	Norwegian - Bokmal
0x0415	Polish
0x0416	Brazilian Portuguese
0x0417	Rhaeto-Romanic
0x0418	Romanian
0x0419	Russian
0x041A	Croato-Serbian (latin)
0x041B	Slovak
0x041C	Albanian
0x041D	Swedish
0x041E	Thai
0x041F	Turkish
0x0420	Urdu
0x0421	Bahasa
0x0804	Simplified Chinese
0x0807	Swiss German
0x0809	U.K. English
0x080A	Mexican Spanish
0x080C	Belgian French
0x0810	Swiss Italian
0x0813	Belgian Dutch
0x0814	Norwegian - Nynorsk
0x0816	Portuguese
0x081A	Serbo-Croatian (cyrillic)
0x0c0C	Canadian French
0x100C	Swiss French

charsetID

Parameter used in the VERSIONINFO statement. To use these constants, VER.H must be included in your resource-definition file.

charsetID can be one of the following values:

Value	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

Parameter used in the VERSIONINFO statement. To use these constants, VER.H must be included in your resource-definition file.

Name	Value
Comments	Additional information for diagnostic purposes. Optional.
CompanyName	The company that produced the file. Required.
FileDescription	File description. You can display this string in a list box during installation. Required.
FileVersion	File version number. Required.
InternalName	File internal name. If file does not have internal name, use original filename, without extension. Required.
LegalCopyright	File copyright notices. Optional.
LegalTrademarks	Trademarks and registered trademarks that apply to file. Optional.
OriginalFilename	Original file name, not including path. Required.
PrivateBuild	Information about private version of file. Required if VS_FF_PRIVATEBUILD is set in FILEFLAGS.
ProductName	Name of product file is distributed with. Required.
ProductVersion	Version of product file is distributed with. Required.
SpecialBuild	Specifies how this version of file differs from standard version. Required if VS_FF_SPECIALBUILD is set in FILEFLAGS.

Constants

Here are the predefined constants:

Control window style constants

Dialog window style constants

Extended window style constants

Window style constants

Window style constants

See Also

Here are the predefined constants for window styles.

Style	Description
WS_BORDER	Window has a thin border (1 pixel wide on EGA and VGA displays).
WS_CAPTION	Window has a title bar and thin border, unless WS_EX__DLGMODALFRAME is set in *CreateWindowEx* (see DS_MODALFRAME).
WS_CHILD	Window is a child window. All controls within a dialog box have this style. This style cannot be used with WS_POPUP or WS_OVERLAPPED.
WS_CLIPCHILDREN	When the programmer obtains a device context for this window using either the *GetDC* or *BeginPaint* functions, the device context's clipping region excludes the area(s) occupied by any child window(s). If this style is not set for a window, it's possible for a parent to paint over one or more of its child windows. This style is not normally applicable to dialog windows because the user program does not usually paint within a dialog window, leaving that function to the Windows dialog manager.
WS_CLIPSIBLINGS	When the programmer obtains a device context for this window using either the *GetDC* or *BeginPaint* functions, the device context's clipping region excludes the area(s) occupied by any sibling child window(s). This style requires the use of WS_CHILD. If this style is not set for a window, it is possible for a child to paint over one or more of its sibling windows. This style is not normally applicable to dialog controls, because the user program does not usually paint within a dialog window, leaving that function to the Windows dialog manager.
WS_DISABLED	Window is disabled at the time of creation. No user input is allowed in the window, thus the control cannot be selected using the keyboard or mouse. For some controls, this style causes the text for the control to be grayed. This style can be changed at run time by calling the *EnableWindow* function.
WS_DLGFRAME	Window has a double border and no title bar. This style was standard for Windows 2 dialog windows. Windows 3 dialog windows, by convention, use the WS_CAPTION, WS_SYSMENU, and DS_MODALFRAME styles. The Windows 3 convention lets a user move the dialog window in order to see the information underneath.
WS_GROUP	This style is used to group controls for user access with the arrow keys. WS_GROUP is a start of group marker. Only the first control in each group should have the WS_GROUP flag set. The next group begins with the next WS_GROUP. Because this style flag has the same numeric value as WS_MAXIMIZEBOX, it can be used only with child windows.
WS_HSCROLL	Window has a horizontal scroll bar along the bottom of the window. The WS_HSCROLL style should not be confused with a scroll bar control. A scroll bar control is an independent child window which can be moved anywhere within its parent's client area, while the WS_HSCROLL creates a scroll bar that is part of the window's frame (also called non-client area).

WS_ICONIC	Window is created initially in the iconic state. This style should only be used for WS_OVERLAPPED windows.
WS_MAXIMIZE	Window is maximized when first shown. This style should only be used for WS_OVERLAPPED windows.
WS_MAXIMIZEBOX	Window contains a maximize button in the title bar. This style should only be used for windows with the WS_CAPTION style.
WS_MINIMIZE	Window is minimized when first shown. This style should only be used for WS_OVERLAPPED windows.
WS_MINIMIZEBOX	Window contains a minimize button in the title bar. This style should only be used for windows with the WS_CAPTION style.
WS_OVERLAPPED	Window is a top-level window. Usually used for the main window of an application. This style cannot be used with WS_POPUP, or WS_CHILD styles.
WS_OVERLAPPEDWINDOW	Window has WS_OVERLAPPED, WS_CAPTION, WS_SYSMENU, and WS_THICKFRAME attributes.
WS_POPUP	Window is the pop-up type. Cannot be used with WS_CHILD or WS_OVERLAPPED styles.
WS_POPUPWINDOW	Window has WS_POPUP, WS_BORDER, and WS_SYSMENU attributes.
WS_SIZEBOX	Window has a thick frame. Valid only for windows with scroll bars or a title bar.
WS_SYSMENU	Window has a system menu in the title bar. Valid only for windows with title bars (WS_CAPTION style).
WS_TABSTOP	This style indicates that when the user presses Tab in a preceding control, the input focus changes to this control. Controls without this style can still be activated using the mouse, or using the arrow keys, if part of a group.
WS_THICKFRAME	Window has a thick frame and can be sized by the user.
WS_VISIBLE	Window is visible when created. All controls within a dialog have this style by default.
WS_VSCROLL	Window contains a vertical scroll bar at its left side. The WS_VSCROLL style should not be confused with a scroll bar control. A scroll bar control is an independent child window, which can be moved anywhere within its parent's client area, while the WS_VSCROLL style creates a scroll bar that is part of the window's frame (also called non-client area).

See Also

Control window style constants

Dialog window style constants

Extended window style constants

Extended window style constants

See Also

Here are the predefined constants for extended window styles:

Style	Description
WS_EX_ACCEPTFILES	Window accepts drag-drop files.
WS_EX_DLGMODALFRAME	Window has a double border that may (optionally) be created with a title bar by specifying the WS_CAPTION style flag in the *c-style* parameter.
WS_EX_NOPARENTNOTIFY	Specifies that a child window created by using this style will not send the WM_PARENTNOTIFY message to its parent window when the child window is created or destroyed.
WS_EX_TOPMOST	Specifies that a window created with this style should be placed above all non-topmost windows and stay above them even when the window is deactivated. An application can use the **SetWindowPos** function to add or remove this attribute.
WS_EX_TRANSPARENT	Specifies that a window created with this style is to be transparent, so that any windows beneath the window are not obscured by the window. A window created with this style receives WM_PAINT messages only after all sibling windows beneath it have been updated.

See Also
Control window style constants
Dialog window style constants
Window style constants

Dialog window style constants

See Also

Here are the predefined constants for dialog window styles.

