ProDelphi64 Handbuch

(Release 43.x für 64-Bit-Anwendungen)

Der Profiler für Delphi XE2 .. XE8, 10 .. 10.4, 11, 12
(Für Pentium und kompatible CPUs)

Profiling

Der Zweck von ProDelphi64 ist es, herauszufinden, welche Teile eines Programms die meiste CPU-Zeit verbrauchen. Da Borland seinen Turbo Profiler eingestellt hat, musste ein neues Werkzeug geschaffen werden. ProDelphi64 mit seinem komfortablen Viewer, Browsern, History-Funktion und Programmier-API kann mittlerweile mehr als der legendäre Turbo Profiler. Der Viewer mit seinen sortierten Ergebnissen ermöglicht es dem Benutzer, die Flaschenhälse in seinem Programm sehr schnell zu finden. Die History-Funktion zeigt ihm, ob eine Optimierung erfolgreich war oder nicht. ProDelphi64's herausragende Granularität macht es möglich, auch zeitkritische Abläufe zu optimieren. Die eingebaute Kalibrierung eliminiert den Overhead der Messroutinen aus den Messergebnissen.

Ab Version 8.0 ist die dynamische Aktivierung der Messung sehr einfach zu bedienen. Statt API-Aufrufe in den Quellcode einzubauen (die Methode war in allen früheren Versionen möglich und ist es immer noch), können jetzt in einem praktischen Dialog die aktivierenden Methoden ausgewählt werden.

Mit dem Release 13.0 kam die Aufrufstruktur (Caller/Called Grafik), der die Navigation durch die Messergebnisse (in Verbindung mit der Öffnung der Quelldatei im IDE-Editor durch Mausklick) zum reinen Vergnügen macht. Die Startpunkt-Liste macht es einfach, zu optimierende Pfade zu identifizieren.

Post Mortem Analyse

Ein weiterer Grund für die Entwicklung von ProDelphi64 war die Notwendigkeit ein Werkzeug zu bekommen, das den Aufrufstack eines Programms im Falle einer Exception zeigt. ProDelphi64 ermittelt die Funktion, in der die Exception auftrat ohne, dass das Testprogramm unter der IDE gestartet werden muss.

Die Unterschiede zwischen der Freeware- und der Professional-Version

Mit der Freeware-Version können bis zu 20 Methoden gemessen werden, in der Professional-Version bis zu 65500.

In der Professional-Version können zusätzlich minimale und maximale Laufzeiten der Methoden im Viewer angezeigt werden. Außerdem kann der User-Teil des Librarypfades gemessen werden.

Datum: 10.7.2025

Der Inhalt dieser Beschreibung

BEVOR Sie ProDelphi64 benutzen, lesen Sie bitte Kapitel A.3.2 und A.16 !

A. Profiling

A.1. Einführung

Der zu optimierende Quellcode des Programms wird instrumentiert mit Aufrufen in eine Zeit-Mess-Unit. Die Einfügungen werden am Anfang und am Ende einer Methode gemacht.

Jedes Mal, wenn eine Prozedur / Funktion (nachfolgend Methode genannt) aufgerufen wird, wird die Startzeit der Methode gespeichert. Am Ende der Methode wird die verbrauchte Zeit berechnet. **Wenn das Programm beendet wird,** werden zwischen drei und fünf Dateien erstellt, die die Laufzeit-Informationen zu den einzelnen Methoden enthalten.

Die **erste** Datei (programname.txt) enthält die verstrichene Zeit in CPU-Zyklen. Das Format ist ASCII, getrennt durch ein Semikolon (;) und kann entweder für den **Datenbank-Import** oder für den **integrierten Viewer von ProDelphi64** verwendet werden. Das Format wird am Ende der vorliegenden Beschreibung beschrieben.

Die **zweite** Datei (programname.tx2) enthält weitere Informationen wie Überschriften und wie oft Messungen an die erste Datei angehängt wurden. Die Informationen werden durch das Online-Bedien-Fenster oder durch die Programmier-API erzeugt.

Die **dritte** Datei (programname.tx3) enthält Informationen zum Öffnen einer Datei im IDE-Editor und zur Positionierung des Editor-Cursors an den Anfang einer Methode.

Die **vierte** Datei (programname.nev) enthält die Namen aller Methoden, die während der Messung der Laufzeit Ihres Programms nicht aufgerufen wurden. Sie wird vom Viewer verwendet, der die Methoden als hierarchische Baumstruktur anzeigt, wenn Sie den Button mit dem Namen "Nicht aufgerufen“ anklicken. Dieser Button ist nicht aktiviert, wenn alle Methoden aufgerufen wurden.

Die **fünfte** Datei ist ebenfalls optional und wird nur dann erstellt, wenn die automatische Deaktivierung aktiviert ist (siehe A.5).

A.2. Basic Profiling

Die Benutzung von ProDelphi64 ist ganz einfach. Es wurde in einem Projekt mit einem großen Programm mit mehr als 370000 Zeilen (mit Code von 12 Programmierern) eingesetzt. Nach mehr als zwei Jahren Entwicklung wurde das Programm mithilfe von ProDelphi optimiert. Die Programmlaufzeit konnte um ca. 50 % gesenkt werden.

Verwenden Sie das Setup-Programm, um ProDelphi64 zu installieren. Das Setup-Programm kann nur dann richtig funktionieren, wenn weder Delphi noch eine frühere Version von ProDelphi(64) gestartet sind. Wenn Sie ein Update von einer früheren Version von ProDelphi64 installieren wollen, müssen Sie zuerst die alte Version deinstallieren. Für diese Verwendung muss das Setup-Programm im alten Installationsverzeichnis verwendet werden.

Nach der Installation kompilieren Sie Ihr Programm, damit die Delphi-Projekt-Datei erstellt wird. **Wenn keine Projekt-Datei vorhanden ist, müssen alle Dateien im gleichen Verzeichnis sein (*. PAS, *. INC, *. DPR, *. EXE und *.DLL).**

Nur wenn Sie Methoden in einem Programm und in einer DLL gleichzeitig messen wollen:

> *Es müssen Programm und DLL identische Sourcecode-Suchpfade haben. Sowohl DPR-Dateien als auch EXE-Dateien müsse jeweils im gleichen Verzeichnis sein. In diesem Fall beides kompilieren: Programm- und DLL. Alle Dateien zu instrumentieren müssen in den Verzeichnissen liegen, die in den für DLL und Programm gleichem Suchpfad genannt werden (mit Ausnahme derjenigen, für die ein expliziter Pfad in der USES-Anweisung in DPR-Dateien von Programm oder DLL angegeben ist). Um Programm und DLL gleichzeitig zu messen, muss die Option Programm + DLL' gesetzt sein (siehe auch Kapitel A9).*

Wenn Ihre zu instrumentierenden Dateien sehr groß sind und sie in der IDE geöffnet sind, sollten Sie geschlossen werden. Es wurde berichtet, dass Delphi die auf der Festplatte geänderten Dateien manchmal nicht im Editor aktualisiert und damit nicht kompiliert.

Wenn bei der Kompilierung keine Fehler auftreten, können Sie Ihr Programm *(und / oder DLL) instrumentieren.*

Verwenden Sie nicht die Originaldateien für die Instrumentierung, für den Fall, dass ProDelphi64 noch Bugs enthält. Sie sollten Ihre Daten sichern, z. B. durch Zippen und archivieren aller PAS-, DPR und INC-Dateien.

Für die Messung der Laufzeit Ihres Programms führen Sie die folgenden Schritte aus:

- Definieren Sie das Compiler-Symbol PROFILE (Projekt / Optionen / Bedingungen).

- Deaktivieren der Option ‘Optimierung’ wird empfohlen (siehe auch A.7.1).

- Optional deaktivieren Sie alle Laufzeitprüfungen.

- Verwenden Sie das Delphi-Kommando "Alles speichern". Dies stellt sicher, dass die Projekt-Datei gespeichert ist.

- Starten Sie ProDelphi64 über das Delphi-Tools-Menü.

- Mit ProDelphi64 wählen Sie das zu messende Programm aus (wenn es nicht schon ausgewählt ist).

- Für den Anfang sollten nur jene Optionen, die im folgenden Beispiel gewählt wurden, benutzt werden.

Folgende Optionen sind nur in der Professional-Version verfügbar:

- Messen von Units im Library-Pfad
- Ermittlung von minimalen und maximalen Laufzeiten (in der Freeware-Version sind nur die viel wichtigeren durchschnittlichen Laufzeiten verfügbar).
- Dateidatum und Attribute nicht verändern. Diese Option bewirkt, dass das Dateidatum beim Instrumentieren um 2 Sekunden erhöht wird. Das ist gerade genug, dass Delphi realisiert, dass sich eine Datei geändert hat. Das Dateidatum wird beim Entfernen der Instrumentierung wieder zurückgestellt.

Übrigens: Die wichtigsten Buttons sind 'First Steps' und 'Handbuch'!

- Wählen Sie die Art der Aktivierung für die Messung aus, die Sie möchten (in diesem Beispiel durch Programmstart).

- Wenn das Profiling Log nicht angezeigt werden soll, den Button ‘Log anzeigen’ deaktivieren (Professional Version).

- Klicken Sie auf den ‘Instrumentieren’-Button. Nach einer sehr kurzen Zeit sind alle Units instrumentiert. Die instrumentierten Dateien werden im Log-Fenster aufgelistet.

- *Wenn Sie die Methoden in DLL’s messen möchten, müssen gleichzeitig auch die notwendigen DLL’s instrumentiert werden.*

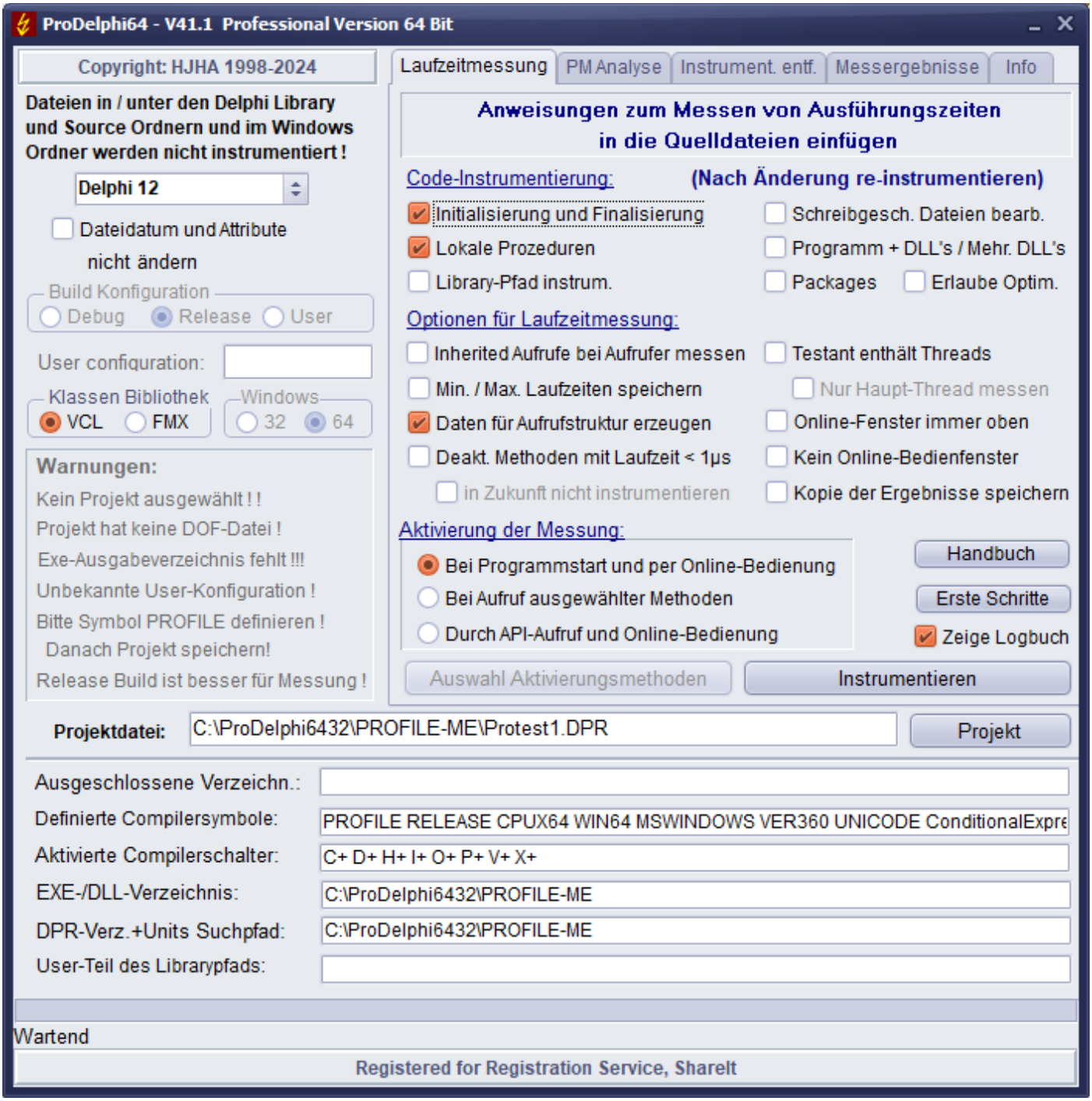
- Kompilieren Sie das Programm *(und / oder die DLL).*

