inTrace

What is inTrace?

InTrace is a debugging tool that captures and displays the **trace** of a program execution. A trace is a collection of **trace-events**.

A trace-event is either a routines-entry, a routines-exit, an exception or a debug string. Each trace-event has a high precision time-stamp.

To limit the number of generated trace-events, inTrace allows filtering events related to specific routines, as for example, events from routines contained in a given unit, events from routines of a given class or events from specific routines.

InTrace allows specifying a **trigger** condition that controls the trace-events capture. When the trigger is reached, inTrace displays the trace-events that occurred before as well as after the trigger.

The trigger as well as the filter can be modified without restarting the traced application.

InTrace displays trace-events so that elements generated by different threads appear in different colors. InTrace provides navigation commands that allows to skip from one event to another related event, as for example "to the next thread switch" or to the corresponding routine-exit point etc...Double clicking on an element displays the corresponding source code line.

InTrace can save the trace either to a text file or to an html file.

InTrace is particularly useful to:

Debug multithreaded situations

Measure the execution time of routines.

See what happens behind the scene, especially when using third parties components.

Debug applications that cannot be debugged in the Delphi IDE.

InTrace does not require changes to the source code.

-o-

Quick start

Start Delphi and load an application.

Select **Project/Options/Linker** and check **Map file/Detailed**.
Build the application.
Save the project.

Start inTrace, select **Project/Load Dpr...** navigate to the application project (dpr) file and click open. The Filter-page displays the default settings, which are **Include source Trace source .**

Select **Trigger/Set**, and check **Condition/On any routine** and **Thread/Any**

Select **Run/Run**

The application is started. The trigger condition is reached immediately (any routine) and trace-events are collected. When the status bar indicates ''Trace Full'' then the trace-events are displayed into the trace-window.

The status bar may also indicates ''Trigged Remaining=xxx", meaning that the trace-buffer is not yet full. You can either use the application in order to generate more trace-events or select **View**/**Trace** to display the trace-events collected so far.

From then on, you could for example change the trigger using **Trigger**/**Set**, selecting **OnRoutine** and clicking the routine browse button. In the routine dialog, select for example a menu-item event handler. Then perform **Run**/**Reset.** The status bar indicates "Waiting for trigger (...)". Using the application, select the menu-item for which a trigger is set. Etc...

-o-

How does it work?

When the loaded application is run (**Run/Run**) inTrace does the following:

It processes the **Include** and **Exclude** statements of the filter-page and replaces the five first bytes of all included routines by a jump to a specific place. This process is called binary code preparation.

It processes the **Trace** and **NoTrace** statements of the filter-page and enable tracing of all the specified routines.

From then on, inTrace collects (in a circular trace-buffer) the trace-events generated by the application's execution. For each event, it checks if the trigger condition is satisfied. If yes, it continues the events acquisition until the trace-buffer contains the specified percentage of pre- and post-trigger events.

As soon as the trigger is hit, the user can display (**View/Trace**) the events collected so far and possibly change the trigger and/or the filter and finally, reset the trigger (**Run/Reset**).

-o-

The filter-page

The filter-page contains statements that specify how inTrace should handle the routines of the traced application. There are four kinds of statements paired in two groups: **Include** vs. **Exclude**, and **Trace** vs. **NoTrace**.

Include specifies routines that must have binary code preparation. Only such routines can be traced.

Exclude specifies routines that must be completely ignored by inTrace. Of course, excluded routines cannot be traced.

Include and **Exclude** statements are only processed <u>when the application is started.</u>

Trace specifies routines that must be traced. Only included routines can be traced.

Similarly, **NoTrace** specifies routines that must not be traced.

Trace and **NoTrace** statements are processed when the application is started as well as when the trigger is reset. Therefore, the set of traced and untraced routines can be adapted as many times as needed during a single inTrace session.

The syntax is the same for all the statements.