Style	Description
DS_ABSALIGN	This style indicates that the x- and y-coordinates specified for the window's position are relative to the screen's origin rather than to the parent or owner window.
DS_SYSMODAL	Makes a system modal window. No other windows can be selected by the user when this window is displayed.
DS_LOCALEDIT	Storage for edit text controls (class "edit") is, by default, allocated in the global heap. By using DS_LOCALEDIT, you can force the Windows dialog manager to create the edit text controls in such a way that the storage is allocated on the local heap of the module whose instance handle is passed to the dialog manager in the call to *DialogBox* or *CreateDialog*. If you want to use the EM_SETHANDLE or EM_GETHANDLE messages in the dialog hook or window function, you must use this style.
DS_SETFONT	This style must be set by a resource compiler for a dialog window that contains the optional font data in the dialog binary data structure. Because it is set by the resource compiler, it must never be used by a user.
DS_MODALFRAME	This style is used in conjunction with WS_CAPTION to create a movable but unsizeable window with a caption and a modal dialog frame for the lower three borders instead of a thin border.
DS_NOIDLEMSG	When a modal dialog box is displayed by an application, Windows periodically sends the dialog's owner window function WM_ENTERIDLE messages. If for some reason, a programmer wants to suppress these messages, use this style.

See Also
Control window style constants
Extended window style constants
Window style constants

Control window style constants

Here are the predefined constants for control window styles.

BUTTON class style constants

COMBOBOX class style constants

EDIT class style constants

LISTBOX class style constants

SCROLLBAR class style constants

STATIC class style constants

BUTTON class style constants

See Also

The BUTTON class includes controls that function like physical buttons. The user "presses" the control using the mouse or keyboard, and the control performs some specific action. Buttons include push buttons, radio buttons, check boxes, and so on.

The predefined constants for the BUTTON class style are:

Style	Description
BS_3STATE	A checkbox with three states: checked, not checked, and grayed. When the user clicks on the control, Windows sends a BN_CLICKED message to the control's parent window indicating that the button has been clicked.
BS_AUTO3STATE	A checkbox with three states: checked, not checked, and grayed. When the user clicks the mouse button on the control, the button automatically toggles to the next state. Windows sends a BN_CLICKED message to the control's parent window indicating that the button has been clicked.
BS_AUTOCHECKBOX	A checkbox with two states: checked and not checked. When the user clicks the mouse button on the control, the button automatically toggles to the next state. Windows sends a BN_CLICKED message to the control's parent window indicating that the button has been clicked.
BS_AUTORADIOBUTTON	A radio button that toggles automatically. When the user clicks on the button, it is checked and other buttons in the group are unchecked. Windows notifies the control's parent window, indicating that the button has been clicked (BN_CLICKED).
BS_CHECKBOX	A checkbox with two states: checked and not checked. When the user clicks on the control, Windows sends a BN_CLICKED message to the control's parent window indicating that the button has been clicked.
BS_DEFPUSHBUTTON	Default push button. This style of control is a push button with a heavy border. The heavy border informs the user that this button is the default action for the window, when the Enter key is pressed. A BN_CLICKED message is sent to the parent window when the user clicks the mouse or presses Spacebar inside the button area. Only one button of this type should be used in a dialog window, and you should be give it the control ID IDOK (1).
BS_LEFTTEXT	This style indicates that the control's text name should be displayed to the left of the button, rather than the default right-hand side. This style can be used with check boxes, BS_3STATE and BS_AUTO3STATE check boxes, and radio buttons.
BS_OWNERDRAW	A button drawn by its owner (parent) window.
BS_PUSHBUTTON	Push button. This style of control is a rectangle containing text, with a rectangular border. A BN_CLICKED message is sent to the parent window when the user clicks the mouse or presses Spacebar inside the button area. With some display drivers the control is partially filled with gray, giving the button a shaded appearance.
BS_RADIOBUTTON	Radio button. When the user clicks on the button, Windows notifies the control's parent window, indicating that the button has been clicked (BN_CLICKED). When using radio buttons, you must use the *CheckRadioButton* function to select and deselect radio buttons within a group when the BN_CLICKED notification is sent

to the parent window or dialog function.

See Also
Control window style constants

COMBOBOX class style constants

See Also

The COMBOBOX class is used for a box that is a combination of a list box and an edit control. The list box can be displayed at all times (CBS_SIMPLE) or dropped by clicking in a drop-down button next to the edit control. The edit control may or may not allow user entry.

The predefined constants for the COMBOBOX class are:

Style	Description
CBS_AUTOHSCROLL	A combo box that scrolls the text in the edit control. When a user enters text beyond the boundary of the rectangle, the text will scroll.
CBS_DISABLENOSCROLL	A combo box with a disabled vertical scroll bar when the list box doesn't contain enough items to scroll. If this style is not used, the scroll bar won't be displayed.
CBS_DROPDOWN	A combo box with a list box displayed only when the user selects the drop-down arrow next to the edit control. The edit control displays the current selection.
CBS_DROPDOWNLIST	A combo box with a list box with a static text item instead of an edit control that displays the current selection. The list box is displayed only when the user selects the drop-down arrow next to the static text.
CBS_HASSTRINGS	An owner-draw combo box with a list box containing items made up of strings. Memory and pointers for the strings are handled by the combo box. Your application can then use CB_GETTEXT to retrieve the text for a particular string.
CBS_OEMCONVERT	A combo box that converts text typed into the edit control from the ANSI character set to the OEM character set and back again to ANSI. When your application calls the *AnsiToOem* function to convert an ANSI string to OEM characters, this style ensures characters are converted properly. Usually this style is used for combo boxes that contain filenames. Use CBS_OEMCONVERT only with combo boxes created with the CBS_SIMPLE or CBS_DROPDOWN styles.
CBS_OWNERDRAWFIXED	An owner-draw fixed-height combo box. The list box's owner draws the contents of the list box. Each list box item is the same height.
CBS_OWNERDRAWVARIABLE	An owner-draw variable-height combo box. The list box's owner draws the contents of the list box. List box items can be of variable heights.
CBS_SIMPLE	A combo box with the list box always displayed. It's edit control displays the list box's current selection.
CBS_SORT	A combo box that sorts all the items displayed in the list box.

See Also
Control window style constants

EDIT class style constants

See Also

The EDIT class is used for editable text fields. These fields are a rectangular area with space for one or more lines of text. The text can be edited by the user with the mouse or keyboard.

The predefined constants for the EDIT class are:

Style	Description
ES_AUTOHSCROLL	Automatic horizontal scroll. When this style is specified and the user enters more text than will fit in the control's rectangle, the text is automatically scrolled. Scrolling occurs 10 characters at a time. If automatic horizontal scrolling is not specified and ES_MULTILINE is specified, text will automatically wrap to the next line when the right side of the control's rectangle is reached. The EM_LIMITTEXT message can be used by to place an absolute limit on the number of characters allowed in the edit text control.
ES_AUTOVSCROLL	Automatic vertical scroll. Text is scrolled vertically when the user presses the Enter key on the last line of text which fits in the control's rectangle. Scrolling occurs one page at a time. (A page is defined as the number of lines visible in the edit text control.) If automatic vertical scrolling is not specified, the control will beep if the user attempts to enter more lines than will fit in the control. Note that the ES_LIMITTEXT message can be used by the programmer to place an absolute limit on the number of characters allowed in the edit text control.
ES_CENTER	Edit text is centered horizontally in control's rectangle. The ES_MULTILINE style must be set for this style to have any effect on the appearance of the text.
ES_LEFT	Edit text is aligned flush left.
ES_LOWERCASE	Edit text is converted to lower case.
ES_MULTILINE	Edit text can occupy more than one line in the control's rectangle. The number of lines allowed by the control is determined by the size of the control's client area. The ES_MULTILINE flag can be used with ES_AUTOVSCROLL and ES_AUTOHSCROLL to make an automatically scrolling window. Additionally, the WS_HSCROLL and WS_VSCROLL styles can be specified to give manual scrolling ability to the control. In such cases, the edit control manages its own scrolling, with no intervention required. If the edit control has no scroll bars, scrolling can be managed by sending the control WM_HSCROLL and WM_VSCROLL messages.
ES_NOHIDESEL	No hide selection. Normally text selected in an edit text control is highlighted only when the control has the input focus. The ES_NOHIDESEL flag specifies that the selected text in edit control remains highlighted even when the control doesn't have the input focus.
ES_OEMCONVERT	Edit text is converted from the ANSI character set to the OEM character set and back again to ANSI. When your application calls the *AnsiToOem* function to convert an ANSI string to OEM characters, this style ensures characters are converted properly. Usually this style is used for combo boxes that contain filenames.
ES_PASSWORD	Edit text is displayed as an asterisk (*) as it's typed in.
ES_READONLY	Edit text is read-only; the user can't enter or edit text.
ES_RIGHT	Edit text is aligned flush right. The text ends at the right side of control's rectangle. Note that the ES_MULTILINE style must be set for this style to have any effect on the appearance of the text.

