#[1]$[2]K[3]{bmc deb.bmp} Debug Event Browser

1# Contents
2$ Debug Event Browser Contents
3K Contents

#[4]$[5]K[6]K[7]General Overview

Debug Event Browser (**DEB**) is a Win32 application demonstrating the Win32 debug API.   This preliminary version of **DEB** only performs the most rudimentary debugging operations.   **DEB** is not a debugger in the traditional sense but a browser which merely displays the debug events occurring in a debuggee.   The handling of debug events is restricted only to those actions which are necessary to display event information and continue the debuggee.

This Sample is brought to you by:
   Microsoft Developer Support
   Developed by Paul Tissue

4# GenOverview
5$ General Overview
6K Overview - General
7K Overviews

#[8]$[9]K[10]K[11]K[12]Programming Overview

The **Debug Event Browser** (**DEB**) sample demonstrates the following Win32 debug API features:

Debug Event Handler
Modifying a Thread's Context
Reading the Executable's Header

8# ProgOverview
9$ Programming Overview
10K Programming
11K Overview - Programming
12K Overviews

#[13]$[14]K[15]K[16]K[17]K[18]Debug Event Handler
The debug event handler is responsible for the processing of the debug events.

13# Handler
14$ Debug Event Handler
15K Debug Event Handler
16K Debug API
17K WaitForDebugEvent
18K ContinueDebugEvent

```
// *********************************************************************
// FUNCTION : DebugEventThread( DWORD )
// PURPOSE  : Main debug event processing loop
// COMMENTS : The same debugger thread which creates a debuggee process or
//            attaches to a currently running process must also handle all
//            the debug events for that process.
// *********************************************************************
DWORD WINAPI
DebugEventThread( DWORD UserDefinedValue )
{
   DEBUG_EVENT DebugEvent;

   for(;;) {
      if( !WaitForDebugEvent( &DebugEvent, INFINITE ) )
         continue;
      switch( DebugEvent.dwDebugEventCode ) {
         case EXCEPTION_DEBUG_EVENT:
            // ...
            switch( DebugEvent.u.Exception.ExceptionRecord.ExceptionCode
               case EXCEPTION_ACCESS_VIOLATION:
                  // ...handle exception
                  break;
               case EXCEPTION_BREAKPOINT:
                  // ...handle exception
                  break;
               //...
               default:   // An unknown exception occurred
                  // ...handle exception
                  break;
            }
         case CREATE_THREAD_DEBUG_EVENT:
            // ...handle debug event
            break;
         case CREATE_PROCESS_DEBUG_EVENT:
            // ...handle debug event
            break;
         // ...
         default:
            // ...handle debug event
            break;
      }
      //-- default action - just continue
      ContinueDebugEvent( DebugEvent.dwProcessId, DebugEvent.dwThreadId,
         DBG_CONTINUE );
   }

   return( NULL );
}
```

#[19]$[20]K[21]K[22]K[23]K[24]Reading the Executable's Header

Reading the information stored in the executable's headers is important for obtaining such things as symbolic information and details about the object.

