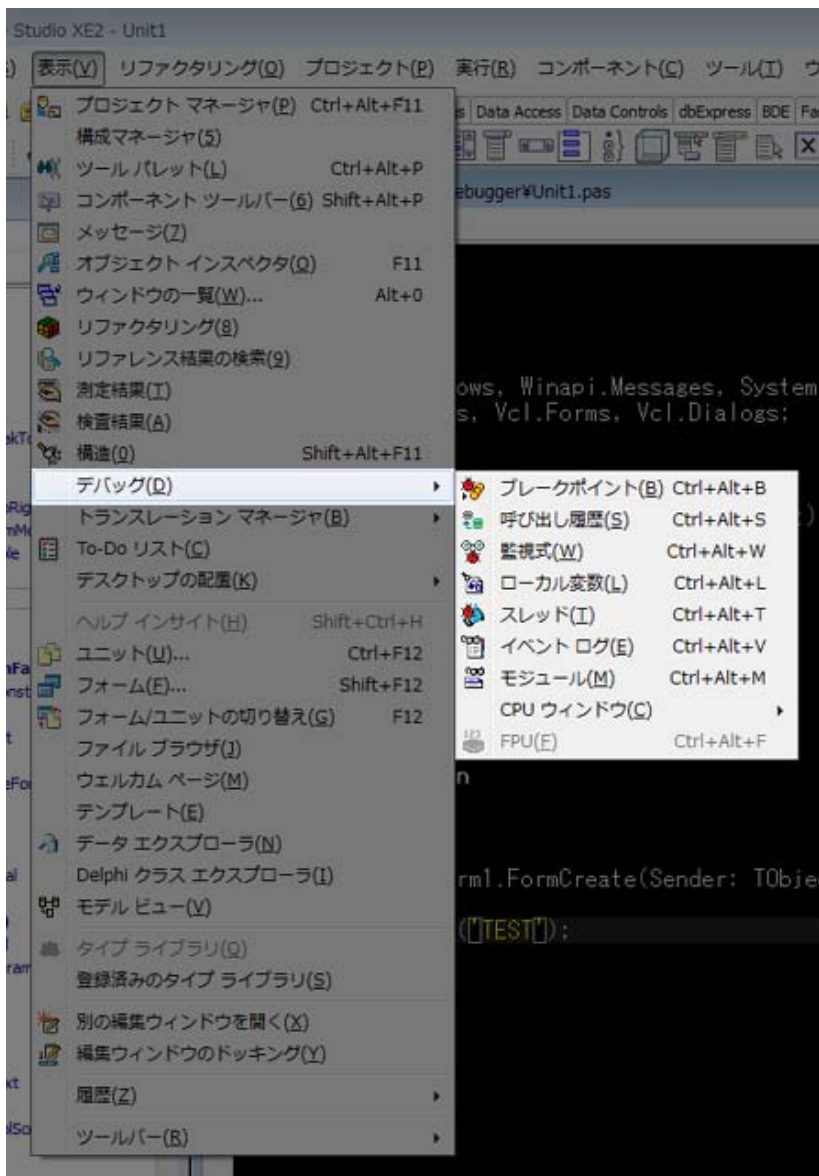




IDE統合デバッガ



Delphi の IDE 統合デバッガ



- IDE 統合デバッガについて「表示」→「デバッグ」で表示される項目ごとに見ていきます



ブレイクポイント一覧



ブレークポイント一覧

```
procedure TForm1.FormCreate(Sender: TObject);  
begin  
27 ShowMessage('TEST');  
30  
end;
```

ファイル名/アドレス	行/長さ	状態	スレッド	アクション	パス カウント	グループ
Unit1.pas	27			ブレーク	0	

ブレークポイント一覧では、一覧以外にも、プロパティや有効無効、データブレークポイント、アドレスで設定、などなど豊富な機能が利用できます。
ここでは特に「プロパティ」と「データブレークポイント」「アドレスで設定」を見ていきます。

ブレークポイント一覧プロパティ(条件)

- ブレーク条件とパスカウント
 - プロパティを開くと「ブレーク条件」と「パスカウント」という項目があります。
 - ブレーク条件
 - if 文の条件式と同じように条件を書けます
 - たとえば「A <> 0」などのように条件を書けます
 - パスカウント
 - そのブレークポイントを何回通った時に停止するかを指定できます。
 - たとえばパスカウントに 3 を指定すると3回目にブレークポイントを通った時に停止します。
- この2つの条件を上手く使うとデバッグが捗ります

ブレークポイント一覧—複数の条件

- ブレークポイントで特筆すべきはもう1つ
 - 同じ行に違うプロパティを持つ複数のブレークポイントを持てます。

The screenshot shows the Delphi IDE with a code editor on the left and a breakpoint list window at the bottom. A red circle highlights the line number '30' in the code editor, which corresponds to the line `OutputDebugString(PChar(IntToStr(i)));`. The breakpoint list window shows two breakpoints for 'Unit1.pas' at line 30, with 'Pass Count' values of '0 of 3' and '0 of 8'. A dialog box titled 'ソースコード ブレークポイントの設定' (Source Code Breakpoint Settings) is open, showing settings for the selected breakpoint. The 'Keep existing breakpoints' checkbox is checked and circled in red. The 'Pass Count' column in the breakpoint list is also circled in red.

```
20  
21  
22  
23  
24  
25  
26  
27  
28  
29  
30  
31  
32  
33  
34
```

```
implementation  
[  
  $R *.dfm  
]  
procedure TForm1.FormCreate(Sender: TObject);  
var  
  i: Integer;  
begin  
  for i := 0 to 9 do  
    OutputDebugString(PChar(IntToStr(i)));  
  end;
```

