# CnWizards专家包调试说明书

作者：刘啸 (liuxiao@cnpack.org)

部门：CnPack开发组专家组管理员

类别：调试说明

版本：V0.3.0.0

创建：2003.12.12

修改：2007.11.11

## 引 言

### 编写目的

本文档描述了CnWizards IDE专家包的基本调试方法以及使用 CnDebugViewer进行日志记录型调试的基本技巧（原先专家包使用的是uDbg/Overseer，目前已经改用自主编写的CnDebug/CnDebugViewer）。为了让更多的开发组成员能参与开发，在此特将专家包的上述调试方法写一份说明。这些调试方法也有通用性，下面会详细介绍。

预期读者为CnPack开发组参与CnWizards开发的成员及对专家开发感兴趣的成员，要求具有一定的Delphi 开发能力。

### 背景

CnPack开发当前处于积极进步的阶段，越来越多的成员正在参与开发中。CnWizards已经基本成型，功能也基本完善，正处于积极推广阶段。

### 定义

CnWizards，CnPack开发包中IDE专家工具包部分，简称专家包。

## 基本调试方法

专家包的存在形式是DLL，因此，其基本调试方法和DLL的调试一样，需要运行宿主程序来加载此DLL。而专家包的宿主程序自然是DELPHI或BCB的IDE，所以，调试专家包就是一句话：用IDE的调试器来运行调试IDE。下面以DELPHI 5下的专家包调试为例，说明专家包调试的基本方法：

首先，需要在Run菜单的Parameters子菜单中设置DLL的宿主程序，也就是Host Application设置为“C:\Program Files\Borland\Delphi5\Bin\delphi32.exe”（随着DELPHI安装路径的不同而不同，这里用默认设置）。

其次，为了让被运行的IDE能加载编译后的专家DLL进行源码级调试，也必须在注册表的HKEY\_CURRENT\_USER\Software\Borland\Delphi\5.0\Experts中添加一项加载此DLL的字串值CnWizards\_D5，如CnWizards\_D5的加载项已经存在，则需要改成目标DLL所在的位置。另外DELPHI调试器有个奇怪的规矩，那就是只有项目文件和编译后输出的DLL必须在同一个目录下才能进行源码级的调试，因此需要临时修改CnWizards\_D5的项目设置，将Directories/Conditional里的Output Directory清空，这样编译后的输出DLL就放到了dpr文件的所在目录（原来是在..\..\Bin目录下），注册表中的值CnWizards\_D5也需要设置成输出DLL的全路径，如“D:\CnPackWorks\cnpack\Source\Wizard\CnWizards\_D5.DLL”。

此时便可点击运行按钮进行专家包的调试。如果源文件中下了断点，刚开始运行delphi32.exe时，断点状态是绿色无效的，等到被运行的delphi32.exe的加载了目标DLL后，断点状态也就生效变红了。值得注意的是专家包和被运行IDE在调试的过程中会抛出不少异常，如果IDE的调试器设置了捕获异常并停止，可能会费些周折才能顺利运行，因此可以让IDE调试器不捕获异常。

## 日志记录型工具的使用

### 基本的调试字符串输出技巧

源码级断点单步调试并不是调试的唯一手段。而且很多场合下，下断点进行调试会给程序流程带来影响，甚至某些环境下根本不可行，于是也就出现了很多替代的调试方法，下面介绍的是日志记录型调试工具的使用。

从广义上来说，一切有利于了解程序运行流程和运行期表现的手段都可以称之为调试。很多朋友经常在程序中的一定位置上使用MessageBox等函数来显示运行期某些值，这就是一种基本的调试手段，只不过使用弹出框显示信息在某些场合下有局限性，比如在涉及到窗体重画或比较复杂的用户界面消息处理的代码时，多余的提示框反倒会产生多余的消息，给程序流程带来影响。这种情况下，日志记录型调试就应运而生了。

