MENUS AND OTHER RESOURCES

Icons

Icons are small graphical images that represent programs, files, or folders. They are displayed in the title bar of application windows, in the Start menu, in the taskbar, in Windows Explorer, and as shortcuts on the desktop. Icons can be in color or black and white, and they can be in any size.

Cursors

Cursors are graphical images that represent the mouse pointer. They change shape depending on the context, such as when the mouse is hovering over a link, when it is selecting text, or when it is resizing a window. Cursors can be in color or black and white, and they can be in any size.

Character Strings

Character strings are text strings that are used by programs. They can be used for menus, dialog boxes, error messages, and other purposes. Character strings can be stored in the program's .EXE file or in a separate resource file.

Custom Resources

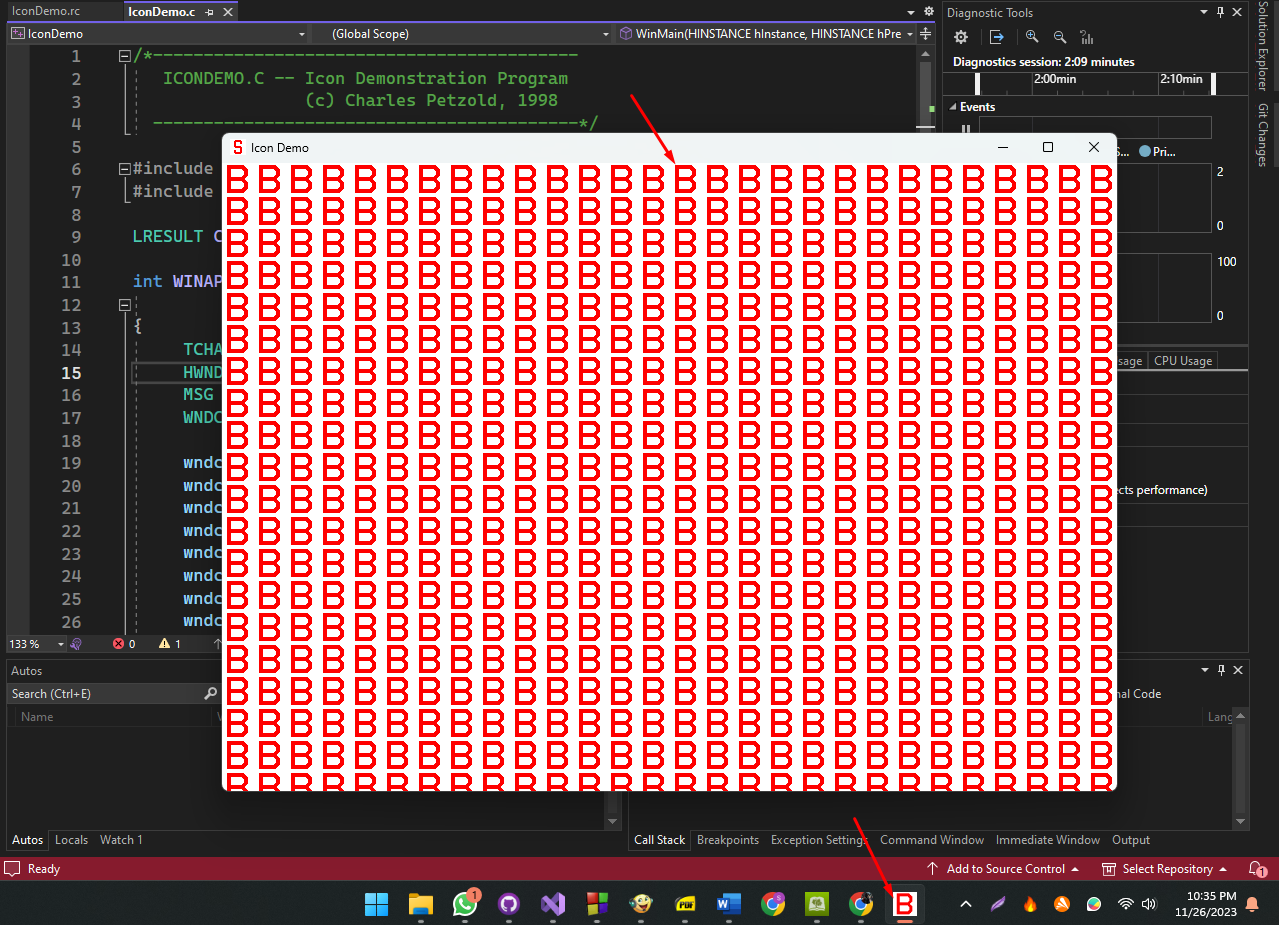
Custom resources are any type of resource that is not an icon, cursor, character string, or menu. They can be used for storing data that is specific to the program, such as images, sounds, or video. Custom resources are stored in the program's .EXE file or in a separate resource file.