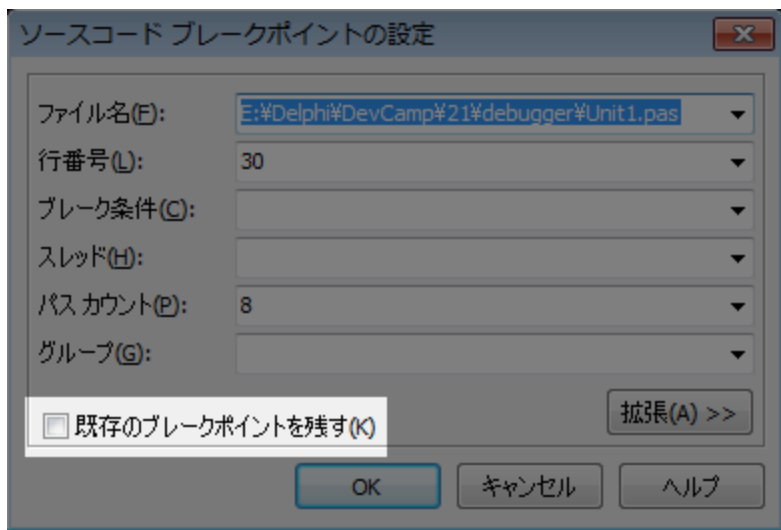
ファイル名/アドレス	行/長さ	状態	スレッド	アクション	パス カウント	グループ
Unit1.pas	30	✓		ブレーク	0 of 3	
Unit1.pas	30	✓		ブレーク	0 of 8	

ソースコード ブレークポイントの設定

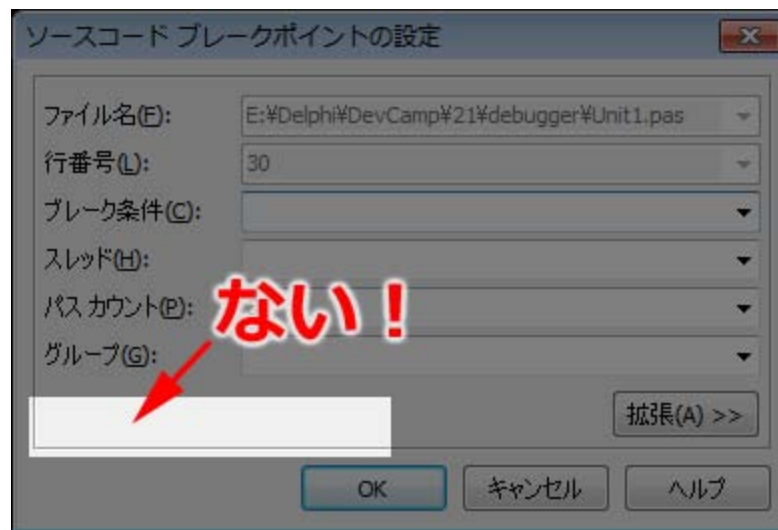
ファイル名(F): E:\Delphi\DevCamp\21\debugger\Unit1.pas
行番号(L): 30
ブレーク条件(C):
スレッド(H):
パス カウント(P): 8
グループ(G):
 既存のブレークポイントを残す(K)
拡張(A) >>
OK キャンセル ヘルプ

ブレークポイント一覧—複数の条件

- 複数の条件を持たせるには「既存のブレークポイントを残す」を利用します。
 - これによって、違う条件を持った複数のブレークポイントを設定できます
 - 注意したいのは、これは「ブレークポイント一覧」ウィンドウからプロパティを表示した時にしか現れないチェックです



ブレークポイント一覧から開いたプロパティ



ブレークポイントのコンテキストメニューから開いたプロパティ

ブレークポイント一覧—ブレークポイントの種類

- ブレークポイント一覧では次の3つのブレークポイントを設定できます

名前	概要	設定可能な状態
ソースブレークポイント	ソースコード上の任意の行に設定できるブレークポイントです	通常時・実行時
アドレスブレークポイント	アドレスに対して設定できるブレークポイントです	実行時
データブレークポイント	変数などのデータが変更される時にブレークするブレークポイントです	実行時

- アドレスブレークポイント

アドレスブレークポイントはユーザーにプログラムを納品した後に威力を発揮します。

例外発生ウィンドウにはアドレスが出ています。

ユーザーから冷害発生時のアドレスを聞いて、そこで停止するようにすれば、例外の原因を追うことができます。

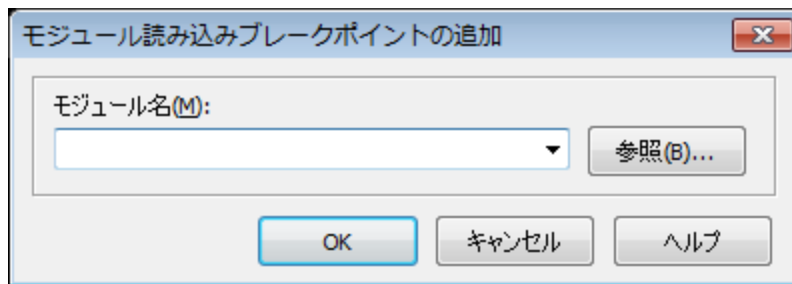
- データブレークポイント

データブレークポイントはデータ(変数)の変更時に停止します。

そのため、どこでデータが変更されたのか？を追うときに役に立ちます。

ブレークポイント一覧—その他のブレークポイント

- モジュールロードブレークポイント
 - ブレークポイント一覧からは設定できず「実行」→「ブレークポイントの追加」→「モジュールロードブレークポイント」から設定できます。
 - DLL など他のモジュールを使っている場合に、モジュールがロードされると停止できます。





ブレークポイントの デモンストレーション





呼び出し履歴と ローカル変数



呼び出し履歴

- 呼び出し履歴はいわゆる「スタックトレース」です。
- ブレークポイントで止まるまでに呼び出されたルートを記録しています

```
procedure TForm1.FormCreate(Sender: TObject);  
var  
    i: Integer;  
begin  
    for i := 0 to 9 do  
        Output(i);  
30 end;  
procedure TForm1.Output(const iIndex: Integer);  
34 begin  
    OutputDebugString(PChar(IntToStr(iIndex)));  
end;  
end.
```