Um eine gleichzeitige Messung von DLL und Programm zu ermöglichen, werden alle Dateien im Suchpfad instrumentiert! (Es sei denn, sie sind schreibgeschützt!). Der Suchpfad muss exakt gleich sein für Programm und DLL, beide DPR-Dateien müssen ebenfalls im gleichen Verzeichnis sein!

Dateien in und unter den Delphi LIB- und SOURCE-Verzeichnissen werden nicht instrumentiert!!!

Danach starten Sie das Programm und lassen Sie es seine zu optimierende Aufgabe ausführen.

Es erscheint ein kleines Fenster, das Sie zum Starten und Stoppen der Zeitmessung verwenden können.



Abhängig von den Optionen für die Aktivierung der Messung ist der Button ‘Start’ aktiviert (Keine Aktivierung bei Programmstart) oder nicht (mit Aktivierung bei Programmstart). Mit Aktivierung bei Programmstart beginnt die Messung mit dem Start des zu testenden Programms. Wenn die Messung nicht gestartet ist, können Sie sie mit dem Start-Button aktivieren. Außerdem können Sie sie mit API-Aufrufen starten oder indem Sie Methoden definieren, die bei Ihrem Aufruf die Messung starten.

Nachdem das Programm beendet ist, können Sie

die Ergebnisse der Messung mit dem integrierten Viewer von ProDelphi64 anschauen.

Um den eingebauten Viewer zu benutzen, starten Sie einfach ProDelphi64 und gehen zum Tab ‘Messergebnisse’. Wenn der Name des Projekts nicht automatisch angezeigt wird, wählen Sie es aus. Dann klicken Sie auf den Button 'Laden und anzeigen'.

Im Prinzip ist dies alles, was getan werden muss. Wenn Sie möchten, dass das Programm ohne Zeitmessung läuft, löschen Sie einfach das Compiler-Symbol PROFILE in den Delphi-Optionen und machen eine komplette Kompilierung.

A.2.1. Dateien, die von ProDelphi64 oder dem gemessenen Programm erzeugt werden

ProDelphi64 erstellt die Datei 'proflst.asc', sie enthält Informationen über die Methoden, die für die Profilierung gemessen werden oder für ein Post Mortem Review getracet werden. Die Datei profile.ini enthält Optionen für die Zeitmessung und die letzten Bildschirm-Koordinaten des Online-Bedien-Fensters. Der Viewer kann eine Datei namens '*. Hst' erzeugen, wenn Sie die History-Funktion (siehe A.4.1) verwenden.

Ihr kompiliertes Programm erzeugt die Datei mit dem Namen 'progname.txt', sie enthält die Messdaten im ASCII-Format (durch Semikolon getrennte Werte) für Datenbank-Export und Viewer. Außerdem 'progname.tx2' für die Überschriften der verschiedenen Zwischenergebnisse für den eingebauten Viewer. Die Datei 'programname.tx3' ist für die Schnittstelle zur Delphi-IDE erzeugt. Die Datei 'progname.swo' enthält die Liste der Methoden, die für die Zeitmessung beim nächsten Programmstart automatisch deaktiviert werden (wenn die Option gewählt wurde). Auch eine Datei mit dem Namen 'progname.nev' wird erstellt, in der die Namen der nicht aufgerufenen Methoden gespeichert werden. Diese Datei wird vom Viewer verwendet.

Ihr Programm erstellt eine Datei namens 'progname.pmr', falls Sie die Post Mortem Review ausgewählt haben und eine Exception aufgetreten ist. Sie enthält den Aufruf-Stack. Die Datei wird in dem Ausgabe-Verzeichnis für die *. EXE gespeichert.

Um eine gleichzeitige Messung von DLL und Programm zu ermöglichen, werden alle Dateien im Units-Suchpfad (mit Ausnahme der Delphi LIB und SOURCE Verzeichnisse und darunter) instrumentiert, wenn sie nicht schreibgeschützt sind! Suchpfad für Programm und DLL müssen in diesem Fall identisch sein.

A.2.2. Die Überprüfung der Ergebnisse mit dem integrierten Viewer

Der bequemste Weg um die Laufzeiten der Methoden zu prüfen ist den integrierten Viewer zu verwenden. Klicken Sie einfach auf den Reiter ‘Messergebnisse’ und dann auf den Button ‘Laden und anzeigen’.

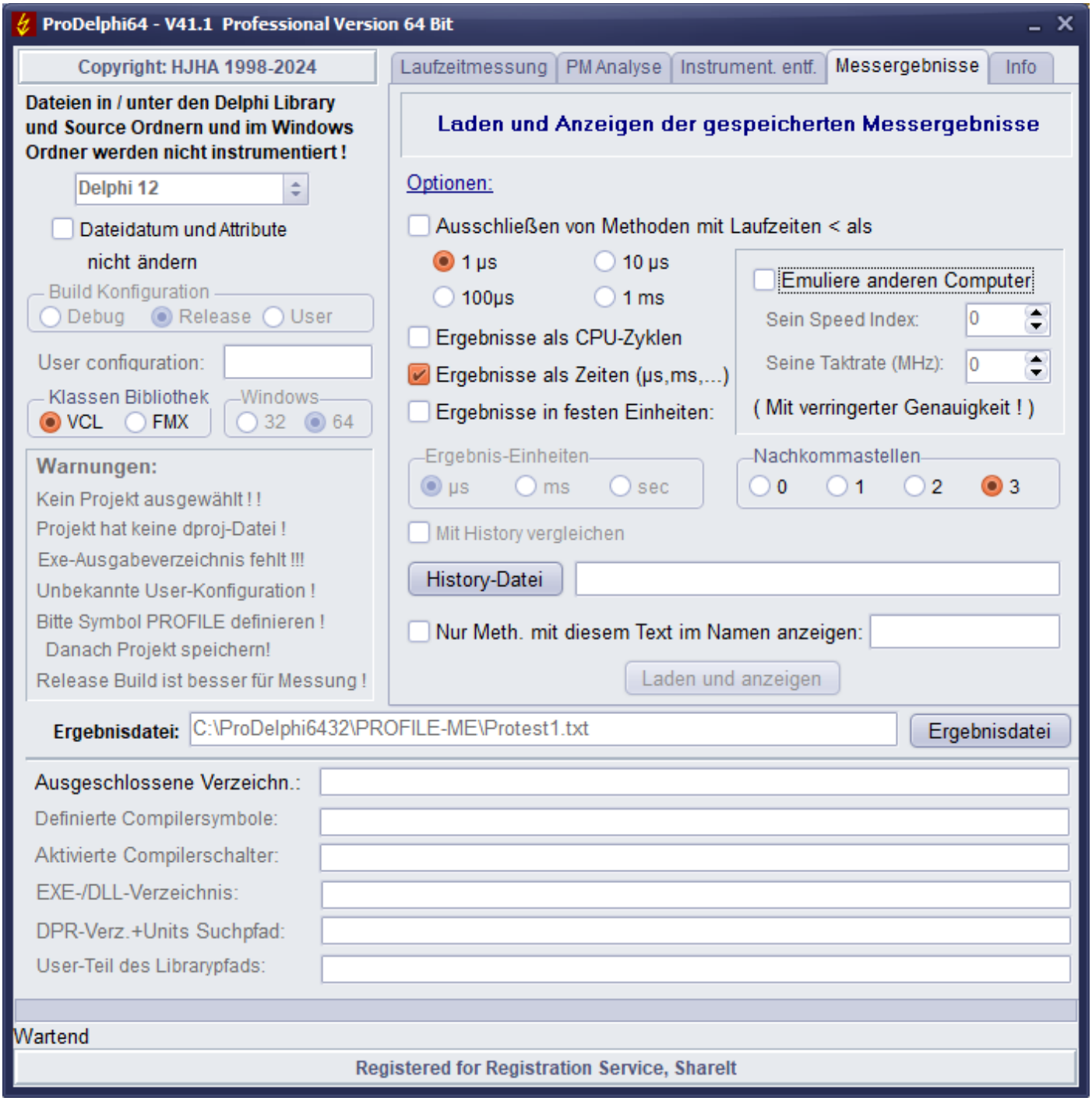
Die Messergebnisse wurden in der Ergebnis-Datei abgespeichert, entweder bei Ende des getesteten Programms oder bei Anklicken des ‘Add’-Buttons des Online-Bedien-Fensters.

Sie können wählen, ob Sie die Ergebnisse in µs, ms und sec oder in CPU-Zyklen sehen möchten.