Menus

Menus are hierarchical lists of options that users can select to perform actions in a program. They can be displayed as pull-down menus, context menus, or toolbars. Menus can be customized to include the specific options that a program needs.

Keyboard Accelerators

Keyboard accelerators are keyboard shortcuts that allow users to quickly perform actions in a program. They are typically combinations of two or more keys, such as Ctrl+S to save a file. Keyboard accelerators can be customized to the user's preferences.



Introduction

Resources in C programming offer a convenient way to bind various components of a program into the executable file.

This eliminates the need for separate files, making it easier to manage and distribute the application. For instance, icons, cursors, strings, and other custom resources can be included within the program's .EXE file.

Icons as Resources

One notable example is the inclusion of icons. Typically, an icon would require a separate file, but with resources, it can be stored in an editable file on the developer's computer and bound into the .EXE during the build process.

This approach streamlines development and ensures that the icon is an integral part of the executable.

Adding an Icon to a Program

To add an icon to a program, Visual C++ Developer Studio provides the Image Editor, allowing developers to draw an icon that gets saved in an .ICO file.

Simultaneously, Developer Studio generates a resource script (with .RC extension) listing all program resources and a header file (RESOURCE.H) enabling the program to reference these resources.

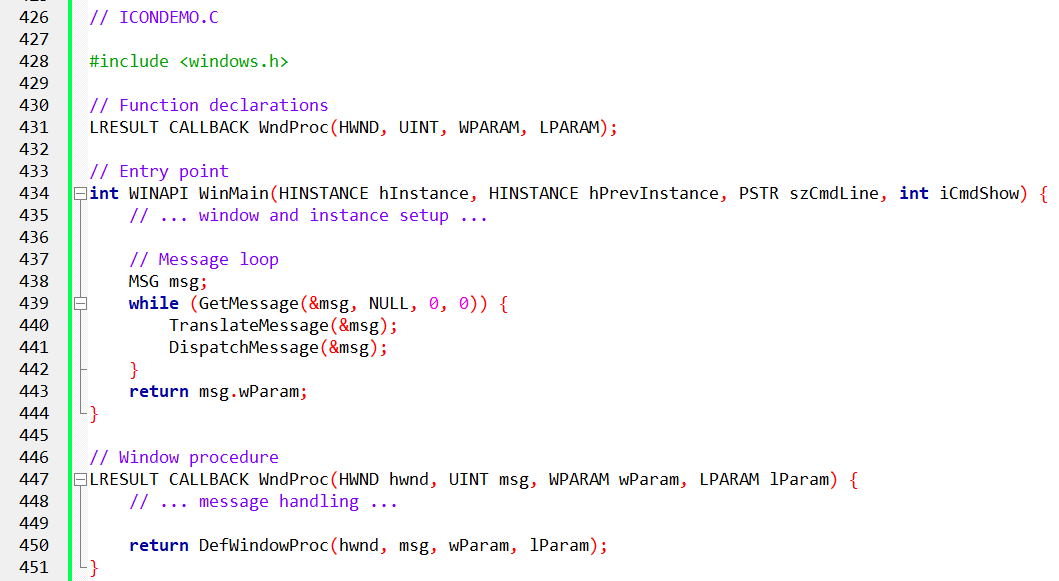
Project Setup: ICONDEMO

Let's illustrate this process by creating a new project named ICONDEMO in Visual C++ Developer Studio.

After creating the project, the studio generates several files, including ICONDEMO.DSW, ICONDEMO.DSP, and ICONDEMO.MAK. Additionally, a C source code file (ICONDEMO.C) is created, where the program logic will be implemented.

Example Program Structure

Here's a simplified version of the program structure:



The source code is in the icondemo folder….

ICONDEMO.C is a Windows program that demonstrates the use of icons in a graphical user interface (GUI) application. It creates a window and fills it with copies of an icon specified in the program's resources.