呼び出し履歴 - スレッド 2508

- Unit1.TForm1.Output(0)
- Unit1.TForm1.FormCreate(\$2498650)
- :004fdfbf Vd + \$4FDFBF
- :004fdc03 Vd + \$4FDC03
- :004fdbd2 Vd + \$4FDBD2
- :005084ed Vd + \$5084ED
- Project1.Project1
- :75f8339a kernel32.BaseThreadInitThunk + 0x12
- :76ef9ef2 ntdll.RtlInitializeExceptionChain + 0x63
- :76ef9ec5 ntdll.RtlInitializeExceptionChain + 0x36

呼び出し履歴—ローカル変数

- ローカル変数ウィンドウは呼び出し履歴と密接に関連しています。
- 呼び出し履歴で呼び出された単位ごとにローカル変数を表示できます

呼び出し履歴一ローカル変数

```

1 procedure TForm1.FormCreate(Sender: TObject);
2 var
3   i: Integer;
4 begin
5   for i := 0 to 9 do
6     Output(i);
7 end;
8
9 procedure TForm1.Output(i: Integer);
10 var
11   SL: TStrings;
12 begin
13   OutputDebugString(PChar(
14     SL := nil;
15     try
16       SL.Add('test');
17     except on E: Exception do
18       OutputDebugString(PChar(
19     end;
20 end;

```

ローカル変数 - スレッド 2744

名前	値
Unit1.TForm1.Output(0)	
Self	(([csInheritable], False, (0, 0), (500, 351), (...
iIndex	0
SL	nil
E	0

```

1 procedure TForm1.FormCreate(Sender: TObject);
2 var
3   i: Integer;
4 begin
5   for i := 0 to 9 do
6     Output(i);
7 end;
8
9 procedure TForm1.Output(i: Integer);
10 var
11   SL: TStrings;
12 begin
13   OutputDebugString(PChar(
14     SL := nil;
15     try
16       SL.Add('test');
17     except on E: Exception do
18       OutputDebugString(PChar(
19     end;
20 end;

```

ローカル変数 - スレッド 2744

- Unit1.TForm1.Output(0)
- Unit1.TForm1.FormCreate(\$248B650)
- :004fdbf Vd + \$4DFBF
- :004fdbc3 Vd + \$4FDC03
- :004fdbd2 Vd + \$4FDBD2
- :005084ed Vd + \$5084ED
- Project1.Project1
- :75f8339a kernel32.BaseThreadInitThunk + 0x12

```

1 procedure TForm1.FormCreate(Sender: TObject);
2 var
3   i: Integer;
4 begin
5   for i := 0 to 9 do
6     Output(i);
7 end;
8
9 procedure TForm1.Output(i: Integer);
10 var
11   SL: TStrings;
12 begin
13   OutputDebugString(PChar(
14     SL := nil;
15     try
16       SL.Add('test');
17     except on E: Exception do
18       OutputDebugString(PChar(
19     end;
20 end;

```

ローカル変数 - スレッド 2744

名前	値
Unit1.TForm1.FormCreate(\$248B650)	
Self	(([csInheritable], False, (0, 0), (500, 351), (...
Sender	0
i	0



呼び出し履歴の デモンストレーション





スレッドの状態と モジュール



スレッドの状態

- 現在動いているスレッドの状態を示しています。
 - カレントスレッドを変更したり、実行させないように凍結したりといった操作ができます。
 - また、ブレークポイントで特定のスレッドでのみブレークさせたい場合は、このウィンドウで表示されているスレッドIDを指定できます。
 - TThread のインスタンスなど同じコードが複数のスレッドから呼ばれるときに便利です。

スレッド ID	状況	状態	位置	待機チェーン
Project1.exe (752)				
2744	停止	ブレークポイント	E:\Delphi\DevCamp\21\debugger\Unit1.pas(40)	
2132	停止	不明	\$76EE013D	
2836	停止	不明	\$76EE1F26	

- ソースの表示(V) Ctrl+V
- ソースの編集(G) Ctrl+S
- カレントスレッドに設定(M) Enter
- 凍結(E)
- 他のスレッドをすべて凍結(O)
- 凍結解除(H)
- すべてのスレッドを凍結解除(A)
- スレッド名の設定(N)
- プロセスオプション(P)
- 常に手前に表示(S)
- ドッキング可能(D)

```
ut(const iIndex: Integer);
```

```
char(IntToStr(iIndex));
```

モジュール

- 現在のプロセスにロードされているモジュールの一覧を取得できます。
- ここで、モジュールロードブレークポイントを置くこともできます。
 - またエントリポイントの逆アセンブル表示も可能です。