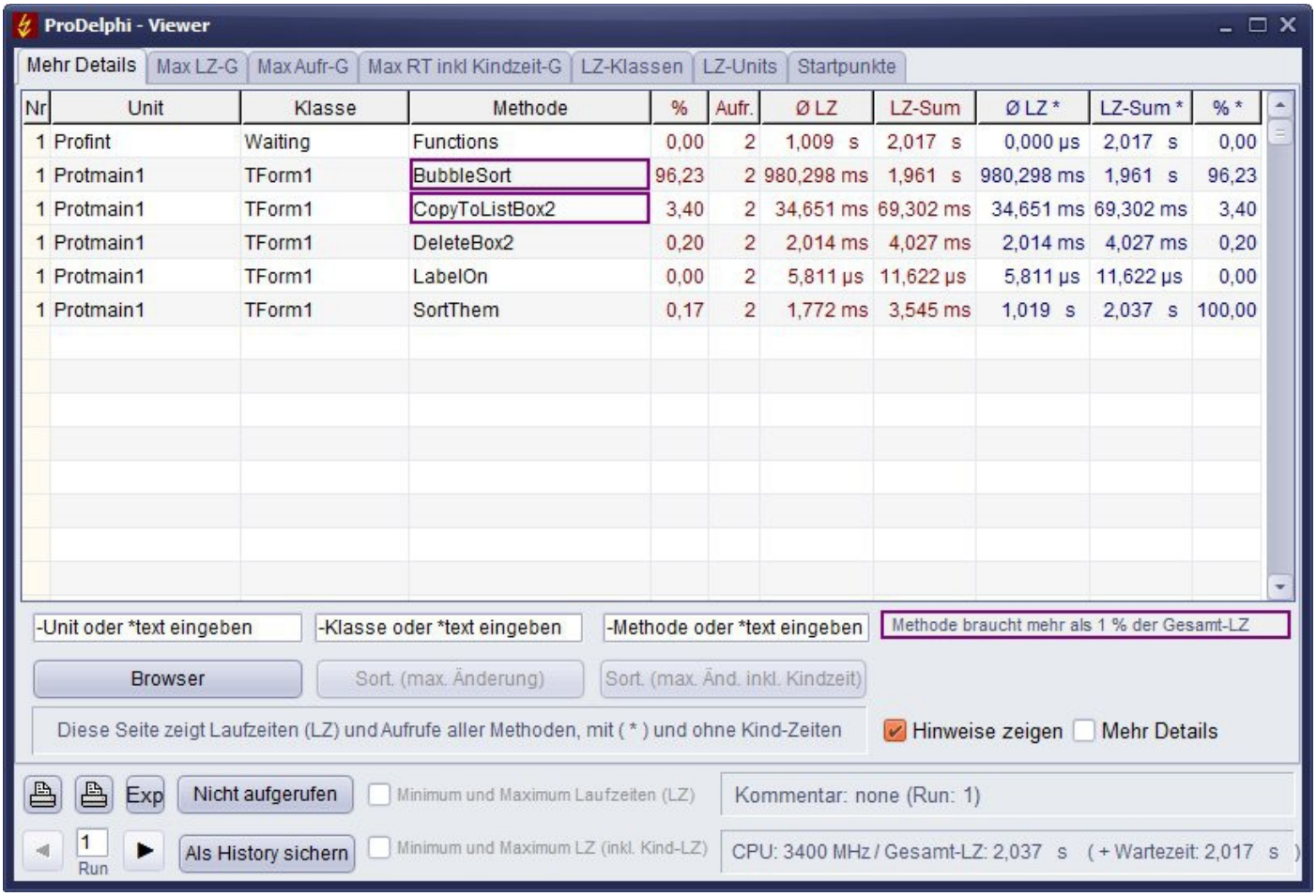
Sie können Methoden mit Laufzeiten von weniger als 1 µs, 10 µs, 100 µs oder 1 ms ausblenden.

Sie können auch einen schnelleren oder langsameren PC emulieren. Die Messungen werden auf die Verarbeitungsgeschwindigkeit des fremden PCs umgerechnet. Keine Notwendigkeit, die IDE auf diesem PC zu installieren! Geben Sie einfach zwei Konstanten in einem Eingabefeld ein und lassen Sie ProDelphi64 Ihnen sagen, wie schnell oder wie langsam Ihr Programm auf diesem PC sein würde (siehe Kapitel A.2.3) .

Bei Klick auf 'Laden und Anzeigen' werden Tabellen mit den Ergebnissen der Messungen angezeigt. Sie können durch die Ergebnisse blättern oder z. B. nach einer bestimmten Unit, Klasse oder Methode suchen.



Alphabetisch sortierte Ergebnisse: erst nach Unit, dann nach Klasse und zuletzt nach Methode



Erläuterung dieses Fensters:

CPU: nnn MHZ zeigt die CPU - Geschwindigkeit.
Insgesamt RT: ttt zeigt die Laufzeit aller gemessenen Methoden (alternativ in CPU-Zyklen)
Wartezeit: Erscheint nur, wenn das Programm ohne etwas zu tun, auf etwas wartet (z.B. Sleep- oder MessageBox Funktion)
Kommentar: ccccc im Online-Bedien-Fenster eingetragener Kommentar bei Zwischenergebnissen oder 'At Finishing Application', wenn die Ergebnisse automatisch bei Programmende gespeichert wurden oder Datum und Uhrzeit, wenn die Ergebnisse zyklisch gespeichert wurden.

Sortieren der Tabelle:

Die angezeigte Tabelle kann nach verschiedenen Kriterien durch Klicken auf die Reiter gefüllt werden, versuchen Sie es einfach! Zwei zusätzliche Tasten dienen dazu, um die gemessenen Detailergebnisse nach Änderung gegenüber der gespeicherten History zu sortieren. Die Ergebnisse werden derart sortiert, dass die Methoden, deren Laufzeiten sich am meisten geändert haben, oben angezeigt werden (siehe auch History).

Die Detail-Daten könne durch Anklicken der Tabellenüberschrift nach verschiedenen Kriterien sortiert werden. Die angezeigte Reihenfolge kann durch ein weiteres Anklicken umgekehrt werden.

Navigation durch die Ergebnisse:

Navigieren durch die Ergebnisse kann durch Scrollen erfolgen, über den Browser oder durch die Suche nach Unit, Klasse oder Objekt.

Die Suche wird durch Eingabe von Text in das Suchfeld gestartet. Mit der F3-Taste kann weiter gesucht werden, auch die Positionierung durch Blättern nach oben und unten ist möglich.

Navigieren vom Viewer zum Quellcode:

Klick mit der rechten Maustaste in eine Zeile des Viewers lässt Delphi die Datei im Editor öffnen und den Cursor auf den Anfang der Methode positionieren. Für die Nutzung dieser Funktion muss Delphi gestartet sein.

Die Print - Buttons:

Gedruckt wird die momentan angezeigte Tabelle (ggf. mit Grafik). Die Tabelle wird automatisch justiert, sodass sie auf das Papier passt. Mit dem ersten Button wird alles in Schwarz gedruckt und nur wenn es unbedingt erforderlich ist auch Farbe verwendet (Color Save-Modus). Mit dem zweiten Button wird alles gedruckt, wie auf dem Bildschirm angezeigt (Full Color Mode).

Der Exp - Button:

Der Inhalt der angezeigten Tabelle wird in eine CSV-Datei exportiert. Die Werte werden durch ';' getrennt. Die exportierten Daten können dann in Excel, LibreOffice oder OpenOffice importiert werden.

Die Minimum / Maximum Checkboxen (Professional-Version):

Aktivieren Sie diese Optionen, um minimale und maximale Laufzeiten anzuzeigen (wenn sie bei der Messung gesammelt wurden), siehe Kapitel A.7.2. Wenn die Checkbox 'Min./Max. Laufzeiten speichern' im ProDelphi64 Hauptfenster zur Programmlaufzeit deaktiviert war, wurden keine Minimal- und Maximalwerte gesammelt.

Der History Button: siehe Kapitel A.4.1

Bedeutung der **Run-Spalte:**

Jedes Mal, wenn das Programm Daten in die Ergebnisdatei speichert, speichert es eine führende Zahl zu den gemessenen Zeiten: die Nummer der Messung. Mit dem ◄ (Zurück) - oder ► (Weiter) - Button können Sie zwischen verschiedenen Messungen wechseln. Auch ist es möglich, die Nummer direkt in das Eingabefeld zwischen den Buttons ◄ und ► einzutragen.

Beim Start des Programms wird immer die erste Messung angezeigt.

Bedeutung der Spalten mit dem roten Text:

%	Prozentsatz der gesamten Laufzeit der Methode ohne Kind-Methoden
Aufr.	Wie oft die Methode aufgerufen wurde
ØLZ	Durchschnittliche Laufzeit der Methode in CPU-Zyklen oder in µs, ms, sec oder Stunden-Einheiten (in der Professional-Version können auch minimale und maximale Laufzeiten angezeigt werden).
LZ-Sum	ØLZ x Aufrufe

Bedeutung der Spalten mit dem blauen Text:

ØLZ*	Durchschnittliche Laufzeit der Methode, einschließlich der aufgerufenen Methoden (Kind Methoden) in CPU-Zyklen oder in µs, ms, ... (In der Professional-Version können auch minimale und maximale Laufzeiten angezeigt werden)
LZ-Sum*	ØLZ* x Aufrufe
%	Prozentsatz der gesamten Laufzeit der Methode inklusive ihrer Kind-Methoden.

Bedeutung des ◄-Buttons und des ►-Buttons:

Wenn Ihr Programm Zwischenergebnisse in die Ergebnis-Datei geschrieben hat (mit der ProDelphi64-API oder
durch das Online-Bedien-Fenster) können Sie in der Ergebnisdatei vorwärts oder rückwärts blättern.

Bedeutung **'Kommentar':**

Es ist der Kommentar, der eingestellt war, als die Messung gespeichert wurde. Im Beispiel sehen Sie die Standardeinstellung.

Die anderen verfügbaren Tabellen-Seiten zeigen:

Die 12 sortierten Methoden, die den größten Teil der Laufzeit **(ohne** Kind-Methoden) verbrauchten als Text und grafische Darstellung.

Die 12 sortierten Methoden, die am meisten aufgerufen wurden als Text und grafische Darstellung.

Die 12 sortierten Methoden, die den größten Teil der Laufzeit **(mit** Kind-Methoden) verbrauchten als Text und grafische Darstellung.

Die 12 Klassen, die die meiste Laufzeit konsumierten.

Die 12 Units, die die meiste Laufzeit konsumierten.

Bedeutung der Laufzeiten in einem roten Rahmen:

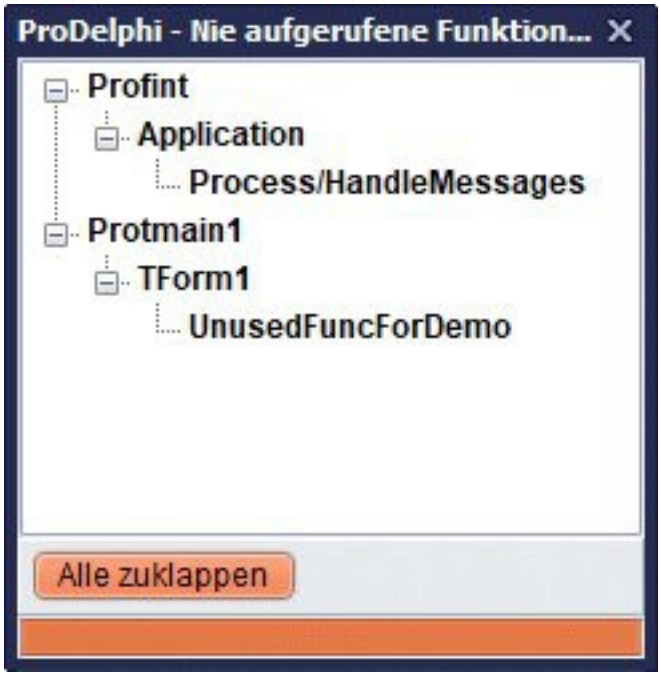
Die Laufzeit ist größer als die Zeit, die in der History-Datei gespeichert wurde. Der Rahmen wird nur dann angezeigt, wenn die Änderung größer ist als 1 % der gesamten Laufzeit der Applikation.

Bedeutung der Laufzeiten in einem grünen Rahmen:

Die Laufzeit ist kleiner als die Zeit, die in der History-Datei gespeichert wurde. Der Rahmen wird nur dann angezeigt, wenn die Änderung größer ist als 1 % der gesamten Laufzeit der Applikation.