Windows Header File: The #include <windows.h> statement includes the Windows header file, which contains essential definitions for interacting with the Windows API.

Resource File Inclusion: The #include "resource.h" statement incorporates the resource file, which holds the program's resources, including icons and cursors.

Window Procedure Function: The LRESULT CALLBACK WndProc (HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam) function serves as the window procedure, responsible for handling messages sent to the window by the operating system.

Program Entry Point: The int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance, PSTR szCmdLine, int iCmdShow) function acts as the program's entry point, executed when the program starts.

Variable Declarations: Variables are declared to store essential program information, including the program's name (szAppName), window handle (hwnd), current message (msg), and window class structure (wndclass).

Window Class Configuration: The window class structure (wndclass) is configured with settings that define the window's appearance and behavior.

Window Registration: The RegisterClass (&wndclass) statement registers the window class with the system, allowing the program to create windows based on that class.

Window Creation: The hwnd = CreateWindow (...) statement creates a window using the registered window class, specifying the window's name, position, size, and other attributes.

Window Display: The ShowWindow (hwnd, iCmdShow) function displays the created window, making it visible to the user.

Window Update: The UpdateWindow (hwnd) function refreshes the window's contents, ensuring it is properly rendered on the screen.

Message Loop: The while (GetMessage (&msg, NULL, 0, 0)) (...) loop continuously retrieves messages from the message queue and dispatches them to the window procedure function.

Message Translation: The TranslateMessage (&msg) statement translates the retrieved message into a format compatible with the window procedure function.

Message Dispatching: The DispatchMessage (&msg) statement sends the translated message to the window procedure function for processing.

1. Creating a Resource Script

To create an icon for the ICONDEMO program, you first need to create a resource script. A resource script is a text file that contains the definitions of the program's resources, such as icons, cursors, and menus.

*To create a resource script, follow these steps:*

* In Developer Studio, select File > New.
* In the New dialog box, select Resource Script and click OK.
* In the File Name field, type ICONDEMO.RC and click OK.
* Developer Studio will create two new files: ICONDEMO.RC, the resource script, and RESOURCE.H, a header file that allows the C source code file and the resource script to refer to the same defined identifiers.

2. Adding an Icon Resource

To add an icon resource to the resource script, follow these steps:

* In Developer Studio, open the ICONDEMO.RC file.
* Select Insert > Resource.
* In the Resource dialog box, select Icon and click New.
* A blank 32-pixel-by-32-pixel icon will appear in the resource editor. You can use the painting tools and colors to create your icon.

3. Saving the Icon Resource

Once you have created your icon, you need to save it as an ICO file. To do this, follow these steps:

* In the icon properties dialog box, change the ID to IDI\_ICON.
* Change the Filename to ICONDEMO.ICO.
* Click OK.
* Developer Studio will save the icon as ICONDEMO.ICO in the project directory.

4. Compiling the Program

Now that you have created the icon resource, you can compile the program. To do this, follow these steps:

* In Developer Studio, select Build > Build ICONDEMO.
* Developer Studio will compile the program and link it with the icon resource.

5. Running the Program

Once the program has been compiled, you can run it by following these steps:

* In Developer Studio, select Debug > Start Debugging.
* The program will run and the icon will be displayed in the window.

Here are some additional tips for creating icons:

* Use a distinctive color palette so that your icon will stand out.
* Use simple shapes and colors so that your icon is easy to understand and remember.
* Avoid using too much detail, as this can make your icon appear cluttered and difficult to see.

Creating Resource Files

Resource files are text files that contain the definitions of the program's resources, such as icons, cursors, and menus.

Resource files are compiled into binary resource files using the resource compiler RC.EXE. The binary resource files are then linked with the program's object files and libraries to create the final executable file.

Loading Icons

The LoadIcon function is used to load an icon from a resource file. The function takes two arguments:

* hInstance: The instance handle of the program
* MAKEINTRESOURCE(IDI\_ICON): The resource identifier of the icon
* The MAKEINTRESOURCE macro takes an integer resource identifier and converts it to a resource identifier that can be used with the LoadIcon function.

Drawing Icons

The DrawIcon function is used to draw an icon on the screen. The function takes four arguments:

* hdc: The device context of the window in which to draw the icon
* x: The x-coordinate of the upper-left corner of the icon
* y: The y-coordinate of the upper-left corner of the icon
* hIcon: The handle of the icon to draw

Small Icons

Windows will automatically use a smaller version of an icon when it is more appropriate, such as in the title bar and the taskbar.