[19]# Header
[20]$ Reading the Executable's Header
[21]K Executable Header
[22]K Image Header
[23]K Debug API
[24]K ReadProcessMemory

```
// *********************************************************************
// FUNCTION : GetModuleFileNameFromHeader( HANDLE, HANDLE, DWORD, LPTSTR,
DWORD )
// PURPOSE   : Retrieves the DLL module name for a given file handle of a
//              the module.  Reads the module name from the EXE header.
// COMMENTS :
//    Retrieves only the module name and not the pathname.  Returns the
//    number of characters copies to the buffer, else returns 0.
// *********************************************************************
DWORD
GetModuleFileNameFromHeader( HANDLE hProcess, HANDLE hFile, DWORD BaseOfDll,
   LPTSTR lpszPath, DWORD cchPath )
{
   #define IMAGE_SECOND_HEADER_OFFSET     (15 * sizeof(ULONG)) // relative to
file beginning
   #define IMAGE_BASE_OFFSET                 (13 * sizeof(DWORD)) // relative
to PE header base
   #define IMAGE_EXPORT_TABLE_RVA_OFFSET (30 * sizeof(DWORD)) // relative to
PE header base
   #define IMAGE_NAME_RVA_OFFSET           offsetof(IMAGE_EXPORT_DIRECTORY,
Name)

   WORD    DosSignature;
   DWORD   NtSignature;
   DWORD   dwNumberOfBytesRead = 0;
   DWORD   PeHeader, ImageBase, ExportTableRVA, NameRVA;

   //-- verify that the handle is not NULL
   if( !hFile ) {
     lstrcpy( lpszPath, "Invalid File Handle" );
     return( 0 );
   }

   //-- verify that the handle is for a disk file
   if( GetFileType(hFile) != FILE_TYPE_DISK ) {
     lstrcpy( lpszPath, "Invalid File Type" );
     return( 0 );
   }

   //-- Extract the filename from the EXE header
   SetFilePointer( hFile, 0L, NULL, FILE_BEGIN );
   ReadFile( hFile, &DosSignature, sizeof(DosSignature), &dwNumberOfBytesRead,
     (LPOVERLAPPED) NULL);

   //-- verify DOS signature found
   if( DosSignature != IMAGE_DOS_SIGNATURE ) {
     wsprintf( lpszPath, TEXT( "Bad MZ Signature: 0x%x" ), DosSignature );
     return( 0 );
   }

   SetFilePointer( hFile, IMAGE_SECOND_HEADER_OFFSET, (LPLONG) NULL,
     FILE_BEGIN );
   ReadFile( hFile, &PeHeader, sizeof(PeHeader), &dwNumberOfBytesRead,
     (LPOVERLAPPED) NULL );
   SetFilePointer( hFile, PeHeader, (LPLONG) NULL, FILE_BEGIN );
   ReadFile( hFile, &NtSignature, sizeof(NtSignature), &dwNumberOfBytesRead,
     (LPOVERLAPPED) NULL);
   //-- verify Windows NT (PE) signature found
   if( NtSignature != IMAGE_NT_SIGNATURE ) {
      wsprintf( lpszPath, TEXT( "Bad PE Signature: 0x%x" ), DosSignature );
      return( 0 );
   }

   SetFilePointer( hFile, PeHeader + IMAGE_BASE_OFFSET, (LPLONG) NULL,
      FILE_BEGIN );
   ReadFile( hFile, &ImageBase, sizeof(ImageBase), &dwNumberOfBytesRead,
      (LPOVERLAPPED) NULL);
   SetFilePointer( hFile, PeHeader + IMAGE_EXPORT_TABLE_RVA_OFFSET,
      (LPLONG) NULL, FILE_BEGIN );
   ReadFile( hFile, &ExportTableRVA, sizeof(ExportTableRVA),
      &dwNumberOfBytesRead, (LPOVERLAPPED) NULL);

   //-- now read from the virtual address space in the process
   ReadProcessMemory( hProcess,
        (LPVOID) (BaseOfDll + ExportTableRVA + IMAGE_NAME_RVA_OFFSET),
        &NameRVA, sizeof(NameRVA), &dwNumberOfBytesRead );
   lstrcpy( lpszPath, "Empty!" );
   if( !ReadProcessMemory( hProcess,
           (LPVOID) (BaseOfDll + NameRVA),
           lpszPath, cchPath, &dwNumberOfBytesRead ) )
        lstrcpy( lpszPath, "Access Denied!" );

   return( dwNumberOfBytesRead );
}
```