ES_UPPERCASE	Edit text is converted to upper case.
ES_WANTRETURN	When the user presses Enter, ES_WANTRETURN puts Enter in the edit buffer, rather than performing the action for the default key. This style applies to multiline text only.

See Also
Control window style constants

LISTBOX class style constants

See Also

The LISTBOX class is used for list boxes. These boxes contain a list of strings or application program drawn items from which the user can select. LISTBOX is typically used for lists of file names, fonts, styles, etc.

The predefined constants for the LISTBOX class are:

Style	Description
LBS_DISABLENOSCROLL	A list box with a disabled vertical scroll bar when it doesn't contain enough items to scroll. If this style is not used, the scroll bar won't be displayed.
LBS_EXTENDEDSEL	A list box with multiple items that are selected using the SHIFT key and the mouse or special key combinations.
LBS_HASSTRINGS	An owner-draw list box that contains string items. The list box maintains the memory and pointers for the string items. The LB_GETTEXT message retrieves the text for a particular item.
LBS_MULTICOLUMN	A multicolumn list box that is scrolled horizontally. The LB_SETCOLUMNWIDTH message sets the width of the columns.
LBS_MULTIPLESEL	A list box where more than one string in the list box can be selected.
LBS_NOINTEGRALHEIGHT	A list box that is not resized when partial items are displayed. Its size is exactly the size specified by the application when it created the list box.
LBS_NOREDRAW	The content of the list box is not redrawn when a change is made. This setting can be changed at run time by sending a WM_SETREDRAW message to the control.
LBS_NOTIFY	Notifies the parent window when the user clicks or double clicks in a control.
LBS_OWNERDRAWFIXED	A list box where the owner is responsible for drawing the contents of the list box. The items in the list box are the same height. The owner window receives a WM_MEASUREITEM message when the list box is created. When a visual aspect of the list box changes, the owner window receives a WM_DRAWITEM message .
LBS_OWNERDRAWVARIABLE	A list box where the owner is responsible for drawing the contents of the list obx. The items in the list box are variable in height. The owner window receives a WM_MEASUREITEM message when the list box is created. When a visual aspect of the list box chnages, the owner windows receives a WM_DRAWITEM message.
LBS_SORT	Strings in the list box are displayed in alphabetical order.
LBS_STANDARD	Strings in the list box are displayed in alphabetical order and parent window receives an input message when the user clicks or double-clicks a string. There are borders on all sides of the list box.
LBS_USETABSTOPS	Allows a list box to recognize and expand tab characters when drawing its strings. The default tab positions are 32 dialog units.
LBS_WANTKEYBOARDINPUT	When the list box has the focus and a user presses a key, the owner of the list box receives a WM_VKEYTOITEM or a WM_CHARTOITEM message. The application can perform special processing on the keyboard input.

See Also
Control window style constants

SCROLLBAR class style constants

See Also

The SCROLLBAR class is used for the standard scroll bar control and for size boxes. Scroll bar controls have the same appearance as a window's scroll bar, except they can be positioned anywhere within a window.

The predefined constants for the SCROLLBAR class are:

Style	Description
SBS_BOTTOMALIGN	The scroll bar appears the standard height, aligned with the bottom side of the control's rectangle. Valid for horizontal scroll bars only.
SBS_HORZ	Horizontal scroll bar. If SBS_BOTTOMALIGN or SBS_TOPALIGN is not specified, the scroll bar occupies the control's client area.
SBS_LEFTALIGN	The scroll bar appears the standard width, aligned with the left side of the control's rectangle. Valid for vertical scroll bars only.
SBS_RIGHTALIGN	The scroll bar appears the standard width, aligned with the right side of the control's rectangle. Valid for vertical scroll bars only.
SBS_SIZEBOXTOPLEFTALIGN	The size box appears the standard size, aligned in the upper-left corner of the control's rectangle. Valid for size boxes only.
SBS_SIZEBOX	Size box. If SBS_SIXEBOXTOPLEFTALIGN or SBS_SIZEBOXBOTTOMRIGHTALIGN is not specified, the size box occupies the control's client area.
SBS_SIZEBOXBOTTOMRIGHTALIGN	The size box appears the standard size, aligned in the lower-right corner of the control's rectangle. Valid for size boxes only.
SBS_TOPALIGN	The scroll bar appears the standard height, aligned with the top side of the control's rectangle. Valid for horizontal scroll bars only.
SBS_VERT	Vertical scroll bar. If SBS_RIGHTALIGN or SBS_LEFTALIGN is not specified, the scroll bar occupies the control's client area.

See Also
Control window style constants

STATIC class style constants

See Also

The STATIC class is used for static controls. These controls include non-editable text and icons.

The predefined constants for the STATIC class are:

Style	Description
SS_BLACKFRAME	Static item's rectangle; displayed with a frame in the system color COLOR_WINDOWFRAME.
SS_BLACKRECT	Static item's rectangle; filled with the system color COLOR_WINDOWFRAME.
SS_CENTER	Static item is text displayed centered in the control's rectangle. Text too long to fit is automatically wrapped to the next line.
SS_GRAYRECT	Static item's rectangle; filled with the system color COLOR_BACKGROUND.
SS_GRAYFRAME	Static item's rectangle; displayed with a frame in the system color COLOR_BACKGROUND.
SS_LEFT	Static item is text displayed left-aligned in the control's rectangle. Text too long to fit is automatically wrapped to the next line.
SS_RIGHT	Static item is text displayed right-aligned in the control's rectangle. Text too long to fit is automatically wrapped to the next line.
SS_WHITEFRAME	Static item's rectangle; displayed with a frame in the system color COLOR_WINDOW.
SS_WHITERECT	Static item's rectangle is filled with the system color COLOR_WINDOW.

See Also
Control window style constants

load-type

load-type specifies when resources are loaded into memory.

Use the following keywords for *load-type*. LOADONCALL is the default load type if no keyword is specified.

Keyword	Value
PRELOAD	Resource loaded at program start.
LOADONCALL	Resource is loaded when referenced by the application.

memory-option specifies *how* resources are loaded into memory.

memory-option

memory-option specifies how resources are loaded into memory.

Use the following keywords for *memory-option*. The resource can be fixed at the same address at which *memory-option* is loaded, or it can be relocatable.

Additionally,the resource can be discardable or nondiscardable. MOVEABLE and DISCARDABLE are the default values.

Keyword	**Value**
DISCARDABLE	Can be purged to make space.
FIXED	Resource stays at same address.
IMPURE	Resource is modified after loading.
MOVEABLE	Resource can be relocated in memory.
NONDISCARDABLE	Must stay in memory.
PURE	Resource is not modified after it is loaded.

load-type specifies when resources are loaded into memory.

Directives

Overview

Directives are special statements that affect how the resource script file is compiled. They do not define resources. Unlike other resource script statements, directives are case sensitive.

You can use directives to:

- assign values to names you use in other resource script statements
- include the contents of other files
- specify how the script file should be compiled

If you are a C or C++ programmer, the directive statements will look familiar to you. They are identical to those you use when compiling your programs.