A statement can specify a unit in which case all routines contained in the unit are concerned:

Include unit 'MyUnit'

A statement can specify a class in which case all routines of the class are concerned:

Exclude class 'Classes:TList'

A statement can also specify a single routine:

Include routine 'Classes:TList.GetCount'

Units, classes and routines names must be specified in single quotes.
A class specification has the format '<unit name> : <class name>'
A routine specification has one of the formats:

```
'<unit name> : < class name> . <routine name>'
'<unit name> : <routine name>'
```

The first variant is used when the routines belong to a class.

To specify all routines at once write:

Include all

To specify all routines with a reachable source file write:

Include source

The filter-page is processed top down and may contain any number of statements in any order. The resulting set of (included, excluded, traced and untraced) is affected by each statement. Therefore, in the following example...

include all
Trace unit 'Unit1'
Trace unit 'Unit2'
Trace unit 'Unit3'
NoTrace all
Trace routine 'Unit1:TClass.R'

...the result is that only routine R will be traced.

Several statements can be factorized out in the following way:

```
Trace (
  unit 'Unit1'
  class 'Unit1:TClass'
  routine 'Unit1:TClass.R'
)
```

Using this syntax, complete sets of routines can be handled at once.

-o-

Setting the trigger

Setting the trigger consists of selecting a trace-event kind and selecting a thread. Both the trace-event kind and the thread must match to fill the trigger condition.

The trace-event kind is one of the followings:

On Routine

The trigger occurs when the specified routine is entered. The full routine name can be directly specified in the edit-field. Clicking the browse button opens the routine-list dialog where the desired routine can be selected. A specific routine can be found by directly typing its name in the routine-list.

On any Routine

The trigger occurs when any routine is entered.

On exception

The trigger occurs when the specified exception is raised. Clicking the browse button opens the exception dialog where the desired exception can be selected.

On any exception

The trigger occurs when any exception is raised.

On debug string

The trigger occurs when the specified debug string is output. The specified string must be exactly the same as the `OutputDebugString` argument.

On any debug string

The trigger occurs when any debug string is output.

The selected thread can be:

IdString

The thread identified by the specified IdString is selected.

Any

Any thread.

-o-

The Trace Window

The trace-window displays a numbered list of trace-events. The trigger-event is displayed in **bold**. Events of different threads are displayed in different colors. The trace-windows has 5 columns:

Time

Is the time stamp of the event relative to the trigger at a resolution of one microsecond.

Thread

Is either a numeric value known as the thread identifier or the string specified by an OutputDebugString('!identification') statement. The prefix ''(T)'' indicates that the thread is terminated.

Event

There are several kinds of events.

a routine-entry is displayed as a '+' followed by the full routine name.

a routine-exit is displayed as a '-' followed by the full routine name.

a routine execution (without intervening events) is displayed as a '|' followed by the full routine name.

an exception is displayed as the word "Exception" followed by a string indicating the kind of exception.

a debugsstring is displayed as the word "DebugString" followed by its string value.

The routine related events have an indentation reflecting the call stack level.

M

Is a numeric value indicating a navigation mark.

H

Is a numeric value indicating that so many events that were displayed after this event are hidden.

-o-

Trace navigation and trace context-menu

The trace navigation as well as other trace related actions are controlled by the trace context menu displayed when clicking right in the trace-window. All actions are related to the selected item, which is called the **current trace-event**.

ToTrigger

Skip to the trigger trace-event

To Routine Enter/Leave

Skip to the corresponding entering or leaving trace-event

ToDelta

Skip to a past or future trace-event. In the edit window, enter the delta-time (in milliseconds) between the current trace-event and a future (positive value) or past (negative value) event. The delta-time can be a fractional number.

To Thread switch

Skip to the next or previous thread switch.

To...

Skip to a specified trace-event. The corresponding dialog allows specifying a text string, a thread and a direction.

Show Source

If the corresponding source-file is reachable, then load it in the source page and shows the current positions.

Show Source of Caller

If possible, load the source-file of the caller and shows the calling position.

Hide This

If the current trace-event is a routine-entry then hide all nested trace-event. The **H** column displays the number of hidden trace-events.

Show This

If the current trace-event is followed by hidden trace-events (**H** column contains a number) then displays the hidden trace-events.