名前	ベースアドレス	パス	シンボルファイル	順序	エントリポイント	アドレス
プロセス 752						
● Project1.exe	\$00400000	E:\Delphi\DevCamp\21#debugger\Win32\Debug\Project1.exe	<Project1.rsm>	1	InterlockedAdd	\$00402764
● ntdll.dll	\$76EC0000	ntdll.dll		2	InterlockedCompareExchange	\$00402770
● KERNEL32.dll	\$75F70000	C:\Windows\system64\kernel32.dll		3	InterlockedCompareExchangePointer	\$00402778
● KERNELBASE.dll	\$76490000	C:\Windows\system64\kernelbase.dll		4	InterlockedDecrement	\$00402780
● OLEAUT32.dll	\$75030000	C:\Windows\system64\oleaut32.dll		5	InterlockedExchange	\$0040278C
● ole32.dll	\$74ED0000	C:\Windows\system64\ole32.dll		6	InterlockedIncrement	\$00402798
● msvcrt.dll	\$74920000	C:\Windows\system64\msvcrt.dll		7	CloseHandle	\$004027A4
● GDI32.dll	\$75220000	C:\Windows\system64\GDI32.dll		8	GetStdHandle	\$004027AC
● USER32.dll	\$74600000	C:\Windows\system64\USER32.dll		9	WriteFile	\$004027B4
● ADVAPI32.dll	\$74C30000	C:\Windows\system64\ADVAPI32.dll		10	FindClose	\$004027BC
● SECHOST.dll	\$76100000	C:\Windows\SysWOW64\sechost.dll		11	FindFirstFile	\$004027C4
● RPCRT4.dll	\$75120000	C:\Windows\system64\RPCRT4.dll		12	InitializeCriticalSection	\$004027CC
● SspiCli.dll	\$745A0000	C:\Windows\system64\SspiCli.dll		13	EnterCriticalSection	\$004027D4
● CRYPTBASE.dll	\$74590000	C:\Windows\system64\CRYPTBASE.dll		14	LeaveCriticalSection	\$004027DC
● LPK.dll	\$75210000	C:\Windows\system64\LPK.dll		15	DeleteCriticalSection	\$004027E4
● USP10.dll	\$74D10000	C:\Windows\system64\USP10.dll		16	CreateThread	\$004027EC
● MSIMG32.dll	\$73F10000	C:\Windows\SysWOW64\msimg32.dll		17	GetCurrentThreadId	\$004027F4
● VERSION.dll	\$74570000	C:\Windows\SysWOW64\version.dll		18	Yield	\$004027FC
● COMCTL32.dll	\$743D0000	C:\Windows\WinSxS\x86_microsoft.windows.common-controls_6595b6414eccf146_x-ww-6595b6414e		19	ExitThread	\$00402804
● SHLWAPI.dll	\$74BD0000	C:\Windows\system64\SHLWAPI.dll		20	ExitProcess	\$0040280C
● SHELL32.dll	\$75320000	C:\Windows\system64\shell32.dll		21	RaiseException	\$00402814
● WINSPOOL.DRV	\$74370000	C:\Windows\SysWOW64\winspool.drv		22	RtlUnwind	\$0040281C
● IMM32.dll	\$75280000	C:\Windows\SysWOW64\IMM32.dll		23	UnhandledExceptionFilter	\$00402824
● MSCTF.dll	\$74730000	C:\Windows\system64\MSCTF.dll		24	GetLastError	\$0040282C
● UxTheme.dll	\$74260000	C:\Windows\SysWOW64\uxtheme.dll		25	FreeLibrary	\$00402834
● apphelp.dll	\$73F50000	C:\Windows\SysWOW64\apphelp.dll		26	LoadString	\$0040283C
● Atok24W.ime	\$70DD0000	C:\Windows\SysWOW64\ATOK24W.IME		27	GetCommandLine	\$00402844
● Atok24Ae.dll	\$742E0000	C:\Windows\SysWOW64\Atok24Ae.dll		28	GetModuleFileName	\$0040284C
● Atok24De.dll	\$73FA0000	C:\Windows\SysWOW64\Atok24De.dll		29	GetModuleHandle	\$00402854
● dwmapi.dll	\$74150000	C:\Windows\SysWOW64\dwmapi.dll		30	GetProcAddress	\$0040285C
● WTSAPI32.dll	\$73F00000	C:\Windows\SysWOW64\Wtsapi32.dll		31	GetStartupInfo	\$00402864
● WINSTA.dll	\$73660000	C:\Windows\SysWOW64\WINSTA.dll		32	LoadLibraryEx	\$0040286C
					GetACP	\$00402874
					MultiByteToWideChar	\$0040287C
					WideCharToMultiByte	\$00402884
					GetLocaleInfo	\$0040288C
					GetUserDefaultUILanguage	\$00402894
					GetSystemDefaultUILanguage	\$0040289C
					IsValidLocale	\$004028A4
					CharNext	\$004028AC
					CompareString	\$004028B4
					MessageBoxA	\$004028BC
					RegCloseKey	\$004028C4
					RegOpenKeyEx	\$004028CC
					RegQueryValueEx	\$004028D4
					GetVersion	\$004028DC

Project1

- System
- System.AnsiStrings
- System.Character
- System.Classes
- System.Contrns
- System.DateUtils
- System.Diagnostics
- System.Generics.Collections
- System.Generics.Defaults
- System.HelpIntfs