Der ‘Nie aufgerufen’ - Button:

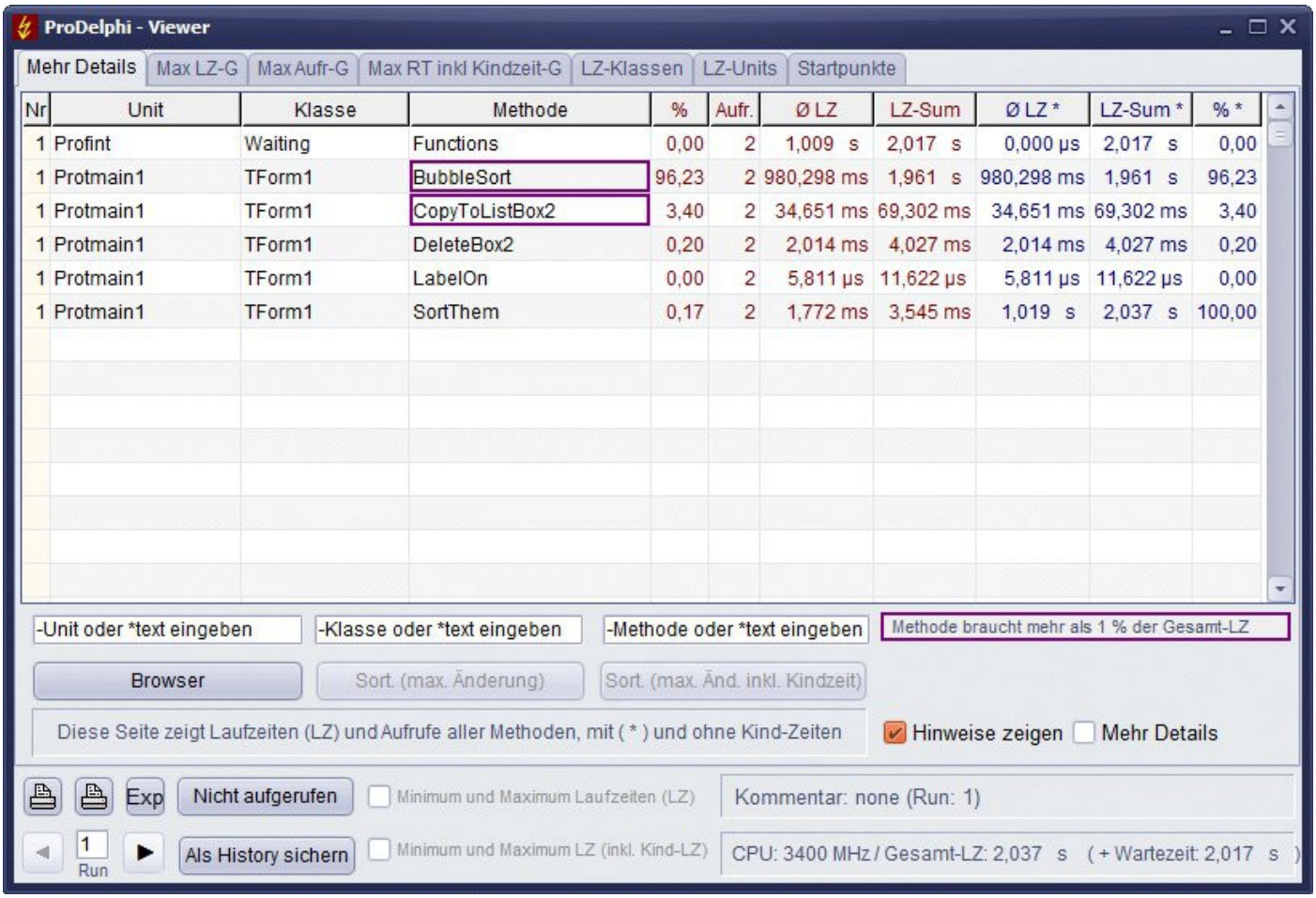
Am Ende der Laufzeit erstellt der Testant eine Datei mit den Namen aller nicht aufgerufenen Methoden. Mit diesem Button werden diese Methoden in einer hierarchischen Darstellung angezeigt: Unit - Klasse – Methode.



Der Browser - Button:

Es öffnet sich ein kleines Browser-Fenster (ähnlich dem Explorer), das Units, Klassen und Methoden in einer hierarchischen Darstellung zeigt. Es kann verwendet werden, um schnell die Ergebnisse der Messung für eine bestimmte Methode zu finden.

Hier ein anderes Viewer- Beispiel:



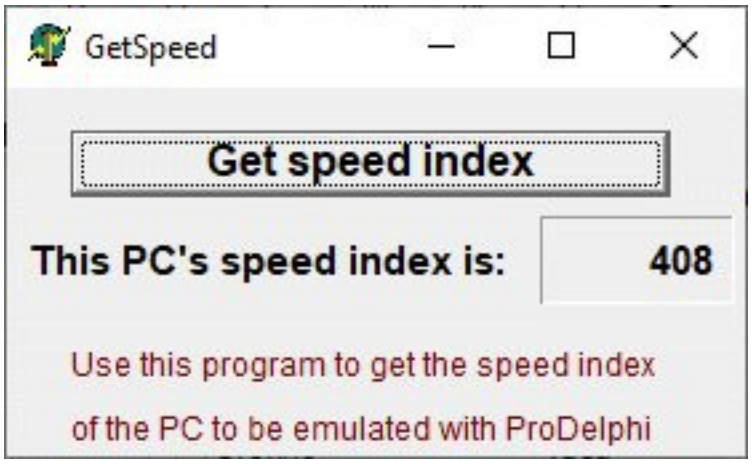
Beispiel: Maximale Laufzeit konsumierende Methoden (grafisch)

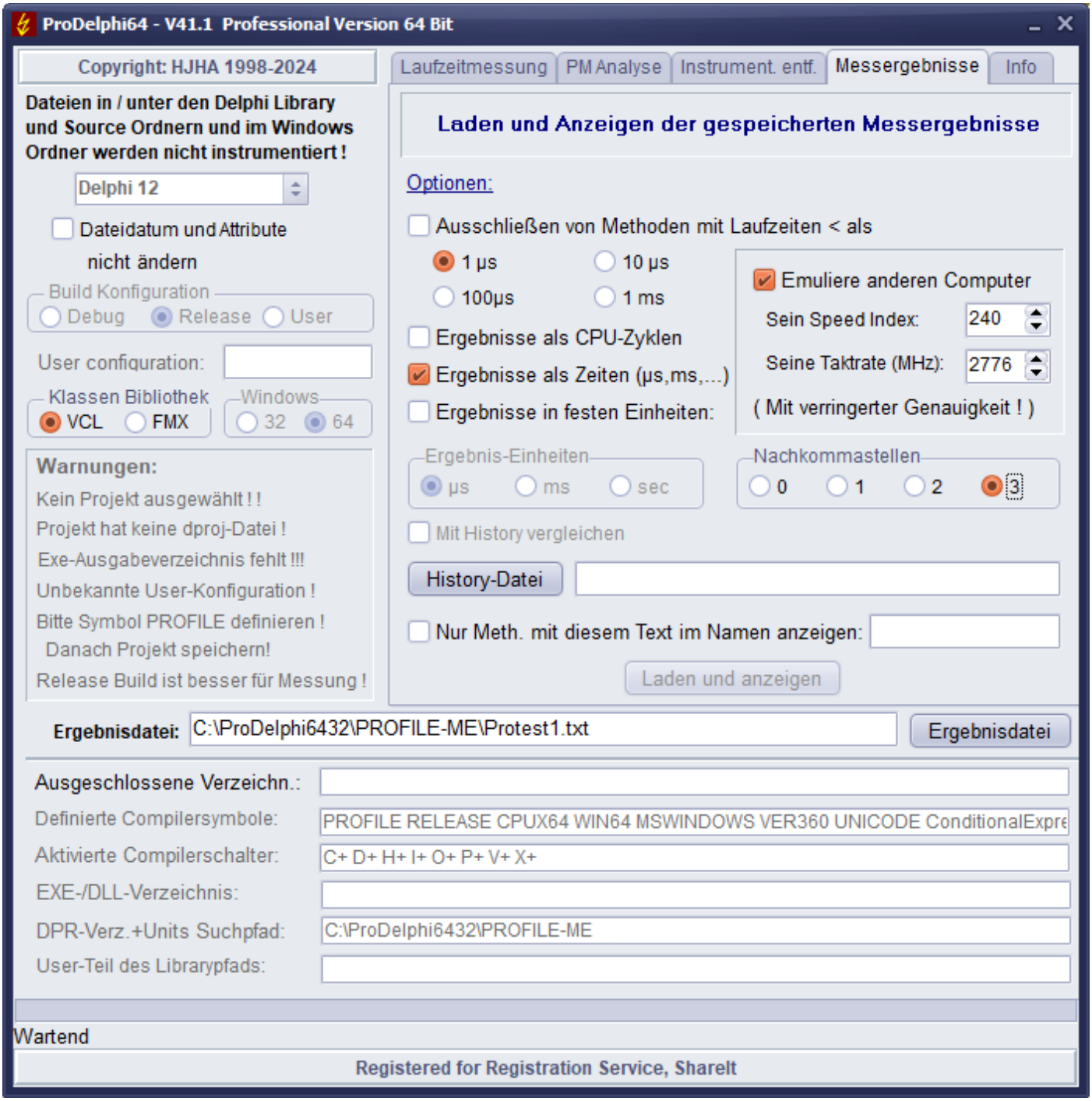
A.2.3. Emulation eines schnelleren oder langsameren PCs

Wenn Sie wissen wollen, wie schnell (oder langsam) Ihr Programm auf einem anderen PC ausgeführt werden würde, verwenden Sie einfach das Programm Getspeed.exe, um den Speed-Index des anderen PCs zu erhalten. Diesen geben Sie, ebenso wie dessen CPU-Geschwindigkeit, in ProDelphi64 ein, nachdem Sie das Häkchen 'Emuliere anderen Computer' im Viewer gesetzt haben. Automatisch werden dann beim Laden der Ergebnisse alle Messungen für den anderen PC umgerechnet. ***Sicherlich sind die Ergebnisse nicht so genau, wie wenn sie tatsächlich gemessen würden.***

Nutzungseinschränkung: Wenn Sie in Ihrem Programm eine Methode, die einen definierten Zeitraum (z. B. für 1 sek.) laufen soll, ist das Emulationsergebnis für diese Methode falsch!

(Der Speed Index und die CPU-Geschwindigkeit des PCs, auf dem ProDelphi64 ausgeführt wird, wird automatisch berechnet und wird von ProDelphi64 benötigt. Deshalb GetSpeed nicht löschen!!!)