The small icon size can be obtained from GetSystemMetrics with the SM\_CXSMSIZE and SM\_CYSMSIZE indices. For most display adapters in current use, the small icon size is 16 by 16 pixels.

To create a small icon, you can select Small (16x16) from the Device combo box in the icon editor. You can then draw a different icon for the small size.

Understanding Resource Script ICON Statements

The line IDI\_ICON ICON DISCARDABLE "icondemo.ico" in the ICONDEMO.RC file is a resource script ICON statement. It defines an icon resource with the following properties:

* Identifier: IDI\_ICON
* Type: ICON
* Filename: icondemo.ico
* Attribute: DISCARDABLE
* Resource Identifiers

The identifier IDI\_ICON is a numeric identifier that uniquely identifies the icon resource within the project. In this case, the identifier is 101. Resource identifiers are used by the LoadIcon function to retrieve specific resources from the compiled resource file.

Resource Types

The type ICON indicates that the resource is an icon. Resource types are used by the resource compiler to organize and manage different types of resources.

Resource Filenames

The filename icondemo.ico specifies the location of the icon file that contains the icon image. The filename can be a relative or absolute path.

Resource Attributes

The attribute DISCARDABLE indicates that the icon can be discarded from memory by Windows if necessary to free up space. This attribute is the default and does not need to be specified.

Obtaining a Handle to an Icon

A program can obtain a handle to an icon by calling the LoadIcon function. The LoadIcon function takes two arguments:

* hInstance: The instance handle of the program
* MAKEINTRESOURCE(IDI\_ICON): The resource identifier of the icon
* The MAKEINTRESOURCE macro converts the integer resource identifier IDI\_ICON to a resource identifier that can be used with the LoadIcon function.

Here is an example of how to obtain a handle to the icon defined in the ICONDEMO.RC file:



Using the Icon Handle

The icon handle can be used to draw the icon on the screen using the DrawIcon function. The DrawIcon function takes four arguments:

* hdc: The device context of the window in which to draw the icon
* x: The x-coordinate of the upper-left corner of the icon
* y: The y-coordinate of the upper-left corner of the icon
* hIcon: The handle of the icon to draw

Here is an example of how to draw the icon defined in the ICONDEMO.RC file at the coordinates (100, 100):



The process of getting a handle to an icon involves defining the icon resource in the resource script, compiling the resource script into a binary resource file, and linking the binary resource file into the program's executable file.

Once the icon is linked into the executable file, the program can obtain a handle to the icon by calling the LoadIcon function. The icon handle can then be used to draw the icon on the screen using the DrawIcon function.

Loading Icons Using LoadIcon

The LoadIcon function is used to load an icon from a resource or from a file. The function takes two arguments:

* hInstance: The instance handle of the program
* resourceIdentifier: The identifier of the icon

The identifier can be a numeric identifier, a character string, or a string prefixed with the # character.

Loading Icons by Numeric Identifier

To load an icon by numeric identifier, you can use the MAKEINTRESOURCE macro.

The MAKEINTRESOURCE macro takes an integer identifier and converts it to a resource identifier that can be used with the LoadIcon function.

Here is an example of how to load an icon by numeric identifier:



Loading Icons by Character String

To load an icon by character string, you can simply pass the string to the LoadIcon function. The string can be the name of the icon or the name of the resource file.

Here is an example of how to load an icon by character string:



Loading Icons by String Prefixed with # Character

To load an icon by string prefixed with the # character, you can pass the string to the LoadIcon function. The string should be a number in ASCII form.

Here is an example of how to load an icon by string prefixed with the # character:



Using LoadIcon in ICONDEMO

ICONDEMO calls the LoadIcon function twice:

* Once when defining the window class.
* Once in the window procedure to obtain a handle to the icon for drawing.

In both cases, ICONDEMO uses the MAKEINTRESOURCE macro to convert the numeric identifier IDI\_ICON to a resource identifier.

Here is an example of how ICONDEMO calls the LoadIcon function in the window procedure:



The LoadIcon function is a versatile function that can be used to load icons by numeric identifier, character string, or string prefixed with the # character. ICONDEMO demonstrates how to use the LoadIcon function to load an icon from a resource file.