スレッドの状態と モジュールの デモンストレーション



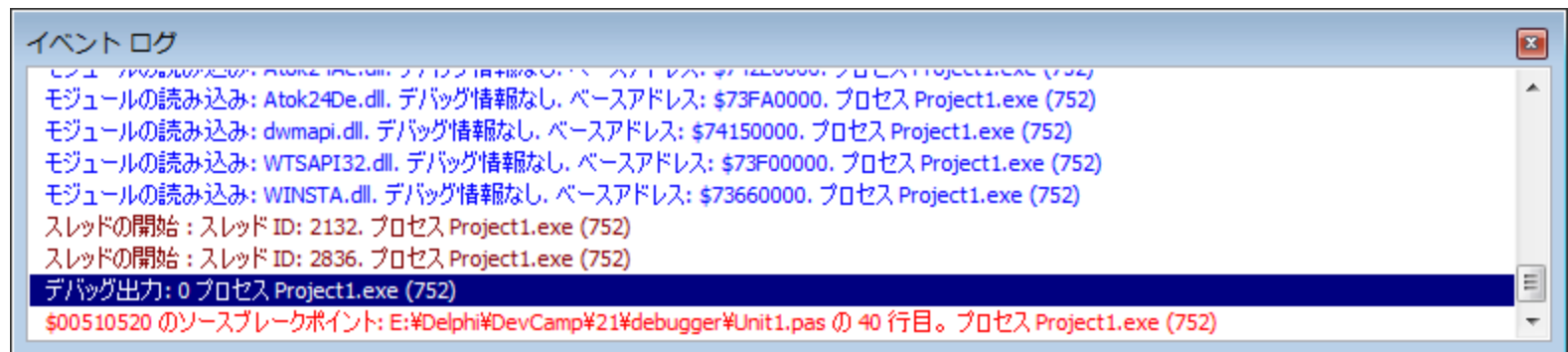


イベントログ



イベントログ

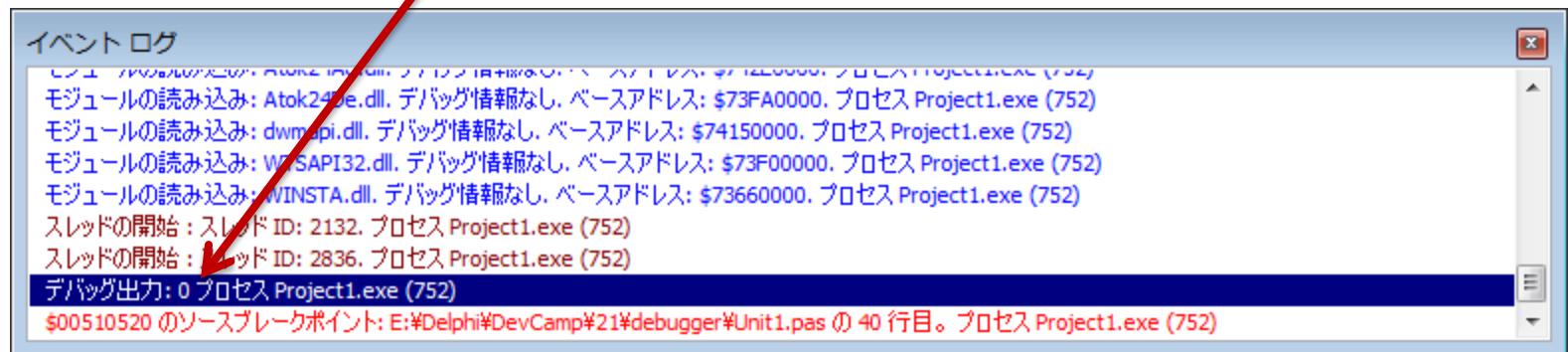
- イベントログは、気軽に使えるデバッグツールです
 - OutputDebugString を使っていわゆる「printfデバッグ」ができます
 - ブレークポイントの停止時にスタックトレースを出力することができます
 - Windowsメッセージを出力できます



イベントログーOutputDebugString

- OutputDebugString は Win32 API です。
 - 呼び出したプロセスにアタッチされているデバッガがあるときに、文字列をデバッガに渡します。
 - Delphi では当然、統合デバッガに文字列が渡ります。
 - 統合デバッガは、受け取った文字列をイベントログに出力します。

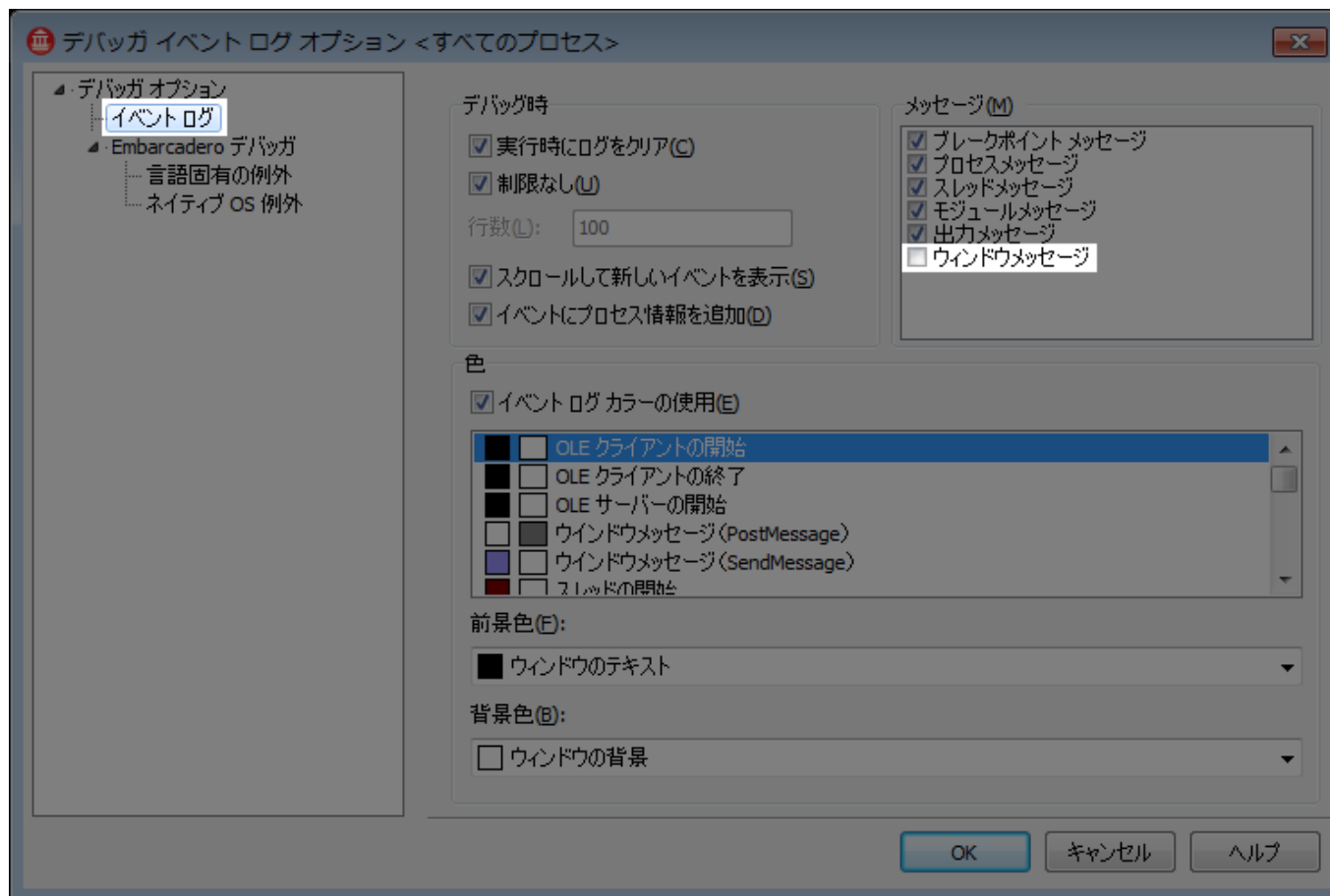
OutputDebugString を使って出力したログ



```
イベント ログ
プロジェクト1.exe (752)
モジュールの読み込み: Atok24Se.dll, デバッグ情報なし, ベースアドレス: $73FA0000, プロセス Project1.exe (752)
モジュールの読み込み: dwmapi.dll, デバッグ情報なし, ベースアドレス: $74150000, プロセス Project1.exe (752)
モジュールの読み込み: WSAPI32.dll, デバッグ情報なし, ベースアドレス: $73F00000, プロセス Project1.exe (752)
モジュールの読み込み: WINSTA.dll, デバッグ情報なし, ベースアドレス: $73660000, プロセス Project1.exe (752)
スレッドの開始: スレッド ID: 2132, プロセス Project1.exe (752)
スレッドの開始: スレッド ID: 2836, プロセス Project1.exe (752)
デバッグ出力: 0 プロセス Project1.exe (752)
$00510520 のソースブレークポイント: E:\Delphi\DevCamp\21\debugger\Unit1.pas の 40 行目。プロセス Project1.exe (752)
```