Here are the directives:

Directive	Description
#define	Assigns a given value to the name you specify.
#elif	Controls conditional compilation and marks an optional block. Used with #if.
#else	Controls conditional compilation and marks an optional block. Used with #if, #ifdef, #ifndef, and #elif.
#endif	Controls conditional compilation and marks the end of a block. Used with #if, #ifdef, #ifndef, and #elif.
#error	Displays user-defined error message.
#if	Marks the start of conditional compilation.
#ifdef	Marks the start of conditional compilation if a specified name has been #defined.
#ifndef	Marks the start of conditional compilation if a specified name has not been #defined.
#include	Puts the contents of the named file into your resource script before it's compiled.
#line	Ignores text on the specified line number.
#pragma	Directive is ignored.
RCINCLUDE	Puts the contents of the named file into your resource script before it's compiled.

#define

See Also Examples

```
#define identifier text
```

Parameters

identifier	Only recognized and replaced by the compiler when it is followed by whitespace and occurs outside a quoted string.
text	Can contain any valid character. Is normally terminated by the end of the line on which the #define occurs. *text* can also be empty. To continue *text* onto another line, place a backslash character (\) at the end of the source line.

Remarks

#define is a directive that causes the compiler to substitute text for each subsequent reference to *identifier* in the resource source text. Both the #define directive and the identifier are case sensitive.

There are two #define identifiers that are always defined in Resource Workshop:

RC_INVOKED

WORKSHOP_INVOKED

Testing for these values in an #if or #ifdef directive always yields 1.

The #define directive is a subset of the #define statement.

See Also

#if

#ifdef

#elif

See Also Examples

```
#elif constant-expression
```

Parameters

constant-expression	Evaluated as a signed long integer. If *constant-expression* evaluates to a nonzero value, the text following the #elif directive is processed by the compiler. Otherwise, the text between the #elif and a following #elif or #endif directive is ignored by the compiler.

Remarks

#elif is a directive that is used in conjunction with the #if directive to control conditional compilation. This directive is lost if it occurs within a text resource and that resource is edited by Resource Workshop.

The #elif keyword is case sensitive; #ELIF causes a compile error.

See Also

#define

#if

#endif

#else

See Also Examples

```
#else
```

Remarks

#else is a directive that is used in conjunction with the #if, #ifdef, #ifndef, and #elif directives to control conditional compilation.

The compiler processes source code following the #else directive until the next #endif directive only when none of the previous conditional directives within the current scope have been processed.

If any previous conditional directives within the current scope have been processed, the source code between the #else and the following #endif will be ignored. This directive is lost if it occurs within a text resource and that resource is edited by Resource Workshop.

The #else keyword is case sensitive; #ELSE causes a compile error.

See Also

#define

#elif

#endif

#if

#ifdef

#ifndef

#endif

See Also   Examples

```
#endif
```

Remarks

#endif is a directive that marks the end of the scope of the current conditional compilation. It is used in conjunction with the #if, #ifdef, #ifndef, #else, and #elifelif directives to control conditional compilation.

This directive is lost if it occurs within a text resource and that resource is edited by Resource Workshop.

The #endif keyword is case sensitive; #ENDIF will cause a compile error.

See Also

#elif

#else

#if

#ifdef

#ifndef

#error

```
Error: filename line# : #error directive encountered: error text
```

Remarks

#errors is a directive that displays a user-defined error message.

This directive usually catches an undesired compile-time condition. In the normal case, the condition would be false. If the condition is true, you want the compiler to print an error message and stop the compilation. You do this by placing an #error directive in a conditional that is true for the undesired case.

#if

See Also Examples

```
#if constant expression
#if [!]defined identifier
#if [!]defined (identifier)
```

Parameters

When #if is followed by *constant_expression*, the constant expression is evaluated as a signed long integer:

If the resulting value is nonzero, the source text following the #if directive until the next #else, #elif, or #endif directive is processed by the compiler.

If the resulting value is zero, the source text following the #if directive until the next #else, #elif, or #endif directive is ignored by the compiler.

The other formats of the #if directive can be used in place of the #ifdef and #ifndef directives. The keyword must be lowercase, but it can be preceded by a ! operator. The following *identifier* (optionally surrounded by parentheses) is searched for in the compiler's table of #defines. If the #define is found, the source text following the #if directive is processed. The ! operator reverses the effect of the preceding test.

Remarks

#if is a directive that marks the start of conditional compilation. #if directives can be nested. This directive is lost if it occurs within a text resource and that resource is edited by Resource Workshop.

The #if keyword is case sensitive; #IF will cause a compile error.

See Also

#define

#elif

#else

#endif

#ifdef

#ifndef

#ifdef

See Also Examples

```
#ifdef identifier
```

Parameters

identifier	Searched for in the compiler's table of #defines.

- If *identifier* is found, the source text following the #ifdef directive until the next #else, #elif, or #endif is processed.
- If *identifier* is not found, the source text following the #ifdef directive until the next else, #elif, or #endif is ignored.

Remarks

#ifdef is a directive that marks the start of conditional compilation.

The #ifdef keyword is case sensitive; #IFDEF will cause a compile error.

See Also

#define

#elif

#else

#endif

#ifndef

See Also Examples

```
#ifndef identifier
```

Parameters

identifier	Searched for in the compiler's table of #defines.

- If *identifier* is found, the source text following the #ifndef directive until the next #else, #elif, or #endif is ignored.
- If *identifier* is not found, the source text following the #ifdef directive until the next #else, #elif, or #endif is processed.

Remarks

#ifndef is a directive that marks the start of conditional compilation.

The #ifndef keyword is case sensitive; #IFNDEF will cause a compile error.

See Also

#define

#elif

#else

#endif

#ifdef

#include

See Also Examples

```
#include "filename"
#include <filename>
```

Parameters

filename	Can be absolute, relative to the current directory, or in one of the directories identified by the INCLUDE environment variable. Must be surrounded by either a pair of double-quote characters (`""`) or a left angle bracket (<) and a right angle bracket (>). Resource Workshop treats both forms identically to maintain compatibility with the Microsoft RC compiler. Also to maintain compatibility, paired backslash characters (`"\\"`) are treated as a single backslash.

Remarks

#include is a directive that causes the Resource Workshop compiler to open the named source file and process directives and resource definitions contained in that file.

Only directives of the format #*...* are processed in a header or C source file that is #included. All other data (for example, comments, structure definitions, code, resource definitions) within #included files are ignored.

This prevents Resource Workshop from seeing or processing RCINCLUDE directives in an #included file in header and C source files.

You can also #include Pascal constants in a Pascal include file or in a unit. The file must not contain any data other than constants.

The #include keyword is case sensitive; #INCLUDE will cause a compile error.

See Also
RCINCLUDE

#line

```
#line integer_constant
```

Remarks

#line is a directive that causes the Resource Workshop compiler to ignore text on the specified line.

#pragma

The #pragma directive is ignored by the Resource Workshop compiler.

#undef

```
#undef identifier
```

Parameters

identifier	Searched for in the compiler's table of #defines.

Remarks

Resource Workshop has limited support for the #undef preprocessor directive. You can use it only with #defines that are not referenced by a resource.

If you use #undef with a #define that's a resource identifier, you get a fatal compiler error when compiling the RC file under Resource Workshop.

The #undef preprocessor directive applies only to the C language. If you're using Resource Workshop with a Pascal compiler, you can disregard this.

