

CnDebugger 接口帮助文档

作者: Blue++ (zy_hon@hotmail.com)

刘啸 (liuxiao@cnpack.org)

部门: CnPack 开发组

类别: 帮助文档

版本: V0.1.0.7

创建: 2006.09.05

修改: 2008.05.01

一、引言

1、CnDebugger 概述

CnDebugger 是 CnPack 开发组为 Windows 平台上的 Delphi/C++ Builder 开发者提供的一套简要的调试信息记录工具的统称, 包括接口单元和外部信息查看控制工具两大方面的内容。另外也指 CnDebug 单元中的调试对象名称。

CnDebugger 的输出调试信息能进行分类过滤, 可包含四种过滤条件: 文字、层次、类型和标签等。此外, CnDebugger 还提供异常过滤以及输出信息统计等功能。

CnDebugger 的每条输出信息中并不只是简单地包含一字符串, 而是包括了以下可自定义的多种内容:

- 文字: Msg, 字符串格式, 最大长度有限制。
- 层次: Level, 一整数, 作为过滤条件之用。用户可自定义某条输出信息的 Level。Level 按层次包容, 譬如如果指定了内部过滤条件中的 Level 为 2, 那么小于等于 2 的 Level 的信息都能被输出, 大于 2 的则被屏蔽了。目前 Level 定为 0 到 3。默认是 3, 也就是全部输出。所以 Level 可以用来控制输出信息的详细程度。
- 类型: MsgType, 标识该信息的类型, 是普通信息还是警告还是错误等。
- 标签: Tag, 固定长度的字符串格式 (目前最长 8 字节), 用于单项过滤之用。输出时如不指定, 默认是空。

接收端可根据这些信息进行方便的分类显示过滤处理。

CnDebugger 对象正常情况下在 CnDebug.pas 单元 initialization 时创建, 在 finalization 时释放。

2、编译 CnDebug.pas

当不需要使用异常拦截功能时, CnDebug.pas 并不需要其他单元参与编译, 直接加入工程内使用即可。

如果需要异常拦截功能, 则需定义 USE_JCL 宏来进行编译, 打开此条件后, CnDebug.pas

需要 JCL 库才能顺利编译，如果未安装 JCL 库，则需要从 JCL 库中复制 CnDebug.pas 头部注释中列出的一批 JCL 单元后才能编译。编译时打开编译选项 Include TD32 debug Info 或生成 MapFile 可在异常截获时获得更多信息。

CnDebug.pas 还提供了在运行期弹出窗体显示一对象 RTTI 信息的功能，此功能需定义编译条件 SUPPORT_EVALUATE，并且同时需要另外一 CnPack 组件包中自带的单元 CnPropSheetFrm.pas 来参与编译。

3、在 C++Builder 中使用 CnDebug.pas

在 C++Builder 使用 CnDebug.pas 也比较简单，只需要将 CnDebug.pas 加入工程，然后在需要使用的单元头部 #include "CnDebug.hpp" 即可，程序中可通过 CnDebugger()->TraceMsg("AMsg"); 等方式来输出信息了 (hpp 文件会自动生成)。

4、查看 CnDebug.pas 输出的内容

CnDebug.pas 输出的内容需要用 CnDebugViewer 来查看。一般可先运行 CnDebugViewer，然后再运行了包含 CnDebug.pas 的程序，CnDebugViewer 便会自动读取记录并显示输出内容。关于 CnDebugViewer 的更多内容请参考 CnWizards 的帮助文档中的 CnDebugger 部分。

二、CnDebugger

1、CnDebugger 对象功能概述

CnDebugger 对象提供了多个记录不同类型的记录函数如 LogMsg、LogMsgWarning、LogColor、LogStrings 等，而且还提供了 LogEnter/TraceEnter、LogLeave/TraceLeave 等函数来标识进入或退出过程的记录，并能自动转换整数、浮点数、颜色值，并且能用 RTTI 来处理对象与组件以生成输出的属性总结、拦截 Exception 发生时的现场信息记录等。

基本用法：一般在 **implementation** 处 **uses** CnDebug 单元并在 Project Options 中 Conditional Defines 里设置 DEBUG 条件，然后在需要的地方使用 CnDebugger 对象的 Log/Trace 等方法来输出必要的信息。

CnDebugger 提供两类输出接口：Log 系列和 Trace 系列。两类输出接口基本上具有相同的参数和功能。所不同的在于，Log 系列功能只在 DEBUG 编译条件被定义的时候有效，而 Trace 系列在普通情况下也有效。不过在 NDEBUG 定义的情况下两者都无效。