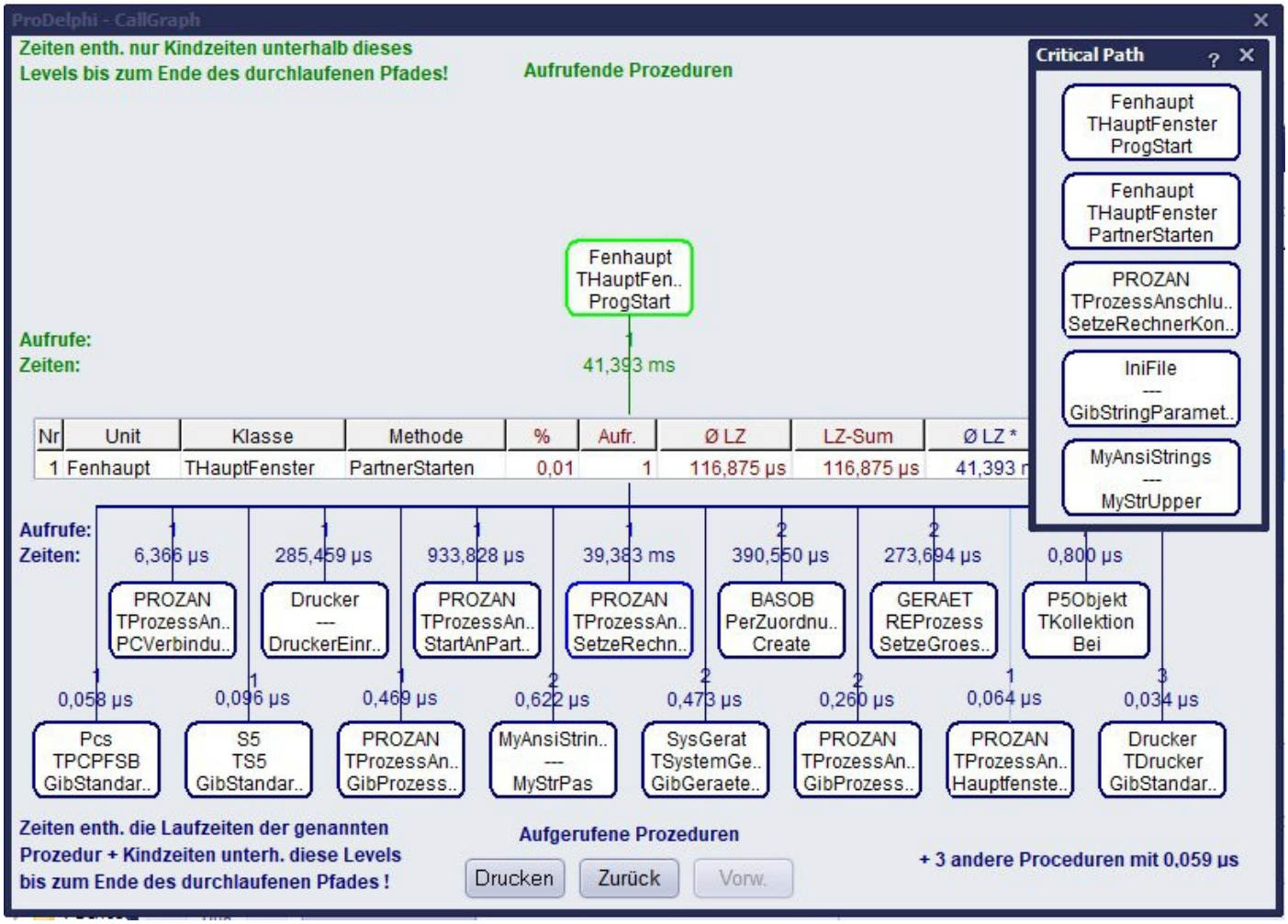



ProDelphi64 - V41.1 Professional Version 64 Bit
Copyright: HJHA 1998-2024
Laufzeitmessung
PM Analyse
Instrument. entf.
Messergebnisse
Info
Dateien in / unter den Delphi Library und Source Ordnern und im Windows Ordner werden nicht instrumentiert !
Delphi 12
Dateidatum und Attribute nicht ändern
Build Konfiguration
Debug
Release
User
User configuration:
Klassen Bibliothek
VCL
FMX
Windows
32
64
Warnungen:
Kein Projekt ausgewählt ! !
Projekt hat keine dproj-Datei !
Exe-Ausgabeverzeichnis fehlt !!!
Unbekannte User-Konfiguration !
Bitte Symbol PROFILE definieren !
Danach Projekt speichern!
Release Build ist besser für Messung !
Laden und Anzeigen der gespeicherten Messergebnisse
Optionen:
Ausschließen von Methoden mit Laufzeiten < als
1 µs
10 µs
100µs
1 ms
Ergebnisse als CPU-Zyklen
Ergebnisse als Zeiten (µs,ms,...)
Ergebnisse in festen Einheiten:
Emuliere anderen Computer
Sein Speed Index:
240
Seine Taktrate (MHz):
2776
(Mit verringerter Genauigkeit !)
Ergebnis-Einheiten
µs
ms
sec
Nachkommastellen
0
1
2
3
Mit History vergleichen
History-Datei
Nur Meth. mit diesem Text im Namen anzeigen:
Laden und anzeigen
Ergebnisdatei:
C:\ProDelphi6432\PROFILE-ME\Protest1.txt
Ergebnisdatei
Ausgeschlossene Verzeichn.:
Definierte Compilersymbole:
PROFILE RELEASE CPUX64 WIN64 MSWINDOWS VER360 UNICODE ConditionalExpre
Aktivierte Compilerschalter:
C+ D+ H+ I+ O+ P+ V+ X+
EXE-/DLL-Verzeichnis:
DPR-Verz.+Units Suchpfad:
C:\ProDelphi6432\PROFILE-ME
User-Teil des Librarypfads:
Wartend
Registered for Registration Service, SharelT

A.2.4. Verwenden der Aufrufstruktur (Caller/Called Graph)

Wenn bei der Messung Daten für die Anzeige der Aufrufstruktur gesammelt wurden, kann die Aufrufstruktur des Programms angezeigt werden. Wenn Sie mit der linken Maustaste auf ein Messergebnis im Viewer klicken, öffnet sich ein neues Fenster mit den Laufzeiten der gewählten Methode in einer Tabellenzeile der Mitte des Formulars.

Oberhalb der Tabelle werden bis zu 15 Methoden angezeigt, die die gewählte Methode aufgerufen haben. Wenn es mehr als 15 aufrufende Methoden gibt, wird dies am oberen Rand des Formulars angezeigt. Es werden immer die Methoden angezeigt, die die meiste Laufzeit konsumiert haben. Für jede Methode werden die Anzahl der Aufrufe für die gewählte Methode und ihre Laufzeit inklusive aller Kindzeiten angezeigt.



Unterhalb der Tabelle werden bis zu 15 Methoden angezeigt, die durch die gewählte Methode aufgerufen wurden. Auch hier werden die Methoden, die die meiste Laufzeit verbraucht haben angezeigt, jeweils mit der Anzahl der Anrufe und der verbrauchten Laufzeit inklusive aller Kindzeiten angezeigt **(Screen Shot ist nicht aus dem Beispiel-Programm in diesem Handbuch)** :

Das "Critical Path" Fenster zeigt die Aufruf-Sequenz mit der höchsten Ausführungszeit, beginnend mit der Methode, die in der Tabelle angezeigt wird.

Ein Linksklick auf das Symbol für eine aufgerufene oder aufrufende Methode bewirkt, dass diese Methode in der Tabelle angezeigt wird.

Ein Linksklick auf die Tabelle bewirkt, dass der kritische Pfad neu aufgebaut wird.

.

Ein Rechtsklick auf die Methode in der Tabelle öffnet die zugehörige Unit im Editor. Der Editor zeigt die Methode am oberen Rand des Fensters.

Ein rotes **"R"** auf der linken Seite der Tabelle in der Mitte des Fensters zeigt an, dass die angezeigte Methode rekursiv aufgerufen wurde. Auch ein roter Methodenname in einem der Symbole hat diese Bedeutung.

A.3. Erzeugung exakter Ergebnisse

Wenn Sie ein paar Mal die Programmlaufzeiten messen, werden Sie sehen, dass die Messergebnisse von Messung zu Messung etwas unterschiedlich sind, ohne dass sich die Quellen geändert haben. Zwei Arten von Ergebnissen werden oft unterschiedlich sein: die Laufzeit einer Methode und der Prozentsatz ihrer Laufzeit vom gesamten Programm. Die Gründe dafür sind:

- Es gibt Ereignisse, die die Messung stören, z. B. Programme, die im Hintergrund laufen.
- Sie messen Methoden, die von Windows mehr oder weniger häufig aktiviert werden.
- Sie messen Operationen, die unterschiedlich oft bei jeder Messung gestartet wurden.
- Sie messen Methoden, die Dateitransfer ausführen: Mal werden die Daten auf die Festplatte, mal in den Disk-Cache geschrieben.

Jeder Profiler hat diese Probleme. Wegen der höchstmöglichen Granularität von ProDelphi64 (1 CPU-Zyklus), sehen Sie die Unterschiede auch. Bei anderen Profilern ist der Unterschied nicht sichtbar.

Um vergleichbare Messwerte zu erhalten, müssen Sie darauf achten, dass der Einfluss von Störungen gering gehalten wird. Hier einige Hinweise:

A.3.1. Häufige Ursachen für störende Einflüsse außerhalb des Programms

Einige Störer, die jeder kennt:

- aktivierte Bildschirmschoner,
- Windows-Energieverwaltung,
- Hintergrund Scheduler,
- Online-Virenschutz,
- automatische Erkennung einer CD.

Diese störenden Einflüsse sind leicht zu beseitigen.

A.3.2. Häufige Ursachen für störende Einflüsse innerhalb des Programms

Einige Störungen könnten Sie dem gemessenen Programm selbst haben, diese werden mitgemessen, wenn man alles misst, z.B. wegen der Autostart-Funktion von ProDelphi64:

- Default-Handler Prozedur (wird für fast jede Nachricht, die Ihr Programm erhält, aufgerufen),
- Methode zur Bearbeitung von Mausbewegungen (aufgerufen bei jeder Mausbewegung),
- Timer-Routine.

Die drei Einflüsse sind auch leicht zu beseitigen. Sie müssen diese Methoden von der Messung ausschließen. Ein anderer Weg ist, nicht die Autostart-Funktion von ProDelphi64 zu verwenden, sondern die Messung bei Beginn einer bestimmten Aktion zu starten. Wie Methoden ausgeschlossen werden, wird in Kapitel A.5 beschrieben, wie man nur definierte Aktionen misst, wird im Kapitel A.6 beschrieben.

A.3.3. Intel SpeedStep Technologie / Turbo Boost Mode

Diese beiden Technologien verändern dynamisch die CPU-Frequenz. Da die Änderungen nicht immer zum gleichen Zeitpunkt erfolgen, können verschiedene Messungen, auch wenn tatsächlich der gleiche Code durchlaufen wird, nicht miteinander verglichen werden, wenn Zeiteinheiten verwendet werden sollen. Wenn ein Vergleich von Zeiteinheiten gewünscht ist, sollten diese Features deaktiviert werden. Bei einigen PCs bietet das BIOS diese Möglichkeit. **Wird zum Vergleich eine History Datei verwendet, ist ein Vergleich möglich: Es werden die jeweils benötigten CPU-Zyklen verglichen.**

A.3.4. Häufige Ursache von störendem Einfluss ist der CPU-Cache

Der Einfluss des Cache ist nicht einfach auszuschließen. Der einzige Weg Messungen zu vergleichen ist, dafür zu sorgen, dass die Mess-Situation immer die gleiche ist. Dazu sollte man eine Aktion zweimal hintereinander starten und nur die zweite Messung für spätere Vergleiche zu benutzen (die zweite Messung als History speichern). Das geschieht am einfachsten über das Online-Bedien-Fenster, indem nach dem ersten und zweiten Ausführen die Ergebnisse gespeichert werden. Im Viewer wird später die zweite Messung als History gespeichert.

Ebenso möglich ist das über Benutzung der API-Aufrufe. Die Aufrufe zum Speichern müssen dann in den Code eingebaut werden.

A.3.5. Profiling auf mobilen Computern

Mobile Computer haben ein Problem: Sie verändern ihre CPU-Geschwindigkeit dynamisch. Wenn ein mobiler Computer am Stromnetz angeschlossen ist, hat er normalerweise die volle CPU-Geschwindigkeit, wenn er im Akkubetrieb arbeitet, ändert sich die CPU-Geschwindigkeit bei einigen Laptops dynamisch.

Dies hat nicht direkt Einfluss auf die Messung: ProDelphi64 misst ja CPU-Zyklen. Wenn CPU-Zyklen im Viewer angezeigt werden, ist die Messung korrekt. Wenn aber Zeiten angezeigt werden, könnte es sein, dass zu lange oder zu kurze Zeiten angezeigt werden. Das hängt von der CPU-Geschwindigkeit ab, die eingestellt war, als ProDelphi64 die CPU-Geschwindigkeit gemessen hat. Unterschiedliche Prozessoren verwenden unterschiedliche Algorithmen, um die Geschwindigkeit zu ändern. Der einzige Weg korrekte Ergebnisse zu erhalten, ist den Power-Safe-Modus auszuschalten.