Using Icons in Windows Programs: A Deep Dive

1. Setting Icons with WNDCLASS and RegisterClass

When defining a window class using the WNDCLASS structure and registering it with RegisterClass, it's common to specify an icon. This is typically done through the hIcon field of the WNDCLASS structure. Windows intelligently selects the appropriate image size from a single icon file when needed.

2. RegisterClassEx and WNDCLASSEX

There exists an enhanced version, RegisterClassEx, which utilizes the WNDCLASSEX structure. This structure introduces two additional fields: cbSize and hIconSm.

The cbSize field denotes the size of the WNDCLASSEX structure, while hIconSm is intended for the small icon handle. However, using WNDCLASSEX doesn't seem necessary since Windows can extract correctly sized icons from a single file.

3. Dynamic Icon Changes with SetClassLong

To dynamically change the program's icon during runtime, you can use the SetClassLong function. For example, if you have a second icon file associated with the identifier IDI\_ALTICON, you can switch to that icon using:



Alternatively, if you prefer not to retain the program's icon handle but instead display it using the DrawIcon function, you can retrieve the handle through GetClassLong. Here's an example:



While some sections of the Windows documentation suggest that LoadIcon is "obsolete" and favor LoadImage instead, LoadImage, documented in /Platform SDK/User Interface Services/Resources/Resources, undoubtedly offers greater flexibility.

However, it has not yet surpassed LoadIcon's simplicity. In the provided ICONDEMO example, LoadIcon is called twice for the same icon without issues or additional memory consumption.

LoadIcon is one of the few functions that acquire a handle without explicit handle destruction. It's worth noting that while a DestroyIcon function exists, it is primarily used in conjunction with functions like CreateIcon, CreateIconIndirect, and CreateIconFromResource, which enable the dynamic creation of icon images algorithmically within a program.

In conclusion, icons are fundamental elements of Windows programming, and understanding their proper implementation is crucial for crafting user-friendly and aesthetically pleasing applications.

Customizing Mouse Cursors in Windows Programming

Similar to customizing icons, customizing mouse cursors enhances the visual appeal and interactivity of your Windows applications.

While most programmers find the default cursors provided by Windows to be sufficient, customizing cursors can add a unique touch to your program.

Creating customized cursors is straightforward and can be done within the Developer Studio.

Follow the same steps as creating icons: select "Resource" from the "Insert" menu and choose "Cursor." Remember to define the hotspot, which is the point on the cursor where interactions occur.

To set a customized cursor for your window class, use the following statement within your class definition:



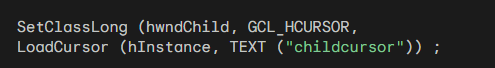
For cursors defined with a text name, use the following statement:



This will display the customized cursor associated with IDC\_CURSOR or szCursor whenever the mouse hovers over a window created based on this class.

For child windows, you can set different cursors depending on the child window below the cursor.

If your program defines the window class for these child windows, assign different cursors to each class by setting the hCursor field accordingly. For predefined child window controls, modify the hCursor field using the following statement:



To change the mouse cursor for specific areas within your client area without using child windows, call the SetCursor function:



Invoke SetCursor during WM\_MOUSEMOVE message processing. Otherwise, Windows will use the cursor specified in the window class when the cursor is moved. Documentation suggests that SetCursor is efficient if the cursor doesn't require significant changes.

Utilizing Character String Resources in Windows Programming

In Windows programming, the integration of character string resources might initially seem unconventional since regular character strings defined in source code are commonly employed.

However, character string resources serve a distinct purpose, primarily facilitating the translation of programs into different languages. This becomes especially relevant when dealing with menus and dialog boxes as part of the resource script, as demonstrated later in this chapter and the next.

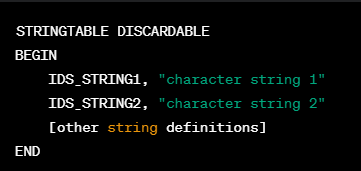
By using character string resources instead of embedding strings directly into the source code, all the text utilized by your program consolidates into one file—the resource script.

This proves advantageous for translation efforts; if the text in the resource script is translated into another language, creating a foreign-language version of your program simply involves relinking, providing a safer alternative to modifying the source code directly.

To create a string table, you can select "Resource" from the Insert menu and then choose "String Table."

The strings appear in a list on the right side of the screen, allowing you to select and define identifiers and corresponding strings for each entry.