以上写成表达式就是：

Log 有效 := IFNDEF NDEBUG and IFDEF DEBUG

Trace 有效 := IFNDEF NDEBUG

Log 系列输出接口仅在调试时 DEBUG 被定义的情况下参与编译，适合用于产生比较详细的记录的场合；而 Trace 系列输出接口在正式版发布时默认也会被编译入产品中，用于产生关键的调试记录。同时，这两类功能可在 NDEBUG 情况下方便地被全部禁用。

CnDebug.pas 还提供了在运行期弹出窗体显示一对象 RTTI 信息的功能，此功能需定义 SUPPORT_EVALUATE。和上面类似，SUPPORT_EVALUATE 也会在 NDEBUG 已被定义的情况下被 UNDEF 而无效。

另外 CnDebug.pas 的输出信息在输出时可同时记录到文件,此功能需定义 DUMP_TO_FILE, 默认的输出文件名是和应用程序所在文件同一目录下的“CnDebugDump.cdd”。和上面类似, DUMP_TO_FILE 也会在 NDEBUG 已被定义的情况下被 UNDEF 而无效。

DEBUG 等编译条件可在 Project Options 中设置。

2、 所在文件

CnDebug.pas : 调试信息输出接口单元

3、 属性说明

property Channel: TCnDebugChannel;

输出信息发送接口属性,类型为 TCnDebugChannel,指向一个有效的 Channel,只读。TCnDebugChannel 是用来发送数据的实现基类,可参见后文的“CnDebug.pas 的内部 Channel 输出机制”部分。

property Filter: TCnDebugFilter;

当前信息输出的过滤条件,类型为 TcnDebugFilter。它包括四类条件:层次、标签、类型和文字,只读。

property Active: Boolean;

调试功能激活标志,类型为布尔型,可读写。

property ExceptTracking: Boolean;

异常拦截激活标志,类型为布尔型,当此标志和 Active 标志同为激活状态(True)时,异常拦截功能有效,可读写。

property AutoStart: Boolean;

自动启动 Debug Viewer 激活标志,类型为布尔型,当此标志为激活状态时,CnDebugger 在第一次输出信息时自动启动 Debug Viewer(目前启动默认的 CnDebugViewer.exe),可读写。

property DumpToFile: Boolean;

是否在输出信息的同时将信息输出到文件,可读写,可通过 DUMP_TO_FILE 编译条件指定。

property DumpFileName: string;

DumpToFile 为 True 时信息的输出文件名,可读写,默认值是 CnDebugDump.cdd。

property UseAppend: Boolean;

UseAppend 为 True 时,如遇到已经存在的文件,则追加到文件尾,否则重写它。可读写,默认值是 False 也就是重写。

property MessageCount: Integer;

输出消息统计属性,类型为整型,只读。

property PostedMessageCount: Integer;

内部发送消息统计属性，类型为整型，通过 Channel 发送消息的计数器，只读。

property DiscardedMessageCount: Integer;

被丢弃消息统计属性，类型为整型，该值为 MessageCount 与 PostedMessageCount 之差，只读。

4、 方法说明

procedure StartDebugViewer;

功能：启动 Debug Viewer，默认是运行本目录下的 CnDebugViewer.exe 文件并等待其初始化完毕。

参数：

无

procedure StartTimeMark(const ATag: Integer; const AMsg: string = '');

功能：开始一次计时，该功能利用 CPU 周期计时。注意该调用不会输出信息，须等到相应的 StopTimeMark 被调用时才输出计时信息。

参数：

1. **const** ATag: Integer; 计时器标识号。
2. **const** AMsg: string; 附加文字信息，默认为空。

procedure StopTimeMark(const ATag: Integer; const AMsg: string = '');

功能：结束一次计时，将计时结果输出。

参数：

1. **const** ATag: Integer; 待输出的计时器标识号，需要和 StartTimeMark 中的对应。
2. **const** AMsg: string; 待输出的附加文字信息，默认为空。

procedure LogMsg(const AMsg: string);

功能：输出指定文字信息。

参数：

1. **const** AMsg: string; 待输出的文字信息。

procedure LogMsgWithTag(const AMsg: string; const ATag: string);

功能：输出带标签的文字信息

参数：

1. **const** AMsg: string; 待输出的文字信息。
2. **const** ATag: string; 待输出的标签。

procedure LogMsgWithLevel(const AMsg: string; ALevel: Integer);