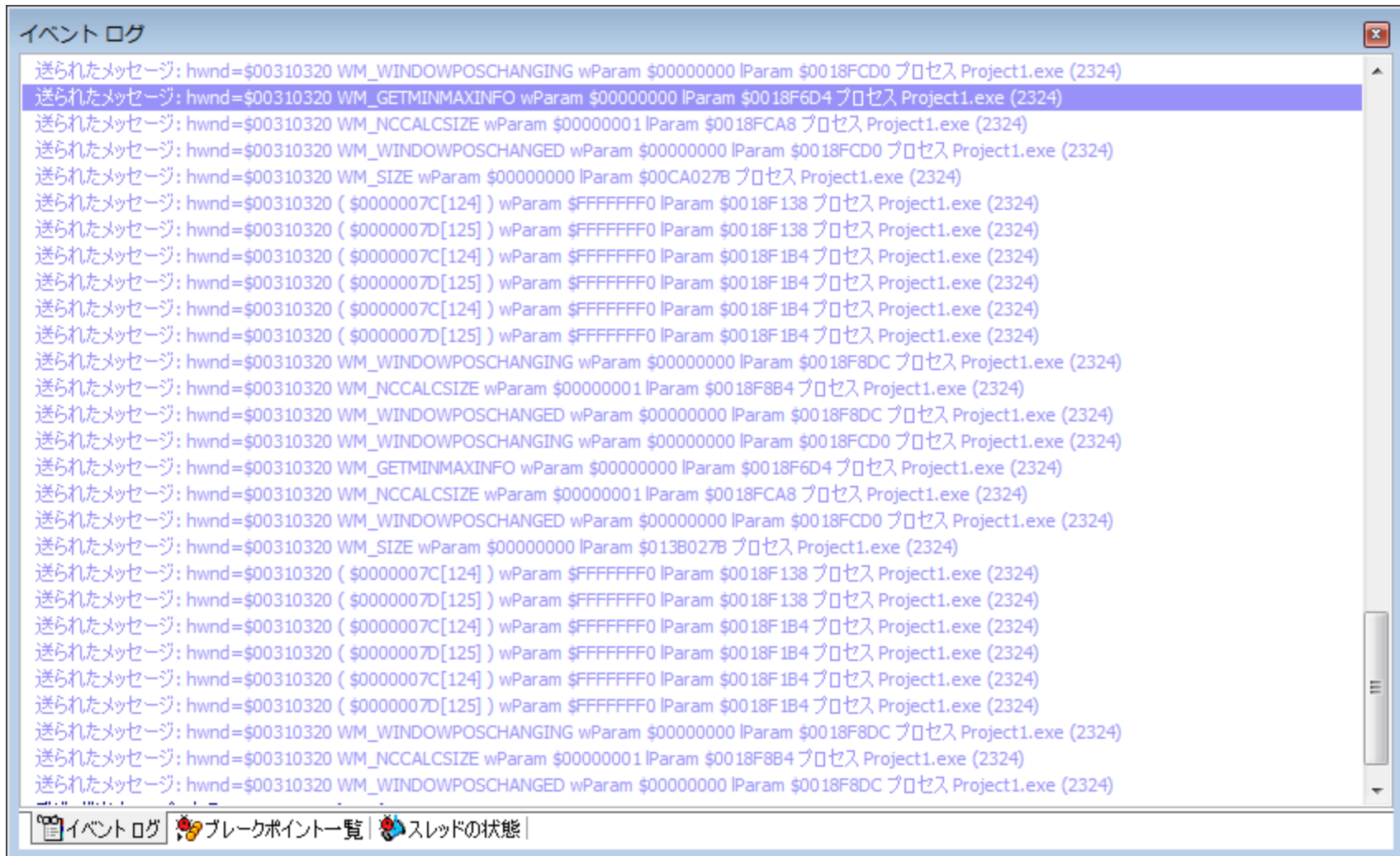

イベントログメッセージ

- イベントログには Application が受け取ったメッセージを出力する機能があります。



イベントログメッセージ

- 実際にメッセージを表示させると、こんな感じになります





イベントログの デモンストレーション





言語で用意されるデバッグ機能



デバッグの容易さ

- モダンな言語ではデバッグ作業も重視されています。
- たとえば Java では、VM 上での動作となるため、ほぼ全ての例外をキャッチできます。キャッチした例外オブジェクトから詳細を知ること、デバッグに生かすことができます
 - ネイティブ言語から考えると Null ポインタへのアクセスを例外としてキャッチでき、かつ、その後、続けて実行できるなんて驚異です。
- もちろん、Delphi 言語も例外なく、デバッグのための機能が用意されています。