In the resource script, the strings are organized within a multiline statement, as illustrated below:



Historically, if you were manually creating this string table in a text editor, you could use left and right curly brackets instead of the BEGIN and END statements.

While a resource script can incorporate multiple string tables, each ID must uniquely identify a single string, and each string can be only one line long with a maximum of 4097 characters.

Control characters like \t and \n for tabs and line breaks are recognized by functions like DrawText and MessageBox.

To utilize string resources in your program, the LoadString function can be employed:



Here, id refers to the ID number preceding each string in the resource script, szBuffer is a pointer to a character array that receives the string, and iMaxLength is the maximum number of characters to transfer. The function returns the number of characters in the string.

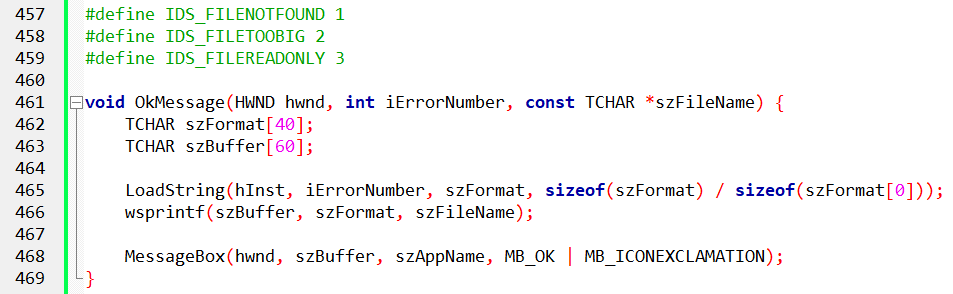
Commonly, string ID numbers are macro identifiers defined in a header file, often prefixed with IDS\_. In scenarios where additional information must be embedded in the string when displayed, C formatting characters can be used, treating the string as a formatting string in wsprintf.

All resource text, including that in the string table, is stored in the .RES compiled resource file and the final .EXE file in Unicode format.

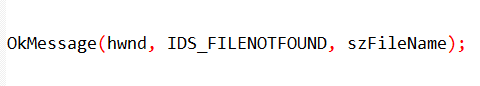
The LoadStringW function loads Unicode text directly, while the LoadStringA function (available under Windows 98) performs a conversion from Unicode to the local code page.

Now, let's explore a function example that employs three character strings to display error messages in a message box. The RESOURCE.H header file contains identifiers for these messages, and the resource script defines a corresponding string table.

The C source code includes this header file and implements a function to display a message box.



To display a message box containing the "file not found" message, the program calls:



This structure exemplifies the seamless integration of character string resources in a Windows program, streamlining the localization process and enhancing code maintainability.

Custom Resources in Windows

Custom resources, also known as user-defined resources, are a powerful feature of the Windows development platform that allows programmers to store and access miscellaneous data within their applications.

Unlike external files, custom resources are embedded directly into the executable file, making them convenient for storing sensitive or frequently accessed data.

Creating Custom Resources

Custom resources are typically created using a resource script file, which is a text file with a .RC extension.

The resource script file defines the resource type, resource name, and the data associated with the resource.

For example, the following resource script defines a custom resource named IDR\_BINTYPE1 of type BINTYPE and associates it with the file BINDATA.BIN:



Loading and Accessing Custom Resources

To load and access a custom resource within your application, you can use the LoadResource and LockResource functions. The LoadResource function takes three parameters:

* hInstance: The handle to the application instance
* lpName: The name or ID of the resource
* lpType: The type of the resource