日志记录型调试属于“事后诸葛亮”型，它需要在被调试程序中使用特定的代码来输出标志流程或调试值的信息，然后通过另外的工具来接受信息并统统记录在案，以备分析程序的运行期信息。WIN32平台本身便提供了类似的API函数OutputDebugString，此函数可以在程序中输出一个字符串到WINDOWS的专用调试字串的存放缓冲区，如果程序在IDE的调试器下运行，那么该字符串会被Event log窗口捕获并显示；单独运行时则会被存在着的日志记录调试器（如DebugView）所捕获；如果什么调试器都没有，则石沉大海无消息了。

日志型调试记录的缺点是需要修改程序源码，在程序中加入信息输出的代码等。虽然可以使用编译开关让这些代码不编译进发布的版本中，但毕竟还是增大了代码量，这是其不方便的地方，因此，很多觉得这种方式比较麻烦的朋友不在必要的时候不使用这种方式，免得费事。

C++中提供了TRACE函数，封装了OutputDebugString来进行调试字符串的输出。Delphi中暂时没有，不过也可以自己写一个Trace函数来实现调试，比如：

procedure Trace(const DebugInfo: string);   
begin   
  OutputDebugString(PChar(DebugInfo));   
end;

这样就可以在形式上和C++实现兼容。

OutputDebugString结合DebugView是常用的一种日志记录型调试方法。由于它是基于Windows内在机制的，和具体的开发环境无关，因此应用比较广泛，在调试后台程序的时候尤其管用。DebugView可以在[http://www.sysinternals.com](http://www.sysinternals.com/)处下载。

### 采用CnDebug单元和CnDebugViewer进行日志输出与记录

OutputDebugString固然方便，但也存在着一些缺点，如信息输出单一，配套的记录工具显示的内容无层次区分等。Delphi的使用者们有个更好的选择：使用uDbg和Overseer，或使用CnPack提供的CnDebug/CnDebugViewer。目前专家包中使用自主编写的后两者。

CnDebug.pas是CnPack开发组开发的一个调试单元，它提供了一个全局CnDebugger对象，这个对象提供了多个记录不同数据类型的记录函数如LogMsg、LogInteger、LogColor、LogPoint等，而且还提供了EnterProc、LeaveProc等函数来标识进入或退出过程的记录。

CnDebugViewer是配合CnDebug单元记录其输出信息的第三方工具，也由CnPack开发组开发维护。它是一个单独的EXE文件，独立于被调试程序之外，拥有显示日志信息的输出窗口，日志信息还可以保存和再次载入以备分析。

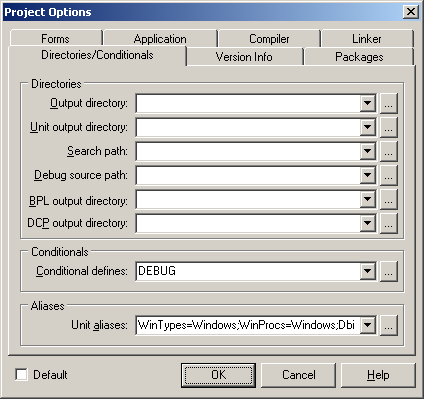
在Delphi程序中，一般使用编译开关在implementation处来uses CnDebug单元并且在需要的地方使用CnDebugger对象的日志方法来输出必要的信息。以下是一个简单的例子：

unit Unit1;   
  
interface   
  
uses   
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,   
  StdCtrls;   
  
type   
  TForm1 = class(TForm)   
    btn1: TButton;   
    procedure FormCreate(Sender: TObject);   
    procedure btn1Click(Sender: TObject);   
  private   
    { Private declarations }   
  public   
    { Public declarations }   
  end;   
  
var   
  Form1: TForm1;   
  
implementation   
  
{$IFDEF DEBUG}   
uses   
  CnDebug;   
{$ENDIF}   
  
{$R \*.DFM}   
  
procedure TForm1.FormCreate(Sender: TObject);   
begin   
  {$IFDEF DEBUG}   
  CnDebugger.LogMsg('FormCreated.');