A.3.6. Zusammenfassung

Wenn Sie die in A.3 erwähnten Störungen beseitigen und definierte Aktionen messen, sind die Unterschiede zwischen zwei Messungen sehr gering, meistens nur ein paar CPU-Zyklen. Größere Unterschiede zeigen sich nur beim Messen von Methoden mit Externspeicherzugriff. Ein guter Trick ist, die zweite Messung für den Vergleich mit späteren Optimierungen verwenden, speziell, wenn der Disk-Transfer ein Lesezugriff ist. Beim ersten Mal kommen die Daten in den Disk-Cache, bei der zweiten Messung werden die Daten aus dem Cache gelesen. Bei Schreibzugriffen funktioniert das nicht so gut.

A.4. Interaktive Optimierung

Interaktive Optimierung bedeutet, dass Sie immer nur eine Funktion optimieren, dann prüfen, ob es etwas gebracht hat oder nicht, dann den nächsten Schritt der Optimierung zu machen und so weiter. Eine Optimierung, die nur wenig bringt aber die Lesbarkeit oder die Wartbarkeit verschlechtert, sollte rückgängig gemacht werden.

Wichtig ist zu wissen, welche Methode es lohnt zu optimieren: Eine Methode, die 10 % Laufzeit verwendet, muss um 50 % optimiert werden, um die gesamte Laufzeit des Programms um 5 % verringern!

Es gibt verschiedene Möglichkeiten für den Vergleich der Ergebnisse mehrerer Messungen:

- Verwenden des Viewers und Drucken der jeweiligen Messergebnisse oder
- Benutzen der ProDelphi64 History - Funktion.

A.4.1. Die History-Funktion

Mit der History-Funktion des Viewers können Sie Ihre Messergebnisse mit denen eines vorhergehenden Laufs vergleichen. So können Sie sehen, ob eine Optimierung eine Erhöhung oder eine Verringerung der Laufzeit gebracht hat.

Nachdem eine Messung gemacht wurde, können Sie die im Viewer angezeigten Ergebnisse auf die Festplatte speichern. Sie können mehrere Messungen auf der Festplatte speichern.

Wenn Sie mehrere Messergebnisse als History gespeichert haben, können Sie eine der History-Dateien mit den Ergebnissen der aktuellen Messung vergleichen. Vor dem Laden der Messergebnisse wählen Sie die History-Datei mit den Vergleichsdaten und setzen die Option 'Mit History vergleichen'. Der Viewer wird dann in der Tabelle die Werte in Zellen mit unterschiedlichen Rahmenfarben darstellen, so haben Sie einen schnellen Überblick über alle Änderungen der Laufzeit: Rot bedeutet ‘Methode wurde langsamer’, grün bedeutet ‘Methode wurde schneller’, kein Rahmen heißt, dass keine wesentliche Änderung eingetreten ist.

Um die Rahmen farbig darzustellen, muss die Änderung der Laufzeit der Methoden wesentlich sein. Wesentlich bedeutet, sie muss sich so viel geändert haben, dass der Einfluss auf die Programmlaufzeit mindestens 1 % beträgt.

Um die Laufzeit einer Methode aus der gespeicherten History anzuzeigen, halten Sie die Strg-Taste und klicken mit der linken Maustaste auf die betreffende Methode.

A.4.2. Praktische Anwendung der History-Funktion

- Führen Sie eine Messung für die definierte Aktion, die Sie optimieren möchten, durch.
- Laden Sie die Ergebnisse in den Viewer.
- Klicken Sie auf den Button ‘Als History speichern’, um die angezeigten Ergebnisse in die History-Datei zu speichern.
- Optimieren Sie eine Methode, die einen großen Teil der Programmlaufzeit benötigt.
- Wiederholen Sie Ihre Messung.

- Laden Sie die neuen Ergebnisse.
  Wenn eine Methode deutlich schneller wurde, bekommt die Zelle mit der Laufzeit einen grünen Rahmen.
  Wenn Ihre Methode deutlich langsamer wurde, ist der Rahmen rot.
  Wenn es keinen signifikanten Unterschied gibt, gibt es keinen Rahmen.
- Wählen Sie eine Zelle in der Zeile, in der Ihre geänderte Methode angezeigt wird (Strg + linke Maustaste).
- Ein kleines Fenster öffnet sich. Es gibt die durchschnittliche Laufzeit der Methode an, so wie sie in der History-Datei gespeichert ist. Wenn "---" angezeigt wird, gibt es zu der Methode keine Daten in der History-Datei.

A.5. Messen von Teilen des Programms

A.5.1. Ausschluss von Teilen des Programms

Alle Windows-Programme sind ‘Nachrichten-Getrieben’. Also, wenn Sie eine Funktion definieren, die zum Beispiel Mausbewegungen bearbeitet, wird ProDelphi64 einen großen Prozentsatz der Laufzeit für diese Methode anzeigen, weil sie jedes Mal aufgerufen wird, wenn Sie mit der Maus über ein Fenster des Programms fahren. Vermutlich haben Sie kein Interesse an dieser Methode.

Eingangs wurde die Standardeinstellung von ProDelphi64 beschrieben: alle Methoden werden gemessen, die Messung startet mit dem Start des Programms (wenn die Option 'Bei Programmstart und per Online-Bedienung' gewählt ist).

Normalerweise möchten Sie nur bestimmte Aktionen des Programms messen oder vielleicht Funktionen ausschließen, die nicht optimiert werden können (z. B. weil sie sehr einfach sind).

Es gibt verschiedene Möglichkeiten, Teile des Programms auszuschließen:

1. Dateien in und unter den Delphi LIB- und SOURCE-Verzeichnissen werden immer ausgeschlossen.

2. Methoden, die die erste ‘BEGIN’-Anweisung und der letzte ‘END’-Anweisung in der gleichen Zeile haben, werden NICHT gemessen. **Das ist kein Bug! Das ist ein Feature!**

3. Ausschluss von Verzeichnissen
   Tragen Sie die Verzeichnisse in das Feld ‘Ausgeschlossene Verzeichn.’ des ProDelphi64-Hauptfensters ein. (Siehe auch A.13.5)

4. Ausschluss kompletter Units
   - Aktivieren Sie den Schreibschutz für die Units, die nicht gemessen werden sollen (sofern Sie nicht die Option ‘Schreibgesch. Dateien bearb.’ aktivieren, werden sie nicht instrumentiert) oder
   - Fügen Sie die folgende Anweisung vor der ersten Zeile der Unit ein:
     //PROFILE-NO

5. Ausschluss einer DLL aber Messung des aufrufenden Programms
   Kompilieren Sie die DLL ohne die Compiler Bedingung PROFILE und das Programm mit dieser Bedingung.

6. Ausschluss des gesamten Programms, aber Messung der aufgerufenen DLL
   Kompilieren Sie das Programm ohne die Compiler Bedingung PROFILE und die DLL mit dieser Definition.

7. Ausschluss von Funktionen
   Vor der Instrumentierung fügen Sie Anweisungen vor und nach den Methoden, die von der Instrumentierung ausgeschlossen sein sollen, ein:

```
//PROFILE-NO            | Diese Anweisung wird beim Clean nicht gelöscht
PROCEDURE A;
BEGIN
  :
END;
PROCEDURE B;
BEGIN
  :
END;
//PROFILE-YES           | Diese Anweisung wird beim Clean nicht gelöscht
```

8. Automatischer Ausschluss

   Sie können Methoden automatisch ausschließen, indem Sie die Option 'Deact. methods with runtime < 1 µs' aktivieren.
   Wenn Sie diese Option aktivieren, wird die Laufzeit einer Methode, die
   - mindestens 10 Mal aufgerufen wurde,
   - keine andere gemessene Methode aufgerufen hat und
   - während des Messzeitraums im Durchschnitt weniger als 1 µs benötigt hat

ab dem nächsten Programmstart nicht mehr explizit erfasst.

Zu diesem Zweck wird bei Beendigung des Programms eine Datei angelegt. Sie enthält alle Methoden, deren Laufzeiten nicht explizit erfasst werden sollen. Beim nächsten Start Ihres Programms wird diese Datei gelesen. Die Laufzeiten der ermittelten Methoden werden zu den Laufzeiten ihrer aufrufenden Methoden addiert. Es kann sein, dass nach der nächsten Ausführung des Programms wieder andere Methoden mit einer Laufzeit von weniger als 1 µs gefunden werden. Auch diese werden dann nicht mehr erfasst.

Optional werden bei der nächsten Instrumentierung keine Messaufrufe in diese Methoden eingefügt. Ihre Laufzeit ist dann automatisch in der Laufzeit ihrer aufrufenden Methode enthalten. Dies führt dazu, dass ein Programm mehr als 65500 Methoden enthalten kann.

Die Methoden, deren Laufzeiten nicht explizit erfasst werden sollen, werden in der Datei 'Programmname.swo' gespeichert.

Achtung! Wenn der Ausschluss dauerhaft sein soll, fügen Sie eine //PROFILE-NO und eine //PROFILE-YES Anweisung um eine solche Methode herum in den Quellcode ein.

A.5.2. Dynamische Aktivierung der Messung

Dies ist der beste Weg der Profilierung. Normalerweise optimiert man die Funktion eines Programms, die zu lange dauert. Wenn z. B. ein Programm viele Messwerte verarbeiten und nur sporadisch Meldungen verarbeiten und drucken soll, macht es Sinn, die Messwertverarbeitung zu optimieren und nicht die Meldungsverarbeitung.

In diesem Beispiel wäre es schön, jedes Mal, wenn ein Messwert verarbeitet wird, die Laufzeitmessung einzuschalten und nach Ende der Verarbeitung wieder abzuschalten. Der Vorteil ist, dass die Zahl der Zeilen im Viewer reduziert wird, ein anderer ist, dass es einfacher ist, zu sehen, welche Funktion zu optimieren sinnvoll ist.

Es gibt drei Möglichkeiten für die dynamische Aktivierung der Messung in ProDelphi64 (Methode 1 und 2 können gleichzeitig verwendet werden):

1. Durch Dialog

   Wählen Sie im Hauptfenster von ProDelphi64 unter der Option 'Aktivierung der Messung': 'Bei Aufruf ausgewählter Methoden'.

   Klicken Sie dann auf den Button ‘Instrument(ieren)./Aktiv(ierungs)meth(oden) ausw(ählen)’. Nach der Instrumentierung können Sie bis zu 16. Methoden wählen, die die Messung starten sollen. Wenn Sie das Programm bereits instrumentiert haben, klicken Sie den Button ‘Auswahl Aktivierungsmethoden’ und wählen Sie die Methoden aus. Jetzt starten Sie Ihr Programm. Immer wenn eine der Aktivierungsmethoden aufgerufen wird, wird die Messung eingeschaltet, wenn die Methode verlassen wird, wird die Messung wieder deaktiviert.

2. Durch Einsetzen spezieller Kommentare in den Quellcode.

   Einfügen des Kommentars //PROFILE-ACTIVATE in den Quellcode bewirkt, dass die nächste Methode nach diesem Kommentar automatisch die Messung startet. Auch hier müssen Sie die Option 'Bei Aufruf ausgewählter Methoden' auswählen.

3. Durch die Verwendung von API-Aufrufen.

   Diese Methode wird im nächsten Kapitel beschrieben. Es war die einzige Möglichkeit, die frühere Versionen von ProDelphi als 8.0 hatten. Grundsätzlich kann auf diese Weise weiter verfahren werden, aber es ist nicht sehr komfortabel. Mithilfe dieser dritten Methode müssen Sie immer zwei Anrufe einfügen, eine zur Aktivierung und eine zur Deaktivierung der Messung (siehe Kapitel A.6).