NoTrace

This command adds a **NoTrace** statement to the filter.
NoTrace/**Unit** adds the line:

```
NoTrace unit 'Unit of the current trace-event'
```

NoTrace/**Class** adds the line:

```
NoTrace class 'Class of the current trace-event'
```

NoTrace/**Routine** adds the line:

```
  NoTrace routine 'Routine of the current trace-event'
```

Flip Mark
Flip the (book) mark at the current trace-event. The column M displays the mark number.

Goto Mark
Go to the trace-event with the specified mark.

-o-

Thread identification and color

Windows assign a unique numeric identifier to each thread it creates. However, these identifiers are not necessarily constant across different executions of the same application. Therefore, it is convenient (but not mandatory) for a thread to identify itself with a constant identification string. Then, when inTrace needs to display a thread identifier, it uses instead the identification string.

To identify itself, a thread must execute (preferably in its first statements) an OutputDebugString() of the following format:

```
OutputDebugString('!IdentificationString');
```

Note the exclamation mark that prefixes the identification string.

By default, inTrace assigns a different color to trace-events generated by different threads. However, any thread can force a specific color if the following way:

```
OutputDebugString('!IdentificationString;RRGGBB');
```

RR, GG and BB are the color components in hexadecimal format. Example:

```
OutputDebugString('!RedThread;FF0000');
```

-o-

The routine view

This view displays all the routines of the application. The list can be sorted by clicking the columns (**unit**, **routine**, **address**) headers.

Routines for which no source file is reachable are marked with an asterisk (*).

A specific routine can be found by entering its name or part of its name in the upper edit control and clicking the **locate** button.

Right clicking on a routine opens a context menu with the following commands:

View source

Load the routine file name in the source page and shows the routine. Double clicking on a routine has the same effect.

Copy unit

Copy the unit name of the selected routine to the clipboard, from where it can be pasted (for example) into the filter-page.

Copy class

Copy the class name of the selected routine to the clipboard.

Copy routine

Copy the routine name of the selected routine to the clipboard.

-o-

The Main Menu

Project/Load dpr...
Load a Delphi project specified by a ".dpr" file

Project/Load bin...
Load a previously saved state (".bin") file.

Project/Save bin...
Save the current session settings (project, filters, trigger etc...) to a state file.

Project/Reload
Reload a previous project or state file.

Project/Export
Export the content of the trace-window to a file. The file may be a text file (extension ".txt") or an html file (extension ".html").

Project/Exit

Run/Run
Run the tested application.

Run/Reset
Reset the trigger.

Run/Terminate
Terminate the tested application.

Trigger/Set

View/Trace
Displays the current content of the trace-buffer.

View/Routines
Display the routine window.

Options

-o-

The options

At shutdown, save the current state

Before shutdown, save the state of the current loaded project to a (.bin) file. By default, the state is saved in the same directory as the executable and has the same name as the executable.

At startup, reload the last saved state

Automatically reload the last save state file.

Stay on top

InTrace stays on top.

Text size

Specify the text size used in the pages and in the trace-window.

Run parameters

String passed as parameter string when the application is launched.

Startup directory

String passed as startup directory when the application is launched.

$(DELPHI) value

Directory pointed to by the '$(DELPHI)' macro.

This macro is used in the Delphi IDE at **project/options/Directories/Search path** to specify directories where source files are located. If this macro is used (as for example in $(DELPHI)\source\vcl), then inTrace has to know its value to be able to reach the related source files.

Trace events quantity

This is the size of the trace-event buffer, the value must be in the range [10..200000]. Changes take effect only when the tested application is restarted.

Trigger position

The value in % of the trace-events that are kept before the trigger. A value of (say) 10% means that, when the buffer is full, at least 10% of the buffer should contains trace-events that occurred before the trigger.]. Changes take effect only when the tested application is restarted.

-o-

The Log page

The log page is filled when the application is loaded for the first time. It contains a list of the reachable source files as well as a list of units for which no source files were found.

The source page displays source code.

-o-

Control of the data acquisition from the traced application

The trace-event acquisition can be controlled from the traced application using OutputDebugString calls:

```
OutputDebugString(PChar('!+'));   // Enable acquisition
OutputDebugString(PChar('!-'));   // Disable acquisition
```

-o-