Assert の使用

- Assert は、System ネームスペースに用意されているグローバルな手続きです。
 - 実態はコンパイラマジックによって System._Assert へ変換される特殊な手続きです。
- Assert を使うと、任意の条件で例外 (EAssertionFailed) を発生させます
 - Assert の第一引数が false の時 EAssersionFailed 発生
 - 例えば Assert(Sender is TButton); とすると Sender が TButton では「無かったとき」例外を発生させます。
- \$ASSERTIONS ON/OFF (もしくは \$C +/-) 指令を使ってコンパイラに直接 Assert のコード生成を抑制させることができます。

Assert の使用

- Assert の内部を見ると AssertionFailed 発生時に AssertErrorProc という手続きを呼んでいます。
- AssertErrorProc はグローバルな変数です。
- AssertErrorProc の中身を自分のハンドラに変えると、詳細な情報を得られます。

Assert の使用—AssertErrorProc

```
procedure AssertErrorHandler(  
    const iMsg, iFilename: String;  
    const iLineNo: Integer;  
    const iAddress: Pointer);  
var  
    Err: String;  
begin  
    Err :=  
        Format(  
            '%s (%s line %d @ %x)',  
            [iMsg, iFilename, iLineNo, Integer(iAddress)]);  
  
    ShowMessage(Err);  
end;  
  
procedure TForm1.FormCreate(Sender: TObject);  
begin  
    AssertErrorProc = AssertErrorHandler;  
    Assert(false);  
end;
```

Assert の使用トリック

- AssertErrorProc を使うとエラーが発生したユニット、行番号、アドレスなどが判ります。
- これを利用して任意の例外で、ユニットや大まかな行番号を取得できます。

Assert の使用—AssertErrorProc

```
procedure AssertErrorHandler(  
  const iMsg, iFilename: String;  
  const iLineNo: Integer;  
  const iAddress: Pointer);  
begin  
  省略  
end;  
  
procedure TForm1.FormCreate(Sender: TObject);  
var  
  SL: TStringList;  
begin  
  AssertErrorProc = AssertErrorHandler;  
  SL := nil;  
  try  
    SL.Add('TEST');  
  except  
    Assert(false); // except 内で Assert を呼び AssertErrorProc を実行させる  
  end;  
end;
```



Assert の デモンストレーション



TApplication.OnException

- TApplication.OnException というイベントハンドラがあります。
- これは例外が発生したときに呼ばれ、例外処理を自前の処理に置き換えることができます。
 - デフォルトでは例外ダイアログが表示されますが、それをログファイルに出力するように書き換える事もできます。

System.NoErrMsg

- NoErrMsg について、ついでに紹介します。
- このグローバル変数を True に設定すると、エラーダイアログが出ません。
- 何らかの理由でエラーダイアログを表示したくない場合に TApplication.OnException とともに設定しておくとも良いかもしれません。

その他にも

- TVirtualMethodInterceptor を使うとメソッド呼び出しの後に処理を実行できます。
- Exception.StackTrace プロパティには簡易な呼び出し履歴が入っています
 - Exception.GetExceptionStackInfoProc や Exception.GetStackInfoStringProc を置き換えたり、Exception.RaiseOuterException を利用するとサードパーティー製(自前)の例外処理を追加できます。
- class helper を使えば任意のクラスのデータを公開することもできます。



付録：外部ツール



EventViewer

- サービスプログラムやサーバープログラムなど GUI を持たないプログラムではログが重要です
- 自前のログシステムでログを吐くことも、もちろんできますが Windows の SystemLog を使うこともできます。
- Windows の SystemLog はイベントビューアで見ることができます。

EventViewer—書き込む(ソース抜粋)

```
procedure TEventLog.Add (  
  const iLogType: TLogType;  
  const iCategory, iEventId: Integer;  
  const iMsg: String;  
  const iDataLength: Integer;  
  const iData: Pointer);  
var  
  Ptr: Pointer;  
begin  
  Ptr := PChar(iMsg);  
  
  FHandle := RegisterEventSource(nil, PChar(FName));  
  try  
    ReportEvent(  
      FHandle,  
      Ord(iLogType), // Event Type  
      0, // Category  
      0, // Event Id  
      nil, // User Sid  
      1, // 書き込む文字列の数  
      iDataLength, // 書き込むバイナリデータのバイト数  
      @Ptr, // 書き込む文字列(ポインタのポインタ)  
      iData); // バイナリデータ  
  finally  
    DeregisterEventSource(FHandle);  
  end;  
end;  
end;
```


EventViewer—書き込んだ情報の閲覧

The screenshot displays the Windows Event Viewer application. The main window shows a tree view on the left with 'アプリケーションとサービス ログ' expanded to 'アプリケーション'. The central pane shows a table of events:

レベル	日付と時刻	ソース	イベント ID	タスクのカテゴリ
情報	2012/02/28 15:50:03	TEST_EVENT	0	なし
情報	2012/02/28 15:36:05	TEST_EVENT	0	なし

Two 'イベントプロパティ' windows are overlaid. The left window shows the '全般' tab with a message: 'ソース "TEST_EVENT" からのイベント ID 0 の説明が見つかりません。このイベントを発生させるコンポーネントがローカル コンピューターにインストールされていないか、インストールが壊れています。ローカル コンピューターにコンポーネントをインストールするか、コンポーネントを修復してください。' Below this is a table of event details:

ログの名前(N):	アプリケーション		
ソース(S):	TEST_EVENT	ログの日付(D):	2012/02/28 15:50:03
イベント ID(E):	0	タスクのカテゴリ(C):	なし
レベル(L):	情報	キーワード(K):	クラシック
ユーザー(U):	N/A	コンピューター(B):	sikinami01
オペコード(O):			
詳細情報(D):	イベント ログのヘルプ		

The right window shows the '全般' tab with '表示(N)' selected and 'XML で表示(X)' unselected. The event data is displayed as follows:

```
+ System
- EventData
    test message 1
```