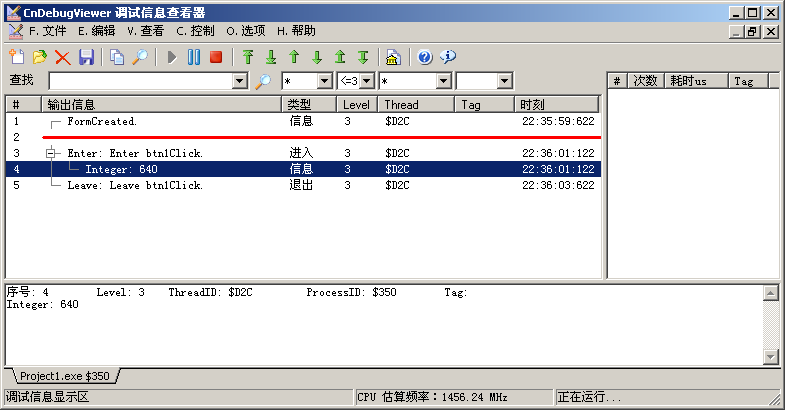
  CnDebugger.LogSeparator;  
  {$ENDIF}   
  // 做其他初始化工作   
end;   
  
procedure TForm1.btn1Click(Sender: TObject);   
begin   
  {$IFDEF DEBUG}   
  CnDebugger.LogEnter('Enter btn1Click.');   
  CnDebugger.LogInteger(Self.Height);   
  {$ENDIF}   
  // 做其他工作   
  
  ShowMessage('Test.');   
  
  {$IFDEF DEBUG}   
  CnDebugger.LogLeave('Leave btn1Click.');   
  {$ENDIF}   
end;   
  
end.

程序本身很简单，仅仅是按一个按钮弹出一个提示框。

欲启用调试功能，需要在Project Options中设置DEBUG条件，如下图所示：



程序运行前，可以先启动CnDebugViewer程序。以下是点击btn1按钮后的日志输出信息：



CnDebugViewer窗口中的每一行都对应着程序中一处输出调试信息的地方。

在这里，{$IFDEF DEBUG} 其实并不是必需的。事实上，如果没有 DEBUG 条件，CnDebugger 的方法不会做任何操作。用户只要在发布版中去掉 DEBUG 条件（或者定义NDEBUG条件。NDEBUG条件在CnDebug单元中优先级更高，能覆盖DEBUG条件），再重新编译一次（注意一定要用 Build 而不是 Compile），所有调试代码都不会有任何作用。

除了前述最基本的日志功能外，用户还可以把对象、组件、Collection、StringList 等直接发送到 CnDebugViewer 中显示出来。另外，CnDebug 里面还提供了其它一些功能，感兴趣的朋友可以自己研究。

如果用户安装了 JCL（Jedi Code Library），还可以在Project Options中设置 DEBUG;USE\_JCL 条件，这样当应用程序发生异常（Exception）时，调试工具会自动记录下异常信息及堆栈调用记录。如果用户编译时生成了 MAP 文件且与输出文件在同一目录下，或是连接时选择了包含 TD32 调试信息以及其它 JCL 调试单元支持的调试符号信息，则在异常的堆栈信息中还可以看到方法名及源代码行号，以方便快速定位错误位置。

### 在CnWizards中使用CnDebug和CnDebugViewer

CnWizards中CnDebug和CnDebugViewer的使用和平常的场合区别不大，只是为了统一起见，目前CVS上的cnpack模块中的CnDebug单元包括进了CnPack.inc文件。如果需要在另外的场合单独使用CnDebug，只要删除文件头部的{ $I CnPack.inc}行即可。另外CnWizards的发布目录的Source目录下也有了已经去除inc引用的CnDebug.pas，可以直接在其他单元中使用。