#define example

See Also

After you compile this example, *SelectAll* and *IDS_SAMPLE* have the value 1. *IDS_ERROR* has the value 2.

```
#define SelectAll  1
#define IDS_SAMPLE 1
#define IDS_ERROR  IDS_SAMPLE + 1
#define MYCAPTION  "hello"
```

See Also
#define

#elif example

See Also

This example causes #define *A* to have the value 2.

```
#if (1 == 0)
#define A 1
#elif (1 == 1)
#define A 2
#endif
```

See Also
#elif

#else example

See Also

This example causes #define *A* to have the value 3.

```
#if (1 == 0)
#define A 1
#elif (1 == 3)
#define A 2
#else
#define A 3
#endif
```

See Also
#else

#endif example

See Also

#endif in this example marks the end of the scope of conditional compilation.

```
#if (1==0)
#define A 1
#elif (1==1)
#define A 2
#endif
```

See Also
#endif

#if example

See Also

After you compile this example, neither *A* nor *B* will be defined.

```
#if (1 == 0)
#define A 1
#endif
#if defined A
#define B 2
#endif
```

See Also

#if

#ifdef example

See Also

The #define *NOATOM* will be defined after you compile this example.

```
#define RC_INVOKED
#ifdef  RC_INVOKED
#define NOATOM
#endif
```

See Also
#ifdef

#ifndef example

See Also

After you compile this example, the file WINDOWS.H will not be included by Resource Workshop.

```
#ifndef  WORKSHOP_INVOKED
#include <windows.h>
#endif
```

See Also
#ifndef

#include examples

See Also

These examples show you how to #include C identifiers:

```
#include "w3demo.h"
#include <project.h>
```

These examples show you how to #include Pascal identifiers:

```
#include "MYCONSTS.INC"
#include <PROJECT.PAS>
```

See Also
#include

ACCELERATOR examples

See Also

Example 1

This example shows one way to define Ctrl+A (in hexadecimal), Ctrl+P, and Shift+Q as accelerators.

```
myaccel ACCELERATORS
BEGIN
  0x41,1, CONTROL, NOINVERT
  "^P",2
  VK_Q,3, VIRTKEY, SHIFT
END
```

If you want to use identifiers rather than numbers to identify your accelerators, use a C-style header file (with an .H extension) or a Pascal include file (with an .INC extension). For Pascal, you can also keep your identifiers in a unit.

Example 2

This example uses #defines in the file KEYNAMES.H.

```
#define SelectAll  1
#define PrintNow   2
#define Quit       3
```

Then your script statements in the .RC file can use identifiers rather than numbers to uniquely identify each accelerator:

```
#include "keynames.h"
myaccel ACCELERATORS
BEGIN
  "^a", SelectAll, NOINVERT
  0x50, PrintNow, CONTROL
  "Q",  Quit
END
```

See Also
ACCELERATOR

BITMAP examples

See Also

The bitmap "image" is in IMAGEF.BMP:

```
image BITMAP imagef.bmp
```

The bitmap 200 is in FRAME.BMP. It is loaded when the application starts:

```
200 BITMAP PRELOAD frame.bmp
```

The bitmap "bigstar" is in STAR.BMP. It cannot be moved around in memory:

```
bigstar BITMAP FIXED star.bmp
```

See Also
BITMAP

CURSOR examples

See Also

Example 1

This example names the cursor resource 20 and tells Resource Workshop that the resource is found in the file HAND.CUR.

```
20 CURSOR hand.cur
```

Example 2

This example names the resource "paintcan" and tells Resource Workshop it can be found in the file PAINT.CUR. The resource is loaded into memory when the application begins. It must remain in the same place in memory while the application runs. The application can't remove it from memory before the application ends.

```
paintcan CURSOR PRELOAD FIXED NONDISCARDABLE paint.cur
```

See Also
CURSOR

DEFPUSHBUTTON example

See Also

This example creates a default push button control labeled "OK." Its control ID is 104. The coordinates of the upper-left corner of the dialog box are 11, 71. The button is 24 dialog units wide and 14 high.

The button class style constant BS_DEFPUSHBUTTON indicates that the button is a default push button. WS_TABSTOP indicates that when the user presses Tab in a preceding control, the input focus changes to this control.

```
DEFPUSHBUTTON "OK", 104, 11, 71, 24, 14, BS_DEFPUSHBUTTON | WS_TABSTOP
```

See Also
DEFPUSHBUTTON

DIALOG Examples

Dialog Box with a Combo Box

Dialog Box with a Group Box and Radio Buttons

Dialog Box with a List Box and Check Boxes

Dialog Box with a Scrollbar and Text Entry Field

Dialog Box with Borland Windows Custom Controls

DIALOG, RADIOBUTTON and GROUPBOX example

See Also   View Resource   Implementation

This example shows how to use the DIALOG statement to display a dialog box that asks the user to select an option using radio buttons:

Resource Script

```
DLG_DIRECTION DIALOG 71, 65, 143, 65
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
CAPTION "Choose a Direction"
FONT 8, "MS Sans Serif"
{
 DEFPUSHBUTTON "OK", IDOK, 87, 8, 50, 14
 PUSHBUTTON "Cancel", IDCANCEL, 87, 26, 50, 14
 PUSHBUTTON "Help", IDHELP, 87, 44, 50, 14
 AUTORADIOBUTTON "&Forward", 2, 13, 20, 44, 12, BS_RADIOBUTTON
 AUTORADIOBUTTON "&Backward", 3, 13, 33, 44, 12, BS_RADIOBUTTON
 GROUPBOX "Direction", IDC_DIRECTION, 5, 8, 67, 42, BS_GROUPBOX
}
```

See Also

CAPTION

CLASS

FONT

MENU

STYLE

Implementation

To use this resource in a Windows application, use one of these functions or classes:

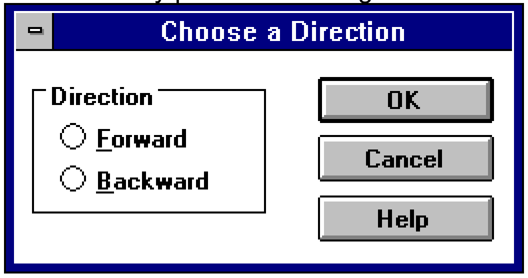
TDialog (ObjectWindows)

DialogBox() (Windows API)

Close

DIALOG example (displayed resource)

Click on any part of the dialog box to view the resource script behind that element:



```
DLG_DIRECTION DIALOG 71, 65, 143, 65
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
CAPTION "Choose a Direction"
```

```
DEFPUSHBUTTON "OK", IDOK, 87, 8, 50, 14
```

```
PUSHBUTTON "Cancel", IDCANCEL, 87, 26, 50, 14
PUSHBUTTON "Help", IDHELP, 87, 44, 50, 14
```

```
AUTORADIOBUTTON "&Forward", 2, 13, 20, 44, 12, BS_RADIOBUTTON
AUTORADIOBUTTON "&Backward", 3, 13, 33, 44, 12, BS_RADIOBUTTON
```

```
GROUPBOX "Direction", IDC_DIRECTION, 5, 8, 67, 42, BS_GROUPBOX
```

DIALOG, CHECKBOX, LISTBOX and LTEXT example

See Also View Resource Implementation

This example shows how to use the DIALOG statement to display a dialog box that asks the user to select an item from a list box:

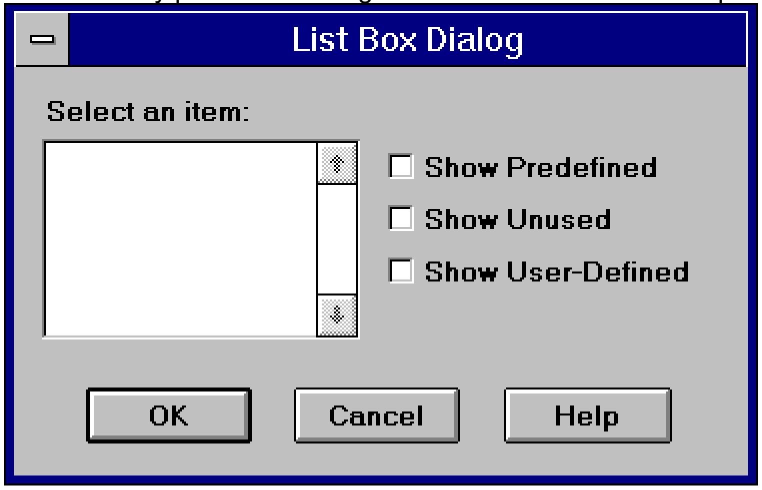
Resource Script

```
ListBox DIALOG 16, 24, 163, 102
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
CAPTION "List Box Dialog"
FONT 8, "MS Sans Serif"
{
  DEFPUSHBUTTON "OK", IDOK, 16, 80, 37, 14
  PUSHBUTTON "Cancel", IDCANCEL, 62, 80, 37, 14
  PUSHBUTTON "Help", IDHELP, 109, 80, 37, 14
  LISTBOX 100, 7, 19, 69, 50, LBS_STANDARD | LBS_MULTIPLESEL
  LTEXT "Select an item:", 100, 7, 7, 48, 8, SS_LEFTNOWORDWRAP | WS_GROUP
  AUTOCHECKBOX "Show Predefined", IDC_SHOWPREDEFINED, 83, 19, 70, 12
  AUTOCHECKBOX "Show Unused", IDC_SHOWUNUSED, 83, 32, 70, 12
  AUTOCHECKBOX "Show User-Defined", IDC_USERDEFINED, 83, 45, 70, 12
}
```

Close

DIALOG example (displayed resource)

Click on any part of the dialog box to view the resource script behind that element:



```
ListBox DIALOG 16, 24, 163, 102
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
CAPTION "List Box Dialog"
```

```
DEFPUSHBUTTON "OK", IDOK, 16, 80, 37, 14
```

```
PUSHBUTTON "Cancel", IDCANCEL, 62, 80, 37, 14
PUSHBUTTON "Help", IDHELP, 109, 80, 37, 14
```

```
LISTBOX 100, 7, 19, 69, 50, LBS_STANDARD | LBS_MULTIPLESEL
```

```
LTEXT "Select an item:", 100, 7, 7, 48, 8, SS_LEFTNOWORDWRAP | WS_GROUP
```

```
AUTOCHECKBOX "Show Predefined", IDC_SHOWPREDEFINED, 83, 19, 70, 12
AUTOCHECKBOX "Show Unused", IDC_SHOWUNUSED, 83, 32, 70, 12
AUTOCHECKBOX "Show User-Defined", IDC_USERDEFINED, 83, 45, 70, 12
```

DIALOG, COMBOBOX and CTEXT example

See Also View Resource Implementation

This example shows how to use the DIALOG statement to display a dialog box that asks the user to select a value from a combo box:

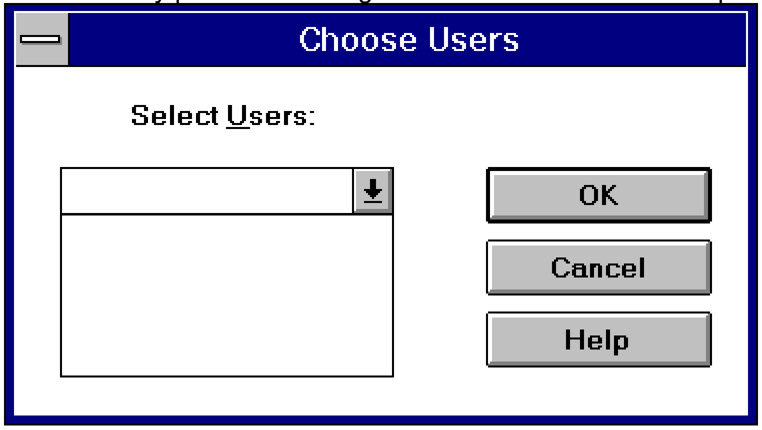
Resource Script

```
ChooseUsers DIALOG 35, 35, 163, 87
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
CAPTION "Choose Users"
FONT 8, "MS Sans Serif"
{
  DEFPUSHBUTTON "OK", IDOK, 105, 25, 50, 14
  PUSHBUTTON "Cancel", IDCANCEL, 105, 43, 50, 14
  PUSHBUTTON "Help", IDHELP, 105, 61, 50, 14
  COMBOBOX IDC_Users, 10, 25, 74, 55, CBS_DROPDOWNLIST | WS_TABSTOP
  CTEXT "Select &Users:", -1, 17, 8, 60, 8
}
```

Close

DIALOG example (displayed resource)

Click on any part of the dialog box to view the resource script behind that element:



```
ChooseUsers DIALOG 35, 35, 163, 87
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
CAPTION "Choose Users"
```

```
DEFPUSHBUTTON "OK", IDOK, 105, 25, 50, 14
```

```
PUSHBUTTON "Cancel", IDCANCEL, 105, 43, 50, 14
PUSHBUTTON "Help", IDHELP, 105, 61, 50, 14
```

```
COMBOBOX IDC_Users, 10, 25, 74, 55, CBS_DROPDOWNLIST | WS_TABSTOP
```

```
CTEXT "Select &Users:", -1, 17, 8, 60, 8
```

DIALOG, SCROLLBAR, EDITTEXT and CONTROL example

See Also View Resource Implementation

This example shows how to use the DIALOG statement to display a dialog box that asks the user to select a value using a scrollbar and edit field:

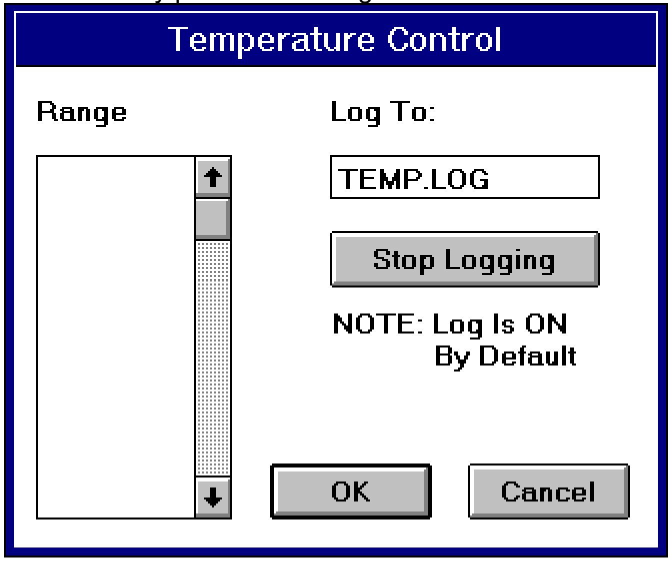
Resource Script

```
TempControl DIALOG 16, 27, 143, 120
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION
CAPTION "Temperature Control"
FONT 8, "MS Sans Serif"
{
  DEFPUSHBUTTON "OK", IDOK, 57, 99, 36, 14
  PUSHBUTTON "Cancel", IDCANCEL, 101, 99, 36, 14
  SCROLLBAR IDC_VSCROLL1, 40, 22, 9, 91, SBS_VERT | WS_GROUP
  LTEXT "Range", -1, 5, 7, 22, 8
  CONTROL "", -1, "static", SS_BLACKFRAME | WS_CHILD | WS_VISIBLE, 5, 22,
36, 91
  LTEXT "Log To:", -1, 70, 7, 27, 8
  EDITTEXT IDC_LOGNAME, 70, 22, 60, 11, WS_BORDER | WS_TABSTOP
  CONTROL "Stop Logging", IDC_STOPLOG, "BUTTON", BS_PUSHBUTTON, 70, 41, 60,
14
  RTEXT "NOTE: Log Is ON By Default", -1, 67, 60, 58, 23
}
```

Close

DIALOG example (displayed resource)