#[25]$[26]K[27]K[28]K[29]K[30]K[31]Modifying a Thread's Context
The ability to query and modify a thread's context is one of the more powerful features of the Win32 debug API set.

```
// **********************************************************************
// FUNCTION : SkipThreadBreakPoint( HANDLE );
// PURPOSE   : Skip over the break point instruction belonging to
//               hThread.
// COMMENTS :
//    Only the MIPS R4x00 and Alpha AXP require this.
// **********************************************************************
BOOL
SkipBreakPoint( HANDLE hThread )
{
   static CONTEXT Context;

   Context.ContextFlags = CONTEXT_CONTROL;
   if( !GetThreadContext( hThread, &Context ) )
      return( FALSE );
   Context.Fir += 4L;   // Fir is the PC (program counter)
                            // BREAK (breakpoint instruction) occupies 4
bytes
   SetThreadContext( hThread, &Context );

   return( TRUE );
}
```

[25]# Context
[26]$ Modifying a Thread's Context
[27]K Thread Context
[28]K Context
[29]K Debug API
[30]K GetThreadContext
[31]K SetThreadContext

#[32]$[33]K[34]How To Use The Toolbar

The Toolbar allows quick and convenient access to several **Debug Event Browser** options simply with the click of the mouse.

{bmc toolbar.bmp}

32# Toolbar
33$ Toolbar
34K Toolbar

#[35]$[36]K[37]K[38]File Menu Commands

Allows the user to select the debuggee for this application.

Open...
Displays a dialog box that will allow you to open and run an executable file using a common dialog box. This executable file will become the debuggee.

Attach...
Displays a dialog box that will allow you to select a currently running process to attach to. This process will become the debuggee.

Exit
Exits the **Debug Event Browser** application.

35# File
36$ File Menu
37K File Menu
38K Menu Commands

#[39]$[40]K[41]K[42]Edit Menu Commands
Allows the user to copy text to the clipboard.

Cut
Copies the text in the Debug Event window and then delete it from the window.

Copy
Copies the text in the Debug Event window.

Delete
Deletes the text in the Debug Event window.

39# Edit
40$ Edit Menu
41K Edit Menu
42K Menu Commands

#[43]$[44]K[45]K[46]Options Menu Commands
Allows the user to set various options and preferences for this application.

Fonts...
Displays a dialog box that will allow you to set the font for the Debug Event window.

Background Color...
Displays a dialog box that will allow you to set the background color for the Debug Event window.

Preferences...
Displays a dialog box that will allow you to set the options and preferences for this application.

Toolbar
Displays the Tool Bar when checked.

Use Saved Directory
This menu option will set the default directory to the one that was previously saved.   This only occurs when checked.

Save Settings On Exit
This menu option will save all the current session settings upon exiting. This only occurs when checked.

Save Settings Now
This menu option will save all the current session settings now.

43# Options
44$ Options Menu
45K Options Menu
46K Menu Commands

#[47]$[48]K[49]K[50]Help Menu Commands
Displays various types of information regarding to the **Debug Event Browser** application.

Contents
Displays the contents of the Online Help.

Search for Help on...
Displays a list of keywords to search for Online Help topics.

How to use Help
Displays the instructions for using the Online Help facilities.

About Debug Event Browser...
Displays product information about the **Debug Event Browser**.

47# Help
48$ Help Menu
49K Help Menu
50K Menu Commands

#[51]$[52]K[53]Keyboard Commands
The keyboard commands allows quick and convenient access to several **Debug Event Browser** options using simple key combinations.

Control Keys

Ctrl+A
Attaches to a currently running process and begins debugging it.

Ctrl+C
Copies the text in the Debug Event window.

Ctrl+O
Opens an executable and debugs it.

Ctrl+X
Copies the text in the Debug Event window and then delete it from the window.

Alternate Keys

Alt+F4
Exits the **Debug Event Browser**.

Single Keys

F1
Invokes the help file for the **Debug Event Browser**.

Del
Deletes the text in the Debug Event window.

51# Keyboard
52$ Keyboard Commands
53K Keyboard Commands