A.5.3. Finden von Punkten für die dynamische Aktivierung

Wenn Sie eine Anwendung, die Sie nicht selbst implementiert haben, profilieren müssen, ist es nicht einfach herauszufinden, wo eine Aktion startet. Die meiste Zeit gibt es eine Menge von Ereignissen und Windows-Meldungen, aber welche Methoden reagieren auf diese Ereignisse oder Nachrichten ?

Um es leichter zu machen dies herauszufinden, werden durch Ereignisse / Nachrichten gestarteten Methoden in eine Liste eingetragen. Führen Sie einfach eine Messung durch, bei der alle Methoden gemessen werden (Messung beginnt automatisch mit dem Start der Anwendung). Nach Durchführung der zu optimierenden Aktion beenden Sie die

Anwendung, starten den Profiler und zeigen die Ergebnisse an. Unter dem letzten Reiter des Viewers werden alle Methoden aufgeführt, die nicht durch andere gemessene Methoden aufgerufen wurden. Das bedeutet, dass sie durch die Ereignisse wie Mausklicks, Windows-Meldungen usw. gestartet wurden. Ausgehend von diesen Funktionen und im Zusammenhang mit dem Aufruf-Graphen sollte es einfach sein, herauszufinden, welches die Aktivierungspunkte für eine zu messende Aktion sind.

A.5.4. Messung bestimmter Teile einer Methode

Für den Fall von sehr großen Methoden ist es vielleicht interessant zu wissen, welcher *Teil* davon die meiste Laufzeit verbraucht. Ein Weg dies herauszufinden ist, Teile der Methode in lokale Prozeduren zu legen.

Eine weitere Idee wäre, ProDelphi64 würde automatisch einzelne Blöcke (z. B. eine WHILE-Schleife, eine IF-Anweisung) und nicht die ganze Methode messen. Diese Lösung würde viel Mess-Overhead kosten und die Messung würde bei zeitkritischen Anwendungen nicht mehr funktionieren.

Für den zweiten Fall hat ProDelphi64 die Möglichkeit, manuell zu definierende Blöcke zu messen.

Mit dem Einbau von zwei einfachen Anweisungen kann ein Block zur Messung festgelegt werden. Diese Anweisungen werden als Kommentare eingebaut und können im Code verbleiben und werden deshalb beim Entfernen der Instrumentierung NICHT entfernt.

Fügen Sie einfach diese Zeile vor den zu messenden Block ein:

```
//PROFILE-BEGIN: Kommentar
```

und diese dahinter:

```
//PROFILE-END
```

Danach ist eine Re-Instrumentierung mit der Option 'Lokale Prozeduren' notwendig!

Instrumentierung der Quellen nach dieser Änderung bewirkt, dass ProDelphi64 Messanweisungen direkt nach den Kommentaren einfügt. Die Laufzeit in diesen so definierten Blöcken wird im Viewer durch die Suche nach
'Methoden-Name (Kommentar)'
gefunden.

Dieses Feature darf nur so eingesetzt werden, dass die Blockstruktur des Programms unverändert bleibt.

Die Zeit, die in diesem Teil gemessen wird, ist nicht in der Laufzeit der Methode enthalten, sondern in ihrer Kind-zeit.

Beispiel:

```
PROZEDUR DoSomething;
BEGIN
        Teil a, Anweisungen mit Laufzeit 5 ms
        Teil b, Anweisungen mit Laufzeit 10 ms
        Teil c, Anweisungen mit Laufzeit 3 ms
END;
```

Die gesamte Laufzeit, die durch den Viewer angezeigt wird, beträgt 18 ms (angezeigt in der Zeile für die Prozedur DoSomething) sein.

Das gleiche Beispiel mit Teil-b getrennt gemessen:

```
PROZEDUR DoSomething;
BEGIN
        Teil a, Anweisungen mit Laufzeit 5 ms
//PROFILE-BEGIN: Teil-b
        Teil b, Anweisungen mit Laufzeit 10 ms
//PROFILE-END
        Teil c, Anweisungen mit Laufzeit 3 ms
END;
```

In diesem Fall wird die Laufzeit der Prozedur ‘DoSomething’ 8 ms betragen (angezeigt in der Zeile für die Prozedur DoSomething), Laufzeit inklusive Kind Zeit würde 18 ms betragen.
In der Zeile für Prozedur DoSomething-Teil-b würde dann 10 ms angezeigt werden.

Es könnte sein, dass die Ergebnisse nicht genau die gleichen sind, da der Prozessor-Cache in einer anderen Art und Weise verwendet wird. Insbesondere haben vor allem Prozessoren mit einem kleinen Cache das Problem, dass nicht die ganze Methode in den Cache passt, sodass zusätzliche Waitstates auftreten.

Bemerkung:

Es ist möglich, mehr als einen Block einer Methode zu messen oder Messblöcke zu verschachteln. Verschachtlung ist aber vielleicht keine gute Idee sein, weil die Messergebnisse leicht falsch interpretiert werden.

Beispiel für die Verschachtelung:

```
PROZEDUR DoSomething;
BEGIN
  //PROFILE-BEGIN: Teil-ab
        Teil a von Anweisungen mit 5 ms
        //PROFILE-BEGIN: Teil-b
          Teil b von Anweisungen mit 10 ms
        //PROFILE-END
  //PROFILE-END
  Teil c der Anweisungen mit 3 ms
END;
```

In diesem Beispiel wird die Laufzeit für Teil ‘b’ separat angezeigt und ist als Kindzeit sowohl in Teil ‘a’ und auch in Do-Something enthalten. Die Laufzeit der Prozedur ‘DoSomething’ wird 3 ms betragen.

A.6. Programmier API

A.6.1. Messung definierter Aktionen des Programms durch Aktivierung und Deaktivierung

Eine gute Möglichkeit unterschiedliche Ergebnis-Dateien vergleichbar zu machen ist, nur die Aktionen des Programms, die Sie optimieren möchten, zu messen. In diesem Fall wählen Sie für 'Aktivierung der Messung' die Option 'Durch API-Aufruf und Online-Bedien-Fenster'. Fügen Sie Aktivierungsaufrufe an die entsprechenden Stellen im Sourcecode ein und instrumentieren Sie das Programm.

Beispiel 1 (für VCL-Anwendungen *):

Sie wollen nur wissen, wie viel Zeit eine Sortierprozedur verbraucht, wie viel Zeit alle aufgerufenen Kind Prozeduren verbrauchen. Sie sind nicht an anderen Prozeduren interessiert. Die Sortierung wird durch eine Prozedur namens ButtonClick gestartet.

```
PROZEDUR TForm1.ButtonClick;
BEGIN
{$IFDEF PROFILE} ProfInt.ProfStop; Try; ProfEnter; end; {$ENDIF}
   SortAll; // die Prozedur, von der Sie die Laufzeit wissen wollen
{$IFDEF PROFILE} finally; ProfInt.ProfExit; end; {$ENDIF}
END;
```

Sie können den Code auf drei verschiedene Arten ändern:

{ 1. Möglichkeit: }

```
PROZEDUR TForm1.ButtonClick;
BEGIN
{$IFDEF PROFILE} Try; ProfInt.ProfEnter; end; {$ENDIF}
        {$IFDEF        PROFILE} Try; ProfInt.ProfActivate; {$ENDIF}
        SortAll;  //die Prozedur, von der Sie die Laufzeit wissen wollen
        {$IFDEF        PROFILE} finally; ProfInt.ProfDeactivate end; {$ENDIF}
{$IFDEF PROFILE} finally; ProfInt.ProfExit; end; {$ENDIF}
END;
```

{ 2. Möglichkeit: }

```
PROZEDUR TForm1.ButtonClick;
BEGIN
{$IFDEF PROFILE} Try; ProfInt.ProfActivate; {$ENDIF}
        SortAll;  //die Prozedur, von der Sie die Laufzeit wissen wollen
{$IFDEF PROFILE} finally; ProfInt.ProfDeactivate; end; {$ENDIF}
END;
```

{ 3. Möglichkeit: }

```
//PROFILE-NO          //Instrumentierung ausschalten
PROZEDUR TForm1.ButtonClick;
BEGIN
{$IFDEF        PROFILE} Try; ProfInt.ProfActivate; {$ENDIF}
        SortAll;  //die Prozedur, von der Sie die Laufzeit wissen wollen
{$IFDEF        PROFILE} finally; ProfInt.ProfDeactivate; end; {$ENDIF}
END;
//PROFILE-YES         //Instrumentierung einschalten
```

Sie sollten Möglichkeit 1 oder 3 verwenden, da eine neue Instrumentierung der Code von Möglichkeit 2 geändert wird. Möglichkeit 3 ist die bessere, sie hat weniger Messungs-Overhead.

Seien Sie sicher, dass Sie mehr als ein Blank zwischen $IFDEF und PROFILE einfügen, da die Anweisungen sonst das nächste Mal, wenn der Quellcode instrumentiert wird, durch ProDelphi64 gelöscht wird. Alternativ können Sie die Anweisungen auch mit Kleinbuchstaben schreiben. Dies verhindert ebenfalls die Löschung.

**** Für FMX-Anwendungen statt ProfInt ..... ProfIntf ..... verwenden***

Beispiel 2 (für VCL-Anwendungen *):

Sie wollen die Zeitmessung durch eine Prozedur namens Button1 aktivieren und durch eine Prozedur namens Button2 deaktivieren. Verwenden Sie folgenden Aufbau:

```
//PROFILE-NO
PROZEDUR TForm1.Button1;
BEGIN
{$IFDEF PROFILE} ProfInt.ProfActivate; {$ENDIF}
END;

PROZEDUR TForm1.Button2;
BEGIN
{$IFDEF PROFILE} ProfInt.ProfDeactivate; {$ENDIF}
END;
//PROFILE-YES
```

Deaktivierung schaltet die Messung völlig aus. Das bedeutet, dass keine Methode bis zur erneuten Aktivierung gemessen wird.

**** Für FMX-Anwendungen statt ProfInt ..... ProfIntf .....*** **verwenden**

A.6.2. Verhinderung der Messung inaktiver Zeiten

Einige Windows-API-Funktionen und Delphi Funktionen unterbrechen die aufrufende Methode und setzen das Programm in einen Ruhezustand. Ein bekanntes Beispiel ist der Windows-Aufruf MessageBox. Dieser Aufruf kehrt nach dem Klicken auf einen Button zurück. Zwischen Aufruf und Rückkehr zur aufrufenden Methode verbraucht das Programm CPU-Zyklen. In einem solchen Fall wäre es gut, nicht diese Totzeit zu messen.

Eine Menge von Windows-API-Aufrufen und einige Delphi-Anrufe werden automatisch durch die Unit 'Profint.pas "ersetzt. Für das oben genannte Beispiel MessageBox, gibt es eine Redefinition. Es unterbricht automatisch die Zählung der CPU-Zyklen für die aufrufende Methode und reaktiviert sie nach Beenden der Windows-Funktion.

Wenn andere Methoden aufgerufen werden, während das Programm auf eine Reaktion des Benutzers wartet, werden sie in der Regel gemessen, z.B. wenn ein WM_TIMER Ereignis auftritt und dafür ein Handler definiert ist.

Um o.g. Unterbrechung der Messung zu ermöglichen, gibt es die ProDelphi64-API-Aufrufe StopCounting und ContinueCounting. In Kapitel A.10 finden Sie die Liste der Aufrufe, die in der Einheit ‘Profint.pas 'neu definiert werden. Sie verwenden automatisch diese Funktionen, bevor die Original-Windows- oder Delphi Aufrufe benutzt werden. Einige Funktionen werden vom Profiler im Sourcecode ersetzt (z. B. Application.HandleMessage).