功能：输出带层次的文字信息

参数：

1. **const** AMsg: string; 待输出的文字信息。
2. ALevel: Integer; 待输出的层次信息。

```
procedure LogMsgWithType(const AMsg: string; AType: TCnMsgType);
```

功能: 输出带消息类型的文字信息

参数:

1. **const** AMsg: **string**; 待输出的文字信息。
2. AType: TCnMsgType; 待输出的消息类型。

```
procedure LogMsgWithTagLevel(const AMsg: string; const ATag: string; ALevel: Integer);
```

功能: 输出带标签和层次的文字信息

参数:

1. **const** AMsg: **string**; 待输出的文字信息。
2. **const** ATag: **string**; 待输出的标签。
3. ALevel: Integer; 待输出的层次信息。

```
procedure LogMsgWithLevelType(const AMsg: string; ALevel: Integer; AType: TCnMsgType);
```

功能: 输出带层次和类型的文字信息

参数:

1. **const** AMsg: **string**; 待输出的文字信息。
2. ALevel: Integer; 待输出的层次信息。
3. AType: TCnMsgType; 待输出的消息类型。

```
procedure LogMsgWithTypeTag(const AMsg: string; AType: TCnMsgType; const ATag: string);
```

功能: 输出带类型和标签的文字信息

参数:

1. **const** AMsg: **string**; 待输出的文字信息。
2. AType: TCnMsgType; 待输出的消息类型。
3. **const** ATag: **string**; 待输出的标签。

```
procedure LogFmt(const AFormat: string; Args: array of const);
```

功能: 根据指定的格式输出信息

参数:

1. **const** AFormat: **string**; 待格式化输出的字符串。
2. Args: **array of const**; 格式化字符串的参数值。

```
procedure LogFmtWithTag(const AFormat: string; Args: array of const; const ATag: string);
```

功能: 根据指定的格式输出带标签的信息

参数:

1. **const** AFormat: **string**; 待格式化输出的字符串。
2. Args: **array of const**; 格式化字符串的参数值。
3. **const** ATag: **string**; 待输出的标签。

```
procedure LogFmtWithLevel(const AFormat: string; Args: array of const;  
ALevel: Integer);
```

功能: 根据指定的格式输出带层次的信息

参数:

1. **const** AFormat: **string**; 待格式化输出的字符串。
2. Args: **array of const**; 格式化字符串的参数值。
3. ALevel: Integer; 待输出的层次信息。

```
procedure LogFmtWithType(const AFormat: string; Args: array of const; AType:  
TCnMsgType);
```

功能: 根据指定的格式输出带类型的信息

参数:

1. **const** AFormat: **string**; 待格式化输出的字符串。
2. Args: **array of const**; 格式化字符串的参数值。
3. AType: TCnMsgType; 待输出的消息类型。

```
procedure LogFull(const AMsg: string; const ATag: string; ALevel: Integer;  
AType: TCnMsgType; CPUPeriod: Int64 = 0);
```

功能: 输出带其他所有内容的信息

参数:

1. **const** AMsg: **string**; 待输出的文字信息。
2. **const** ATag: **string**; 标签信息。
3. ALevel: Integer; 层次信息。
4. AType: TCnMsgType; 类型信息。
5. CPUPeriod: Int64; CPU 周期信息, 默认为 0。

```
procedure LogSeparator;
```

功能: 输出分隔线

参数:

无

```
procedure LogEnter(const AProcName: string; const ATag: string = '');
```

功能: 输出一方法被调用时的信息

参数:

1. **const** AProcName: **string**; 待输出的方法名称信息。
2. **const** ATag: **string**; 待输出的标签, 默认为空。

```
procedure LogLeave(const AProcName: string; const ATag: string = '');
```

功能: 输出一方法调用结束时的信息

参数:

1. **const** AProcName: **string**; 待输出的方法名称信息。
2. **const** ATag: **string**; 待输出的标签, 默认为空。

```
procedure LogMsgWarning(const AMsg: string);
```

功能: 输出警告类型的文字信息

参数:

1. **const** AMsg: **string**; 待输出的文字信息。

procedure LogMsgError(**const** AMsg: **string**);

功能: 输出错误类型的文字信息

参数:

1. **const** AMsg: **string**; 待输出的文字信息。

procedure LogErrorFmt(**const** AFormat: **string**; Args: **array of const**);

功能: 输出错误类型的文字信息, 并格式化输出文字

参数:

1. **const** AFormat: **string**; 待格式化输出的文字信息。
2. Args: **array of const**; 格式化字符串的参数值。

procedure LogLastError;