EventViewer の デモンストレーション





EurekaLog

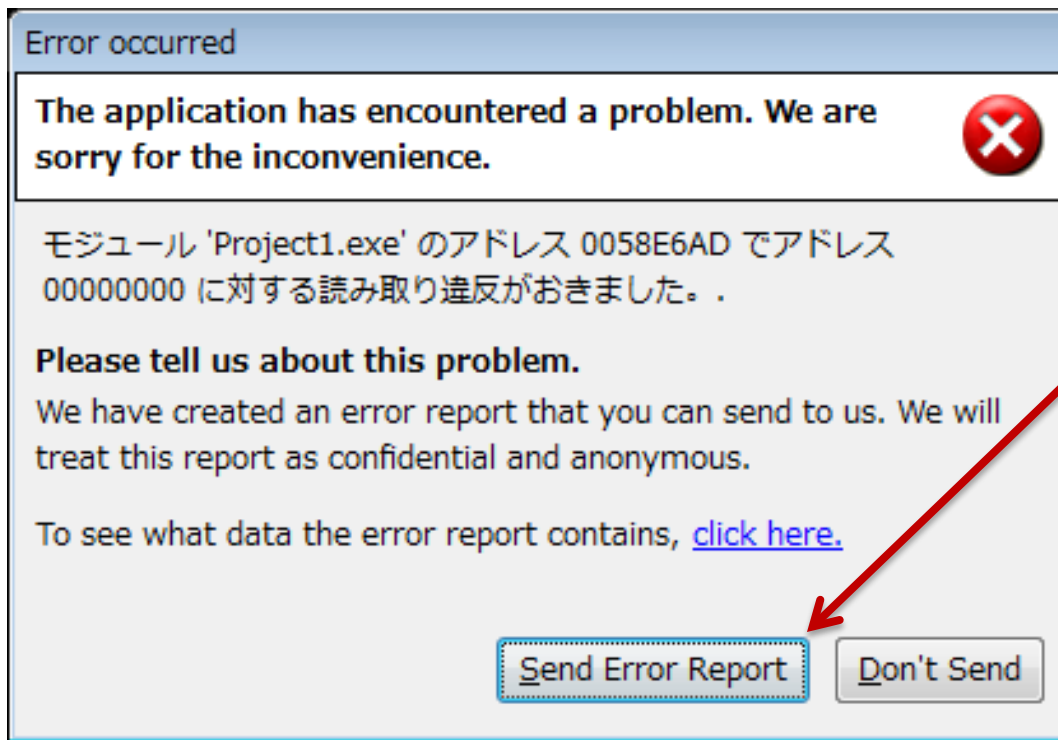


EurekaLog

- EurekaLog はパワフルなデバッグ用ツールです
- 全ての例外、無限ループ、メモリの未解放などを検出できます
- しかも！ 検出した例外はスタックトレース、スクリーンショット、逆アセンブル結果などの情報付きで実行先から送信可能です。
- 入手先： <http://eurekalog.com/>
 - シングルデベロッパーライセンスで 99ドル

EurekaLog

- エラーが発生するとエラーダイアログが表示されますがEurekaLog を組み込んだ実行ファイルでは EurekaLog のエラーダイアログが表示されます



ちなみにちゃんと設定すれば、このダイアログに出る文言はローカライズできます

email クライアントで添付として送ったり、EurekaLog が搭載する SMTP サーバを介して送ったり、BugZilla 形式のウェブにポストしたりと色々な形式でバグレポートを送信できます

EurekaLog

- EurekaLog が送ってきたレポートは、専用のビューアでこんな感じに表示されます

EurekaLog Viewer 3.0.9

Import Delete Clear XML Print Help Exit Password Refresh

Drag a column header here to group by that column

ID	Status	Note	Date (local)	Type	Message	Module
067F	Fixed		2011/12/08	EAccessViolation	モジュール 'EvoTreeEditor.exe' のアドレス 0055C439 でアドレス 000...	EvoTreeEditor.exe - (Evolution Tree) v. 1.2.7.219

General Call Stack Modules Processes Assembler CPU

Address	Module	Unit	Class	Procedure/Method	Line	Rel. line
Exception Thread: ID=6252; Priority=0; [Main]						
0055C439	EvoTreeEditor.exe	uTinyLayer.pas	TLayerCanvas	FillMem	2958	6
0055C430	EvoTreeEditor.exe	uTinyLayer.pas	TLayerCanvas	FillMem	2952	0
0055C68D	EvoTreeEditor.exe	uTinyLayer.pas	TLayerCanvas	FillColor	3115	1
0055C680	EvoTreeEditor.exe	uTinyLayer.pas	TLayerCanvas	FillColor	3114	0
005A06B5	EvoTreeEditor.exe	uTree.pas	TTreeBase	ClearBack	5466	10
0055AE39	EvoTreeEditor.exe	uTinyLayer.pas	TLayers	CallChange	1271	4
0055AF2F	EvoTreeEditor.exe	uTinyLayer.pas	TLayers	Add	1374	5
005A5088	EvoTreeEditor.exe	uTree.pas	TTreeBase	PaintView	7774	28
005A4F28	EvoTreeEditor.exe	uTree.pas	TTreeBase	PaintView	7746	0
005A771D	EvoTreeEditor.exe	uTree.pas	TTreeBase	WMPaint	8987	4
75DAC477	USER32.dll			InvalidateRect		
778BE5B0	ntdll.dll			RtlLeaveCriticalSection		
75DAC09E	USER32.dll			EndPaint		
005A7C20	EvoTreeEditor.exe	uTree.pas	TTreeBase	WndProc	9176	113
7728140B	kernel32.dll			GetCurrentThreadId		
75DA9D0F	USER32.dll			PostMessageW		
75DA9EC2	USER32.dll			GetPropW		
75DA9E97	USER32.dll			GetPropW		
75DA8B0B	USER32.dll			DispatchMessageW		
75DA8B01	USER32.dll			DispatchMessageW		
005CC259	EvoTreeEditor.exe	EvoTreeEditor.dpr			43	4
772FF13A	kernel32.dll			BaseThreadInitThunk		



Q & A