Click on any part of the dialog box to view the resource script behind that element:



```
TempControl DIALOG 16, 27, 143, 120
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION
CAPTION "Temperature Control"
```

```
DEFPUSHBUTTON "OK", IDOK, 57, 99, 36, 14
```

```
PUSHBUTTON "Cancel", IDCANCEL, 101, 99, 36, 14
```

```
SCROLLBAR IDC_VSCROLL1, 40, 22, 9, 91, SBS_VERT | WS_GROUP
```

```
LTEXT "Range", -1, 5, 7, 22, 8
LTEXT "Log To:", -1, 70, 7, 27, 8
```

```
EDITTEXT IDC_LOGNAME, 70, 22, 60, 11, WS_BORDER | WS_TABSTOP
```

```
CONTROL "Stop Logging", IDC_STOPLOG, "BUTTON", BS_PUSHBUTTON, 70, 41, 60, 14
```

```
RTEXT "NOTE: Log Is ON By Default", -1, 67, 60, 58, 23
```

DIALOG example (BWCC)

See Also View Resource Implementation

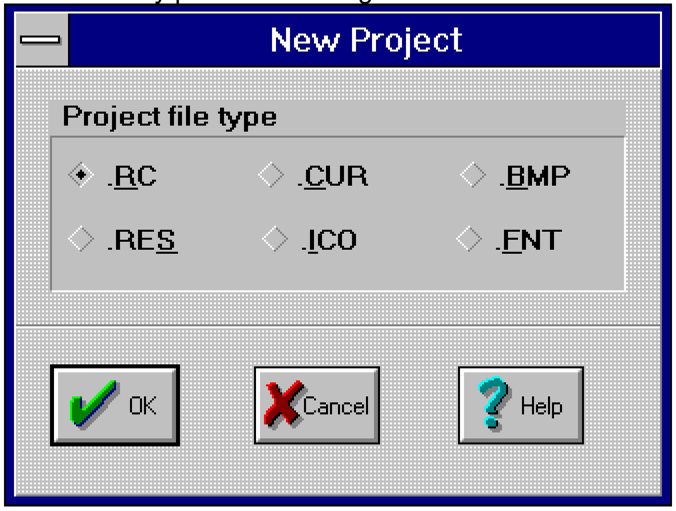
This example shows how to use the DIALOG statement to display a Borland-style dialog box that uses Borland Windows Custom Controls:

Resource Script

```
BWCC_Dialog DIALOG PRELOAD MOVEABLE DISCARDABLE 20, 20, 144, 107
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CLASS "BorDlg"
CAPTION "New Project"
FONT 8, "Helv"
{
  CONTROL "Project file type", -1, "BorShade", BSS_GROUP | BSS_CAPTION |
BSS_LEFT | WS_CHILD | WS_VISIBLE | WS_GROUP, 8, 8, 128, 48
  CONTROL ".&RC", 151, "BorRadio", BS_AUTORADIOBUTTON | WS_CHILD |
WS_VISIBLE | WS_GROUP | WS_TABSTOP, 12, 21, 33, 12
  CONTROL ".RE&S", 154, "BorRadio", BS_AUTORADIOBUTTON | WS_CHILD |
WS_VISIBLE, 12, 37, 33, 12
  CONTROL ".&CUR", 158, "BorRadio", BS_AUTORADIOBUTTON | WS_CHILD |
WS_VISIBLE, 54, 21, 33, 12
  CONTROL ".&ICO", 159, "BorRadio", BS_AUTORADIOBUTTON | WS_CHILD |
WS_VISIBLE, 55, 37, 33, 12
  CONTROL ".&BMP", 160, "BorRadio", BS_AUTORADIOBUTTON | WS_CHILD |
WS_VISIBLE, 99, 21, 33, 12
  CONTROL ".&FNT", 162, "BorRadio", BS_AUTORADIOBUTTON | WS_CHILD |
WS_VISIBLE, 98, 37, 33, 12
  CONTROL "", -1, "BorShade", BSS_HDIP | BSS_LEFT | WS_CHILD | WS_VISIBLE,
0, 64, 144, 2
  CONTROL "", 1, "BorBtn", BS_DEFPUSHBUTTON | WS_CHILD | WS_VISIBLE |
WS_GROUP | WS_TABSTOP, 8, 74, 37, 25
  CONTROL "", 2, "BorBtn", BS_PUSHBUTTON | WS_CHILD | WS_VISIBLE |
WS_TABSTOP, 53, 74, 37, 25
  CONTROL "", 998, "BorBtn", BS_PUSHBUTTON | WS_CHILD | WS_VISIBLE |
WS_TABSTOP, 98, 74, 37, 25
}
```

DIALOG example (displayed resource)

Click on any part of the dialog box to view the resource script behind that element:



```
BWCC_Dialog DIALOG PRELOAD MOVEABLE DISCARDABLE 20, 20, 144, 107
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CLASS "BorDlg"
CAPTION "New Project"
```

```
CONTROL "Project file type", -1, "BorShade", BSS_GROUP | BSS_CAPTION |
BSS_LEFT | WS_CHILD | WS_VISIBLE | WS_GROUP, 8, 8, 128, 48
CONTROL ".&RC", 151, "BorRadio", BS_AUTORADIOBUTTON | WS_CHILD | WS_VISIBLE
| WS_GROUP | WS_TABSTOP, 12, 21, 33, 12
CONTROL ".RE&S", 154, "BorRadio", BS_AUTORADIOBUTTON | WS_CHILD |
WS_VISIBLE, 12, 37, 33, 12
CONTROL ".&CUR", 158, "BorRadio", BS_AUTORADIOBUTTON | WS_CHILD |
WS_VISIBLE, 54, 21, 33, 12
CONTROL ".&ICO", 159, "BorRadio", BS_AUTORADIOBUTTON | WS_CHILD |
WS_VISIBLE, 55, 37, 33, 12
CONTROL ".&BMP", 160, "BorRadio", BS_AUTORADIOBUTTON | WS_CHILD |
WS_VISIBLE, 99, 21, 33, 12
CONTROL ".&FNT", 162, "BorRadio", BS_AUTORADIOBUTTON | WS_CHILD |
WS_VISIBLE, 98, 37, 33, 12
```

```
CONTROL "", -1, "BorShade", BSS_HDIP | BSS_LEFT | WS_CHILD | WS_VISIBLE, 0,
64, 144, 2
```

```
CONTROL "", 1, "BorBtn", BS_DEFPUSHBUTTON | WS_CHILD | WS_VISIBLE | WS_GROUP
| WS_TABSTOP, 8, 74, 37, 25
```

```
CONTROL "", 2, "BorBtn", BS_PUSHBUTTON | WS_CHILD | WS_VISIBLE | WS_TABSTOP,
53, 74, 37, 25
```

```
CONTROL "", 998, "BorBtn", BS_PUSHBUTTON | WS_CHILD | WS_VISIBLE |
WS_TABSTOP, 98, 74, 37, 25
```

FONT example

See Also

Example 1

This example identifies a font as 2 and tells Resource Workshop the font is in the file VETICA.FON.

```
2 FONT vetica.fon
```

Example 2

This example identifies a font as 27 and tells the Resource Workshop the font is in the file BOOK.FON. The font resource is loaded when the application starts and remains in memory until the application ends:

```
27 FONT PRELOAD NONDISCARDABLE book.fon
```

See Also

FONT

ICON examples

See Also

Example 1

This Type 1 example names an icon "myicon" and tells Resource Workshop to look for it in the file MYICON.ICO:

```
myicon ICON myicon.ico
```

Example 2

This Type 2 example creates an icon named "myicon." The icon will be defined elsewhere in the resource file with a Type 1 icon statement. The icon's control ID is 120, and the upper-left corner coordinates are 30, 40. Notice that there are no width and height arguments.

```
ICON "myicon" 120, 20, 20
```

See Also
ICON

MENU examples

Application main menu

Default File menu

Menu with secondary menu

MENU, MENUITEM and POPUP example

See Also View Resource Implementation

This example specifies the the File menu of an application. File is a pop-up menu (commonly known as a drop-down menu) with several options, each defined with a MENUITEM substatement.