功能: 输出程序最后一次错误的错误码

参数:

无

procedure LogAssigned(Value: Pointer; **const** AMsg: **string** = '');

功能: 输出指针型变量分配信息(是否为 nil)

参数:

1. Value: Pointer; 待输出的指针变量。
2. **const** AMsg: **string**; 待输出附加文字信息, 默认为空。

procedure LogBoolean(Value: Boolean; **const** AMsg: **string** = '');

功能: 输出布尔型变量

参数:

1. Value: Boolean; 待输出的布尔型变量。
2. **const** AMsg: **string**; 待输出的附加文字信息, 默认为空。

procedure LogColor(Color: TColor; **const** AMsg: **string** = '');

功能: 输出颜色值信息(把 Color 转为文字输出)

参数:

1. Color: TColor; 待转为文字的颜色值。
2. **const** AMsg: **string**; 待输出的附加文字信息, 默认为空。

procedure LogFloat(Value: Extended; **const** AMsg: **string** = '');

功能: 输出浮点型变量值

参数:

1. Value: Extended; 浮点型变量。
2. **const** AMsg: **string**; 待输出的附加文字信息, 默认为空。

```
procedure LogInteger(Value: Integer; const AMsg: string = '');
```

功能: 输出整型变量值

参数:

1. Value: Integer; 整型变量值。
2. **const** AMsg: **string**; 待输出的附加文字信息, 默认为空。

```
procedure LogChar(Value: Char; const AMsg: string = '');
```

功能: 输出字符型变量的 ASCII 码和虚拟键值及其描述字符串等内容

参数:

1. Value: Char; 字符型变量值。
2. **const** AMsg: **string**; 待输出的附加文字信息, 默认为空。

```
procedure LogDateTime(Value: TDateTime; const AMsg: string = '');
```

功能: 输出日期时间值, 默认格式为 “yyyy-mm-dd hh:nn:ss.zzz”

参数:

1. Value: TDateTime; 日期时间值。
2. **const** AMsg: **string**; 待输出的附加文字信息, 默认为空。

```
procedure LogDateTimeFmt(Value: TDateTime; const AFmt: string; const AMsg:  
string = '');
```

功能: 按指定格式输出日期时间值

参数:

1. Value: TDateTime; 日期时间值。
2. **const** AFmt: **string**; 指定的日期时间格式字符串。
3. **const** AMsg: **string**; 待输出的附加文字信息, 默认为空。

```
procedure LogPointer(Value: Pointer; const AMsg: string = '');
```

功能: 以十六进制的格式输出 Pointer 型指针地址

参数:

1. Value: Pointer; 指针变量值。
2. **const** AMsg: **string**; 待输出的附加文字信息, 默认为空。

```
procedure LogPoint(Point: TPoint; const AMsg: string = '');
```

功能: 输出 TPoint 型变量值

参数:

1. Point: TPoint; TPoint 变量值。
2. **const** AMsg: **string**; 待输出的附加文字信息, 默认为空。

```
procedure LogRect(Rect: TRect; const AMsg: string = '');
```

功能: 输出 TRect 型变量值

参数:

1. Rect: TRect; TRect 变量值。
2. **const** AMsg: **string**; 待输出的附加文字信息, 默认为空。

```
procedure LogStrings(Strings: TStrings; const AMsg: string = '');
```

功能: 输出 TStrings 型变量值

参数:

1. Strings: TStrings; TStrings 变量值。
2. **const** AMsg: **string**; 待输出的附加文字信息, 默认为空。

```
procedure LogMemDump(AMem: Pointer; Size: Integer);
```

功能: 输出指定内存块信息

参数:

1. AMem: Pointer; Pointer 变量, 指向某内存块。
2. Size: Integer; 输出大小。

```
procedure LogVirtualKey(AKey: Word);
```

功能: 输出虚拟键值及其描述字符串。

参数:

1. AKey: Word; 虚拟键值。

```
procedure LogVirtualKeyWithTag(AKey: Word; const ATag: string);
```

功能: 输出带标签的虚拟键值及其描述字符串。

参数:

1. AKey: Word; 虚拟键值。
2. **const** ATag: **string**; 待输出的标签。

```
procedure LogObject(AObject: TObject);
```

功能: 输出指定对象信息

参数:

1. AObject: TObject; 指定输出对象。

```
procedure LogObjectWithTag(AObject: TObject; const ATag: string);
```

功能: 根据标签条件输出指定对象信息

参数:

1. AObject: Tobject; 待输出的对象。
2. **const** ATag: **string**; 待输出的标签。

```
procedure LogCollection(ACollection: TCollection);
```