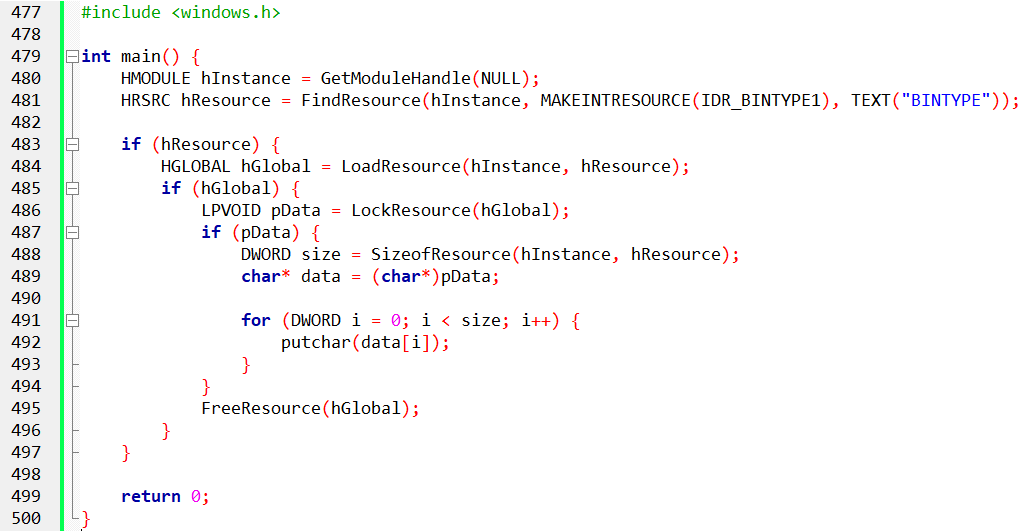
The LoadResource function returns a handle to the resource, which can then be passed to the LockResource function to lock the resource into memory. The LockResource function returns a pointer to the resource data, which can be used to access the data.

Freeing Custom Resources

Once you have finished accessing a custom resource, you should free it from memory using the FreeResource function. This will prevent memory leaks and ensure that the resource data is properly released.

Sample Code

The following code demonstrates how to load and access a custom resource named IDR\_BINTYPE1 and display its contents to the console:

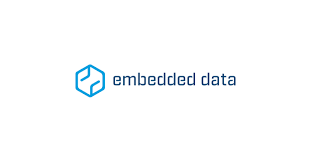


This code will load the custom resource IDR\_BINTYPE1, lock it into memory, and print its contents to the console.

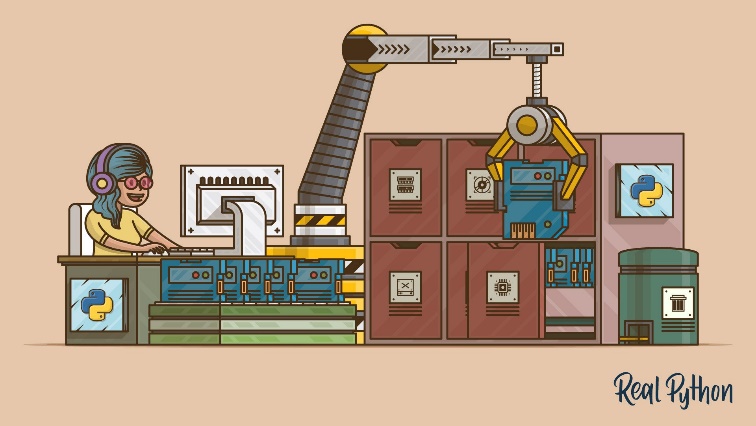
Benefits of Using Custom Resources

Custom resources offer several advantages over external files for storing data in Windows applications:

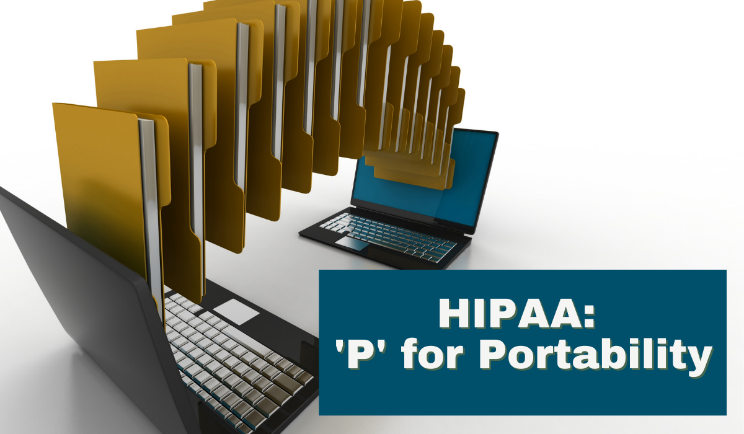
Embedded Data: Custom resources are embedded directly into the executable file, making them convenient for storing sensitive or frequently accessed data.



Memory Management: Windows handles the loading and unloading of custom resources, eliminating the need for explicit file I/O operations.



Portability: Custom resources are embedded in the executable file, making applications portable without relying on external files.



Security: Custom resources are protected by the executable file's permissions, enhancing data security.