Resource Script

The ampersand (&) underlines the letter that immediately follows it. This lets the user choose the command from the menu by typing that letter.

```
mainmenu MENU PRELOAD
BEGIN
  POPUP "&File"
  BEGIN
    MENUITEM "&New", 100
    MENUITEM "&Open", 101
    MENUITEM "&Close", 102, GRAYED
    MENUITEM "&Save",103
    MENUITEM "Save &As", 104
    MENUITEM SEPARATOR
    MENUITEM "&Print", 105
    MENUITEM "&Draft Printing", 107, CHECKED
    MENUITEM SEPARATOR
    MENUITEM "E&xit", 106
  END
  POPUP "&Help"
END
```

Implementation

To use this resource in a Windows application, use one of these functions or classes:

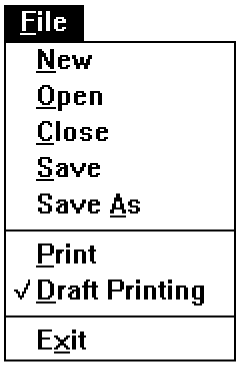
TMenu (ObjectWindows)

CreateMenu (Windows API)

CreatePopupMenu (Windows API)

MENU, MENUITEM and POPUP example (displayed resource)

Click on any part of the menu to view the resource script behind that element:



See Also

MENU

MENUITEM

POPUP

```
POPUP "&File"
```

```
MENUITEM "&New", 100
MENUITEM "&Open", 101
MENUITEM "&Close", 102, GRAYED
MENUITEM "&Save",103
MENUITEM "Save &As", 104
MENUITEM SEPARATOR
MENUITEM "&Print", 105
MENUITEM "&Draft Printing", 107, CHECKED
MENUITEM "E&xit", 106
```

MENU, MENUITEM and POPUP example

See Also View Resource Implementation

This example specifies the Options menu of an application. Options is a pop-up menu (commonly known as a drop-down menu) with a secondary menu.

Resource Script

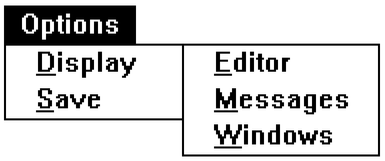
The ampersand (&) underlines the letter that immediately follows it. This lets the user choose the command from the menu by typing that letter.

```
OptionsMenu MENU
{
  POPUP "Options"
  {
    POPUP "&Display"
    {
      MENUITEM "&Editor", 3101
      MENUITEM "&Messages", 3102
      MENUITEM "&Windows", 3103
    }
    MENUITEM "&Save", 3200
  }
}
```

Close

MENU, MENUITEM and POPUP example (displayed resource)

Click on any part of the menu to view the resource script behind that element:



```
POPUP "Options"
POPUP "&Display"
```

```
MENUITEM "&Editor", 3101
MENUITEM "&Messages", 3102
MENUITEM "&Windows", 3103
MENUITEM "&Save", 3200
```

MENU, MENUITEM and POPUP example

See Also View Resource Implementation

This example specifies the main menu of an application. The main menu contains the default File, Edit, and Help menus. Each of these menus are defined as pop-up menus (commonly known as drop-down menus) with several options, each defined with a MENUITEM substatement.

Resource Script

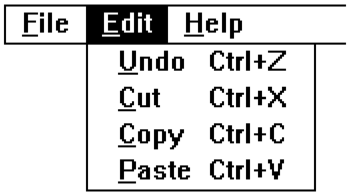
The ampersand (&) underlines the letter that immediately follows it. This lets the user choose the command from the menu by typing that letter.

```
Main_Menu MENU
{
  POPUP "&File"
  {
    MENUITEM "&New", CM_FILENEW
    MENUITEM "&Open...", CM_FILEOPEN
    MENUITEM "&Save", CM_FILESAVE
    MENUITEM "Save &as...", CM_FILESAVEAS
    MENUITEM SEPARATOR
    MENUITEM "&Print...", CM_FILEPRINT
    MENUITEM "Page se&tup...", CM_FILEPAGE_SETUP
    MENUITEM "P&rinter setup...", CM_FILEPRINTER_SETUP
    MENUITEM SEPARATOR
    MENUITEM "E&xit", CM_FILEEXIT
  }
  POPUP "&Edit"
  {
    MENUITEM "&Undo\tCtrl+Z", CM_EDITUNDO
    MENUITEM "&Cut\tCtrl+X", CM_EDITCUT
    MENUITEM "&Copy\tCtrl+C", CM_EDITCOPY
    MENUITEM "&Paste\tCtrl+V", CM_EDITPASTE
  }
  POPUP "&Help"
  {
    MENUITEM "&Index\tF1", CM_HELPINDEX
    MENUITEM "&Keyboard", CM_HELPKEYBOARD
    MENUITEM "&Commands", CM_HELPCOMMANDS
    MENUITEM "&Procedures", CM_HELPPROCEDURES
    MENUITEM "&Using help", CM_HELPUSING_HELP
    MENUITEM SEPARATOR
    MENUITEM "&About...", CM_HELPABOUT
  }
}
```

Close

MENU, MENUITEM and POPUP example (displayed resource)

Click on any part of the menu to view the resource script behind that element:



```
POPUP "&File"
POPUP "&Edit"
{
  MENUITEM "&Undo\tCtrl+Z", CM_EDITUNDO
  MENUITEM "&Cut\tCtrl+X", CM_EDITCUT
  MENUITEM "&Copy\tCtrl+C", CM_EDITCOPY
  MENUITEM "&Paste\tCtrl+V", CM_EDITPASTE
}
```

```
POPUP "&Help"
```

PUSHBUTTON example

See Also

This example defines an "OK" push button. Its control ID is 108. The coordinates of the upper-left corner of the button are 15, 55. The button is 24 dialog units wide and 14   long.

It uses the defauult window style WS_TABSTOP, indicating that when the user presses Tab in a preceding control, the input focus changes to this control. It uses the default button style BS_PUSHBUTTON, indicating that the button is a push button.

```
PUSHBUTTON "OK", 108, 15, 55, 24, 14
```

See Also
PUSHBUTTON

RCDATA examples

See Also

These examples show you the variety of data you can include in a raw data resource:

```
// a single string (not null-terminated)
mystring RCDATA PRELOAD
BEGIN
  "Now is the time."
END
// a bunch of data
mydata RCDATA
BEGIN
  0x1000,
  255,
  "Null-terminated string\0",
  0777,
END
```

See Also
RCDATA

RCINCLUDE example

See Also

This example opens the source file SEARCH.DLG and processes the directives and resource definitions in the file.

```
RCINCLUDE SEARCH.DLG
```

See Also
RCINCLUDE

STRINGTABLE example

See Also

This example lists strings an application might use for displaying messages to the user:

```
STRINGTABLE
BEGIN
  1,  "Press any key to continue..."
  2,  "Click mouse to continue..."
  3,  "All rights reserved"
  4,  "Disk Full"
END
```

To use the #define, set up the .H file with the proper #define statements. This example calls the file STRNAMES.H.

```
#define PressKey    1
#define PressMouse  2
#define AllRights   3
#define DiskFull    4
```

Then in the .RC file:

```
#include strnames.h
/* a single table with defines */
STRINGTABLE
BEGIN
  PressKey,   "Press any key to continue..."
  PressMouse, "Click mouse to continue..."
  AllRights,  "All rights reserved"
  DiskFull,   "Disk Full"
END
```

See Also
STRINGTABLE

User-Defined Resources example

See Also

Example 1

In the first example, the new resource type is DEFAULTS and is named "defdata". The new resource is in the file DFLT.RES.

```
defdata DEFAULTS dflt.res
```

Example 2

In this example, the resource type is 256 and is named "printertype". The resource is in the file PRTYP.RES. The resource can be moved around in memory and then removed from memory when the application no longer needs it.

```
printertype 256 MOVEABLE DISCARDABLE prtyp.res
```

See Also
User-Defined Resources