功能: 输出 TCollection 型变量值

参数:

1. ACollection: TCollection; 待输出的 TCollection 变量。

```
procedure LogCollectionWithTag(ACollection: TCollection; const ATag: string);
```

功能: 根据标签条件输出 TCollection 信息

参数:

1. ACollection: TCollection; 待输出的 TCollection 变量。

2. **const** ATag: **string**; 待输出的标签。

procedure LogComponent (AComponent: TComponent);

功能: 输出组件型变量值

参数:

1. AComponent: TComponent; 待输出的组件类型变量。

procedure LogComponentWithTag (AComponent: TComponent; **const** ATag: **string**);

功能: 输出带标签的组件类型变量信息

参数:

1. AComponent: TComponent; 待输出的组件类型变量。

2. **const** ATag: **string**; 待输出的标签。

procedure TraceMsg (**const** AMsg: **string**);

功能: 输出指定文字信息

参数:

1. **const** AMsg: **string**; 待输出的文字信息。

procedure TraceMsgWithTag (**const** AMsg: **string**; **const** ATag: **string**);

功能: 输出带标签的文字信息

参数:

1. **const** AMsg: **string**; 待输出的文字信息。

2. **const** ATag: **string**; 待输出的标签。

procedure TraceMsgWithLevel (**const** AMsg: **string**; ALevel: Integer);

功能: 输出带层次的文字信息

参数:

1. **const** AMsg: **string**; 待输出的文字信息。

2. ALevel: Integer; 待输出的层次信息。

procedure TraceMsgWithType (**const** AMsg: **string**; Atype: TCnMsgType);

功能: 输出带消息类型的文字信息

参数:

1. **const** AMsg: **string**; 待输出的文字信息。

2. Atype: TCnMsgType; 待输出的消息类型。

procedure TraceMsgWithTagLevel (**const** AMsg: **string**; **const** ATag: **string**;
ALevel: Integer);

功能: 输出带标签的和层次的文字信息

参数:

1. **const** AMsg: **string**; 待输出的文字信息。

2. **const** ATag: **string**; 待输出的标签。

3. ALevel: Integer; 待输出的层次信息。

```
procedure TraceMsgWithLevelType(const AMsg: string; ALevel: Integer; AType: TCnMsgType);
```

功能: 输出带层次和类型的文字信息

参数:

1. **const** AMsg: **string**; 待输出的文字信息。
2. ALevel: Integer; 待输出的层次信息。
3. AType: TCnMsgType; 待输出的消息类型。

```
procedure TraceMsgWithTypeTag(const AMsg: string; AType: TCnMsgType; const ATag: string);
```

功能: 输出带消息类型和标签的文字信息

参数:

1. **const** AMsg: **string**; 待输出的文字信息。
2. AType: TCnMsgType; 待输出的消息类型。
3. **const** ATag: **string**; 待输出的标签。

```
procedure TraceFmt(const AFormat: string; Args: array of const);
```

功能: 根据指定的格式输出信息

参数:

1. **const** AFormat: **string**; 待格式化输出的字符串。
2. args: **array of const**; 格式化字符串的参数值。

```
procedure TraceFmtWithTag(const AFormat: string; Args: array of const; const ATag: string);
```

功能: 根据指定的格式输出带标签的信息

参数:

1. **const** AFormat: **string**; 待格式化输出的字符串。
2. Args: **array of const**; 格式化字符串的参数值。
3. **const** ATag: **string**; 待输出的标签。

```
procedure TraceFmtWithLevel(const AFormat: string; Args: array of const; ALevel: Integer);
```

功能: 根据指定的格式输出带层次的信息

参数:

1. **const** AFormat: **string**; 待格式化输出的字符串。
2. Args: **array of const**; 格式化字符串的参数值。
3. ALevel: Integer; 待输出的层次信息。

```
procedure TraceFmtWithType(const AFormat: string; Args: array of const; AType: TCnMsgType);
```

功能: 根据指定的格式输出带消息类型的信息

参数:

1. **const** AFormat: **string**; 待格式化输出的字符串。
2. Args: **array of const**; 格式化字符串的参数值。

3. AType: TCnMsgType; 类型过滤条件。

```
procedure TraceFull(const AMsg: string; const ATag: string; ALevel: Integer;  
AType: TCnMsgType; CPUPeriod: Int64 = 0);
```

功能: 输出带其他所有内容的信息

参数:

1. const AMsg: string; 待输出的文字信息。
2. const ATag: string; 标签信息。
3. ALevel: Integer; 层次信息。
4. AType: TCnMsgType; 类型信息。
5. CPUPeriod: Int64; CPU 周期信息, 默认为 0。

```
procedure TraceSeparator;
```

功能: 输出分隔线

参数:

无

```
procedure TraceEnter(const AProcName: string; const ATag: string = '');
```

功能: 输出一方法被调用时的信息

参数:

1. const AProcName: string; 待输出的方法名称。
2. const ATag: string; 待输出的标签, 默认为空。

```
procedure TraceLeave(const AProcName: string; const ATag: string = '');
```

功能: 输出一方法调用结束时的信息

参数:

1. const AProcName: string; 待输出的方法名称。
2. const ATag: string; 待输出的标签, 默认为空。

```
procedure TraceMsgWarning(const AMsg: string);
```

功能: 输出警告类型的文字信息

参数:

1. const AMsg: string; 待输出的文字信息。

```
procedure TraceMsgError(const AMsg: string);
```

功能: 输出错误类型的文字信息

参数:

2. const AMsg: string; 待输出的文字信息。

```
procedure TraceErrorFmt(const AFormat: string; Args: array of const);
```

功能: 输出错误类型的文字信息, 并格式化输出文字

参数:

1. const AFormat: string; 待格式化输出的文字信息。
2. Args: array of const; 格式化字符串的参数值。

procedure TraceLastError;

功能: 输出程序最后一次错误的错误码

参数:

无

procedure TraceAssigned(Value: Pointer; **const** AMsg: **string** = '');

功能: 输出指针型变量分配信息(是否为 nil)

参数:

1. Value: Pointer; 待输出的指针变量。
2. **const** AMsg: **string**; 待输出的附加文字信息, 默认为空。

procedure TraceBoolean(Value: Boolean; **const** AMsg: **string** = '');

功能: 输出布尔型变量赋值信息

参数:

1. Value: Boolean; 待输出的布尔型变量。
2. **const** AMsg: **string**; 待输出的附加文字信息, 默认为空。

procedure TraceColor(Color: TColor; **const** AMsg: **string** = '');

功能: 输出颜色值信息(把 Color 转为文字输出)

参数:

1. Color: TColor; 待转为文字的颜色值。
2. **const** AMsg: **string**; 待输出的附加文字信息, 默认为空。

procedure TraceFloat(Value: Extended; **const** AMsg: **string** = '');

功能: 输出浮点型变量值

参数:

1. Value: Extended; 浮点型变量。
2. **const** AMsg: **string**; 待输出的附加文字信息, 默认为空。

procedure TraceInteger(Value: Integer; **const** AMsg: **string** = '');

功能: 输出整型变量值

参数:

1. Value: Integer; 整型变量值。
2. **const** AMsg: **string**; 待输出的附加文字信息, 默认为空。

procedure TraceChar(Value: Char; **const** AMsg: **string** = '');

功能: 输出字符型变量的 ASCII 码和虚拟键值及其描述字符串等内容

参数:

1. Value: Char; 字符型变量值。
2. **const** AMsg: **string**; 待输出的附加文字信息, 默认为空。

procedure TraceDateTime(Value: TDateTime; **const** AMsg: **string** = '');

功能: 输出日期时间值, 默认格式为 “yyyy-mm-dd hh:nn:ss.zzz”

参数:

1. Value: TDateTime; 日期时间值。
2. **const** AMsg: **string**; 待输出的附加文字信息, 默认为空。

```
procedure TraceDateTimeFmt(Value: TDateTime; const AFmt: string; const AMsg: string = '' );
```

功能: 按指定格式输出日期时间值

参数:

1. Value: TDateTime; 日期时间值。
2. **const** AFmt: **string**; 指定的日期时间格式字符串。
3. **const** AMsg: **string**; 待输出的附加文字信息, 默认为空。

```
procedure TracePointer(Value: Pointer; const AMsg: string = '' );
```

功能: 以十六进制的格式输出 Pointer 型指针地址

参数:

1. Value: Pointer; 指针变量值。
2. **const** AMsg: **string**; 待输出的附加文字信息, 默认为空。

```
procedure TracePoint(Point: TPoint; const AMsg: string = '' );
```

功能: 输出 TPoint 型变量值

参数:

1. Point: TPoint; TPoint 变量值。
2. **const** AMsg: **string**; 待输出的附加文字信息, 默认为空。

```
procedure TraceRect(Rect: TRect; const AMsg: string = '' );
```

功能: 输出 TRect 型变量值

参数:

1. Rect: TRect; TRect 变量值。
2. **const** AMsg: **string**; 待输出的附加文字信息, 默认为空。

```
procedure TraceStrings(Strings: TStrings; const AMsg: string = '' );
```

功能: 输出 TStrings 型变量值

参数:

1. Strings: TStrings; TStrings 变量值。
2. **const** AMsg: **string**; 待输出的附加文字信息, 默认为空。

```
procedure TraceMemDump(AMem: Pointer; Size: Integer);
```

功能: 输出指定内存块的内容

参数:

1. AMem: Pointer; Pointer 变量, 指向某内存块。
2. Size: Integer; 输出大小。

```
procedure TraceVirtualKey(AKey: Word);
```

功能: 输出虚拟键值及其描述字符串。

参数:

1. AKey: Word; 虚拟键值。

procedure TraceVirtualKeyWithTag(AKey: Word; **const** ATag: **string**);

功能: 输出带标签的虚拟键值及其描述字符串。

参数:

1. AKey: Word; 虚拟键值。
2. **const** ATag: **string**; 待输出的标签。

procedure TraceObject(AObject: TObject);

功能: 输出指定对象信息

参数:

1. AObject: TObject; 指定输出对象。

procedure TraceObjectWithTag(AObject: TObject; **const** ATag: **string**);

功能: 根据标签条件输出指定对象信息

参数:

1. AObject: TObject; 待输出的对象。
2. **const** ATag: **string**; 待输出标签。

procedure TraceCollection(ACollection: TCollection);

功能: 输出 TCollection 型变量值

参数:

1. ACollection: TCollection; 待输出的 TCollection 变量。

procedure TraceCollectionWithTag(ACollection: TCollection; **const** ATag: **string**);

功能: 根据标签条件输出 TCollection 信息

参数:

1. ACollection: TCollection; 待输出的 TCollection 变量。
2. **const** ATag: **string**; 待输出的标签。

procedure TraceComponent(AComponent: TComponent);

功能: 输出组件型变量值

参数:

1. AComponent: TComponent; 待输出的组件类型变量。

procedure TraceComponentWithTag(AComponent: TComponent; **const** ATag: **string**);

功能: 根据标签条件输出组件类型变量信息

参数:

1. AComponent: TComponent; 待输出的组件类型变量。
2. **const** ATag: **string**; 待输出的标签。

```
procedure AddFilterExceptClass(E: ExceptClass); overload;
```

功能: 添加待过滤异常类的类, 添加后此异常类被抛出时不捕捉。

参数:

1. E: ExceptClass; 待添加的过滤的异常类, 添加后此异常类被抛出时不捕捉。

```
procedure RemoveFilterExceptClass(E: ExceptClass); overload;
```

功能: 删除待过滤异常类的类, 删除后此异常类会被捕捉。

参数:

1. E: ExceptClass; 待删除的异常类, 删除后此异常类会被捕捉。

```
procedure AddFilterExceptClass(const EClassName: string); overload;
```

功能: 添加待过滤异常类的类名, 添加后此异常类被抛出时不捕捉。

参数:

1. const AMsg: string; 待添加的异常类的类名, 添加后此异常类被抛出时不捕捉。

```
procedure RemoveFilterExceptClass(const EClassName: string); overload;
```

功能: 删除待过滤异常类的类名, 删除后此异常类会被捕捉。

参数:

1. const EClassName: string; 待删除的异常类的类名, 删除后此异常类会被捕捉。

```
procedure EvaluateObject(AObject: TObject); overload;
```

功能: 弹出类似于 Object Inspector 的窗体显示某对象的 RTTI 信息。此过程需要定义条件 SUPPORT_EVALUATE, 否则为空过程。

参数:

1. AObject: TObject; 待显示信息的对象实例。

```
procedure EvaluateObject(APointer: Pointer); overload;
```

功能: 同上, 弹出类似于 Object Inspector 的窗体显示某指针的 RTTI 信息, 该指针将被强制转换成对象。此过程同样需要定义条件 SUPPORT_EVALUATE, 否则为空过程。

参数:

1. APointer: Pointer; 待显示信息的指针, 过程内将被强制转换成对象实例。

三、 CnDebug.pas 的内部 Channel 输出机制

CnDebug.pas 内部使用了一 TCnDebugChannel 实例来发送数据到接收端。TCnDebugChannel 是一抽象类, 被 CnDebugger 使用, 实现基本的启动 Viewer, 发送数据块, 等待发送完毕等功能。CnDebug.pas 内部已经实现了它的一个子类 TCnMapFileChannel, 该子类使用内存共享映射文件来完成数据的复制与发送, CnDebugViewer 也使用同样的约定接口来读取发送的数据, 因此可以说, TCnMapFileChannel 是和 CnDebugViewer 进行实际数据交互的实现类。

CnDebug.pas 中有这么一句:

```
var
```

```
CnDebugChannelClass: TCnDebugChannelClass = TCnMapFileChannel;
```

这句指明了 CnDebugger 当前使用的 Channel 默认是 TCnMapFileChannel, 也即默认就发送到 CnDebugViewer 端。

如果用户需要输出到别的内容, 可自行实现另外一 TCnDebugChannel 的子类(比如叫 TMyChannel)并在 CnDebug.pas 单元的 initialization 部分写上如下代码便可指定使用此 Channel:

```
CnDebugChannelClass := TMyChannel;
```

据此, 用户可通过写一 TCnDebugChannel 的新的子类来实现将输出数据内容转存到文件等功能。

以下是 TCnDebugChannel 类的声明与说明:

```
TCnDebugChannel = class(TObject)
{* 信息输出 Channel 的抽象类}
private
  FAutoFlush: Boolean;
  FActive: Boolean;
  procedure SetAutoFlush(const Value: Boolean);
protected
  procedure SetActive(const Value: Boolean); virtual;
  // 供子类重载以处理 Active 变化
  function CheckReady: Boolean; virtual;
  // 检测是否准备好
  procedure UpdateFlush; virtual;
  // AutoFlush 属性更新时供子类重载以进行处理
public
  constructor Create(IsAutoFlush: Boolean = True); virtual;
  // 构造函数, 参数为是否自动送出并等待接收完成
  procedure StartDebugViewer; virtual;
  // 启动 Debug Viewer
  function CheckFilterChanged: Boolean; virtual;
  // 检测过滤条件是否改变
  procedure RefreshFilter(Filter: TCnDebugFilter); virtual;
  // 过滤条件改变时重新载入
  procedure SendContent(var MsgDesc; Size: Integer); virtual;
  // 发送信息内容
  property Active: Boolean read FActive write SetActive;
  // 是否激活
  property AutoFlush: Boolean read FAutoFlush write SetAutoFlush;
  // 是否自动送出并等待接收方接收
end;
```

需要说明的是，TCnDebugChannel 中使用的 SendContent 过程是整个 CnDebugger 发送数据所使用的基础方法，其参数 MsgDesc 符合 TCnMsgDesc 结构所定义的内容：

```
{ $NODEFINE TCnMsgAnnex }
TCnMsgAnnex = packed record
{ * 放入数据区的每条信息的头描述结构 }
    Level: Integer; // 自定义 Level 数, 供用户过滤用
    Indent: Integer; // 缩进数目, 由 Enter 和 Leave 控制
    ProcessId: DWORD; // 调用者的进程 ID
    ThreadId: DWORD; // 调用者的线程 ID
    Tag: array[0..CnMaxTagLength - 1] of Char; // 自定义 Tag 值, 供用户过滤用
    MsgType: DWORD; // 消息类型
    MsgCPIInterval: DWORD; // 计时结束时的 CPU 周期数
    TimeStampType: DWORD; // 消息输出的时间戳类型
    case Integer of
        1: (MsgDateTime: TDateTime); // 消息输出的时间戳值 DateTime
        2: (MsgTickCount: DWORD); // 消息输出的时间戳值 TickCount
        3: (MsgCPUPeriod: Int64); // 消息输出的时间戳值 CPU 周期
    end;

{ $NODEFINE TCnMsgDesc }
{ $NODEFINE PCnMsgDesc }
TCnMsgDesc = packed record
{ * 放入数据区的每条信息的描述结构, 包括一信息头 }
    Length: Integer; // 总长度, 包括信息头
    Annex: TCnMsgAnnex; // 一个信息头
    Msg: array[0..CnMaxMsgLength - 1] of Char; // 需要记录的信息
end;
```

用户需要自行解析它们。解析可参考 CnDebug 头部常量结构等的完整定义与 CnDebugViewer 的数据读取线程的实现代码。

另外，CnDebugger 的 protected 方法 InternalOutputMsg 是所有输出接口最终会调用到的方法，它内部组装了数据块并最终调用了 Channel 的 SendContent 方法来发送数据块。如果用户想免去解析数据块的麻烦，可修改 CnDebug.pas 中的此方法并自行书写代码来重定向到别需要的地方，具体实现用户可自由发挥。