Einige Funktionen können nicht durch ‘Profint.pas’ ersetzt werden, speziell Objekt-Methoden. Wenn Sie solche Methoden verwenden und ihre Wartezeiten nicht messen wollen, können Sie diese Aufrufe von der Messung ausschließen, indem Sie sie in folgende Zeilen einschließen:

```
{$IFDEF PROFILE} ProfInt.StopCounting; {$ENDIF}

    Object.IdleModeSettingMethod; // Diese Methode sollte nicht INSTRUMENTIERT WERDEN!
                                  // Siehe: / / PROFILE-NO

{$IFDEF PROFILE} ProfInt.ContinueCounting; {$ENDIF}
```

Wichtig:

Verwenden Sie mehr als ein Leerzeichen zwischen $IFDEF und PROFILE, da sonst die Zeilen bei der nächsten Instrumentierung oder durch die Reinigung der Quellen entfernt werden. Alternativ können Sie auch Kleinbuchstaben verwenden, dies schützt ebenfalls vor Entfernung.

**** Für FMX-Anwendungen statt ProfInt ..... ProfIntf .....*** **verwenden**

A.6.3. Programmiertes Speichern von Messergebnissen

Bei Start des Programms werden die Messergebnisse in der Ergebnisdatei gelöscht. Bei Ende des Programms werden die Messergebnisse angehängt. Wenn Sie weitere Informationen benötigen, können Sie Aufrufe in Ihre Quellen einbauen, um Ergebnisse zu speichern, wann Sie möchten.

Fügen Sie einfach die Anweisung:

{$IFDEF PROFILE} ProfInt.ProfAppendResults(false); {$ENDIF}

in den Code ein. In diesem Fall werden die aktuellen Zählerstände ans Ende der Datei angehängt und dann alle Zähler zurückgesetzt.

Normalerweise ist die Überschrift für alle Messungen 'At Finishing-Anwendung'.

In diesem Beispiel möchten Sie vielleicht eine andere Überschrift verwenden. Wenn ja, können Sie den Text für die Überschrift durch Einsetzen von

{$IFDEF PROFILE} ProfInt.ProfSetComment ('Ihr spezieller Kommentar'); {$ ENDIF}

im Quellcode ändern.

Ein anderer Weg, Zwischenergebnisse zu produzieren, ist die Nutzung des *Online-Bedien-Fensters*. Jedes Mal, wenn Sie den 'Append'-Button anklicken, werden die aktuellen Messwerte an die Ergebnis-Datei angehängt und alle Ergebnis-Zähler auf Null gesetzt (siehe auch A.8).

Wichtig:

Verwenden Sie mehr als ein Leerzeichen zwischen $IFDEF und PROFILE, da die Zeilen sonst mit der nächsten Instrumentierung oder durch die Reinigung der Quellen entfernt werden. Alternativ können Sie auch Kleinbuchstaben verwenden, dies schützt ebenso vor Löschung.

Noch wichtiger:

Dieser Aufruf sollte nur in einer nicht gemessenen Methode verwendet werden und nur dann, wenn alle gemessenen Funktionen verlassen sind. Für Methoden, die noch nicht verlassen worden sind, wird die Laufzeit ihres aktuellen Aufrufs Null.

**** Für FMX-Anwendungen statt ProfInt ..... ProfIntf ..... verwenden***

A.7. Optionen für die Instrumentierung und Messung

Profiling-Optionen werden in vier Gruppen unterteilt:

- Instrumentierungs-Optionen: Wie und was man instrumentiert.
- Laufzeitmessungs Optionen: Wie zu messen ist und was mit den Ergebnissen zu tun ist.
- Aktivierung der Messung: Wo oder wann die Messung zu starten ist.
- Allgemeine Optionen: Delphi Version / Datei Datum.

A.7.1. Optionen für die Code-Instrumentierung:

Ändern dieser Optionen werden erst nach der nächsten Instrumentierung wirksam!

Initialisierung und Finalisierung

Normalerweise werden Initialisierungs- und Finalisierungsteile der Units nicht gemessen. Falls Sie dies wollen und die Schlüsselwörter INITIALIZATION und FINALIZATION in Ihren Units verwenden, aktivieren Sie die entsprechende Option.

Erlaube Opti(mierung)

Es wird dringend geraten, die zu messende Anwendung ohne Optimierung zu kompilieren ! ProDelphi64 prüft die Projektdatei auf die Option ‘Optimierung’. Wenn die Optimierung aktiviert ist, fügt ProDelphi64 eine $O- Anweisung in die Quellen ein. Die Anweisung ist nur aktiv, wenn das Compiler-Symbol ‘PROFILE’ definiert ist.

Wenn es absolut notwendig ist, dass die gemessene Anwendung mit Optimierung kompiliert werden muss (z.B. wegen schlechter 3rd-Party-Komponenten) und die $O- Anweisung nicht in die Quellen eingefügt werden soll, muss diese Option aktiviert werden.

Beachten Sie, dass die Messergebnisse weniger genau sind, wenn die gemessene Anwendung mit Optimierung kompiliert wurde!

Packages

Wenn diese Option aktiviert ist, werden auch die DPK-Dateien in dem Verzeichnis, in dem die DPR-Datei gespeichert ist, verarbeitet. Für weitere Informationen siehe Kapitel A.9.

Lokale Prozeduren

Normalerweise werden lokale Prozeduren nicht instrumentiert und getrennt gemessen, außer diese Option ist aktiviert.

Instrumentieren im Library Pfad (nur Professional-Version)

Normalerweise werden nur die Dateien, die zu einem Projekt gehören, instrumentiert. Diese Dateien sind alle Dateien im Unit-Suchpfad des aktuellen Projekts. Dateien, die zu Komponenten gehören und vom Linker zum Programm dazu gebunden werden, werden nicht instrumentiert. Normalerweise sind sie separat schon einmal optimiert worden und müssen dann nicht wieder mit jedem Projekt erneut optimiert werden. Deshalb sind sie in der Regel ausgeschlossen.

Diese Option eröffnet die Möglichkeit, auch die Dateien im Library-Pfad zu messen. Um dies zu tun, ist Vorsicht angebracht. Wenn Sie diese Dateien inkludieren, werden die Quellen instrumentiert. Wenn Sie nach der Optimierung zu einem anderen Projekt wechseln, das sie aber gar nicht optimieren wollen, sind die Dateien noch instrumentiert und werden so in das andere Projekt rein-kompiliert. Das bedeutet, dass Sie die Laufzeit der Library-Units in allen anderen Projekten messen. Diese Messung verlangsamt dann alle anderen Projekte, die diese Units verwenden. Um dies zu verhindern, sollten Sie die Instrumentierung der Quellen vor dem Wechsel zu einem anderen (nicht instrumentierten) Projekt entfernen.

Auch wenn Sie ein weiteres Projekt optimieren wollen, müssen Sie vorsichtig sein. Solange die Dateien im Library-Pfad noch instrumentiert sind, müssen Sie diese Option in allen Projekten aktivieren.

Wenn Sie Verzeichnisse aus dem Library-Pfad ausschließen, müssen diese in allen Projekten ausgeschlossen werden, sonst kann es passieren, dass in den Ergebnissen undefinierte Prozeduren vorkommen können.

Bearbeitung schreibgeschützter Dateien

Aktivierung dieser Option bedeutet, dass der Schreibschutz Ihrer schreibgeschützten Dateien ignoriert wird und die Dateien instrumentiert werden. Ohne diese Option werden schreibgeschützte Dateien nicht bearbeitet.

Programm + DLL’s / Mehr(ere) DLL’s

Aktivierung dieser Option bedeutet, dass Sie entweder eine DLL oder ein Programm + benutzte DLL's messen wollen. Siehe Kapitel A.9 für weitere Einzelheiten.

A.7.2. Optionen für die Laufzeitmessung

Ändern dieser Optionen erfordert KEINE neue Instrumentierung.

Inherited Aufrufe bei Aufrufer messen

Diese Option wirkt nur bei Klassen-Methoden.

Normalerweise werden alle Methoden getrennt gemessen. Verwenden Sie diese Option, wenn Sie wollen, dass die Laufzeit einer Methode, die von einer Methode gleichen Namens aus einer oberen Klasse aufgerufen wird (Vererbung), zu der Zeit des Aufrufers der Methode addiert wird.

Deaktivieren von Funktionen, die weniger als 1 µs verbrauchen

Jedes Mal, wenn die Messergebnisse gespeichert werden, werden die Methoden zur Deaktivierung vorgemerkt, die
- mindestens zehnmal aufgerufen wurden
- keine Kind-Methoden aufriefen, außer bereits deaktivierte und
- weniger als 1 µs verbrauchen.

Die Namen der zu deaktivierenden Funktionen werden in der Datei 'ProgramName.SWO' für den nächsten Lauf gespeichert.
Die Laufzeit der dann deaktivierten Funktionen wird der aufrufenden Funktion hinzugefügt.

und in Zukunft nicht mehr instrumentieren (nur Professional-Version) *(nur wenn vorherige Option ausgewählt)*

Bei der nächsten Instrumentierung diese Methoden nicht mehr instrumentieren (der Instrumentierungs-Code wird für die deaktivierten Methoden gelöscht). Der Zweck dieser Funktion ist es, den durch die Instrumentierung auftretenden Overhead bei der Laufzeitmessung zu reduzieren.

Daten für Aufrufstruktur generieren (Caller / Called graph)

Wenn Sie diese Option aktivieren, werden Daten abgespeichert, die zur Anzeige der Aufrufstruktur benötigt werden. Natürlich wird mit dieser Funktion mehr Mess-Overhead erzeugt, als ohne diese Funktion.

Online-Bedien-Fenster immer oben

Normalerweise wird das Online-Bedien-Fenster als sekundäres Fenster angezeigt. Dies bedeutet, dass es vom Hauptfenster verborgen ist. Mit dieser Option können Sie erzwingen, dass es über dem Hauptfenster liegt.

Kein Online-Bedien Fenster

Das Online-Bedien-Fenster wird nicht angezeigt. Dadurch können keine Zwischenergebnisse gespeichert werden.

Kopie der Ergebnisse speichern

Die Messergebnis-Dateien haben die Namen

'Anwendungsname.xxx'

und enthalten die letzten Messungen. Wenn z.B. mehr als ein Entwickler die gleiche Anwendung bearbeitet, weiß niemand, wessen Ergebnisse gerade gespeichert sind. Wenn diese Option aktiviert ist, werden die Messergebnisse zusätzlich unter folgendem Namen gespeichert:

"Anwendungsname-Username-Datum-Zeit.xxx '.

Auch können so die verschiedenen Schritte der Optimierung dokumentiert werden.

Testant enthält Threads

Wenn diese Option aktiviert ist, wird die Messung von Threads ermöglicht. Es ist nicht sinnvoll, diese Option zu aktivieren, wenn Ihr Programm keine Threads enthält, das Programm läuft nur langsamer. Aber es ist absolut notwendig, diese Option zu aktivieren, wenn Sie Threads verwenden, sonst sind die Ergebnisse der Messung völlig falsch.

Nur Haupt-Thread messen

Wenn diese Option aktiviert ist, werden nur die gemessenen Zeiten des Haupt-Threads gemessen. Zeiten von Kindthreads werden ignoriert.

Min. / Max. Laufzeiten speichern (Professional-Version)

Wenn diese Option aktiviert ist, ermittelt ProDelphi64 zusätzlich minimale und maximale Laufzeiten der Methoden. Normalerweise werden nur durchschnittliche Zeiten gespeichert. Minimum und Maximum können später bei Bedarf durch den integrierten Viewer angezeigt werden. Natürlich wird mit dieser Funktion mehr Overhead gebraucht als nur bei Messung von durchschnittlichen Zeiten.

A.7.3. Optionen für die Aktivierung der Messung

Ändern dieser Optionen erfordert KEINE neue Instrumentierung.

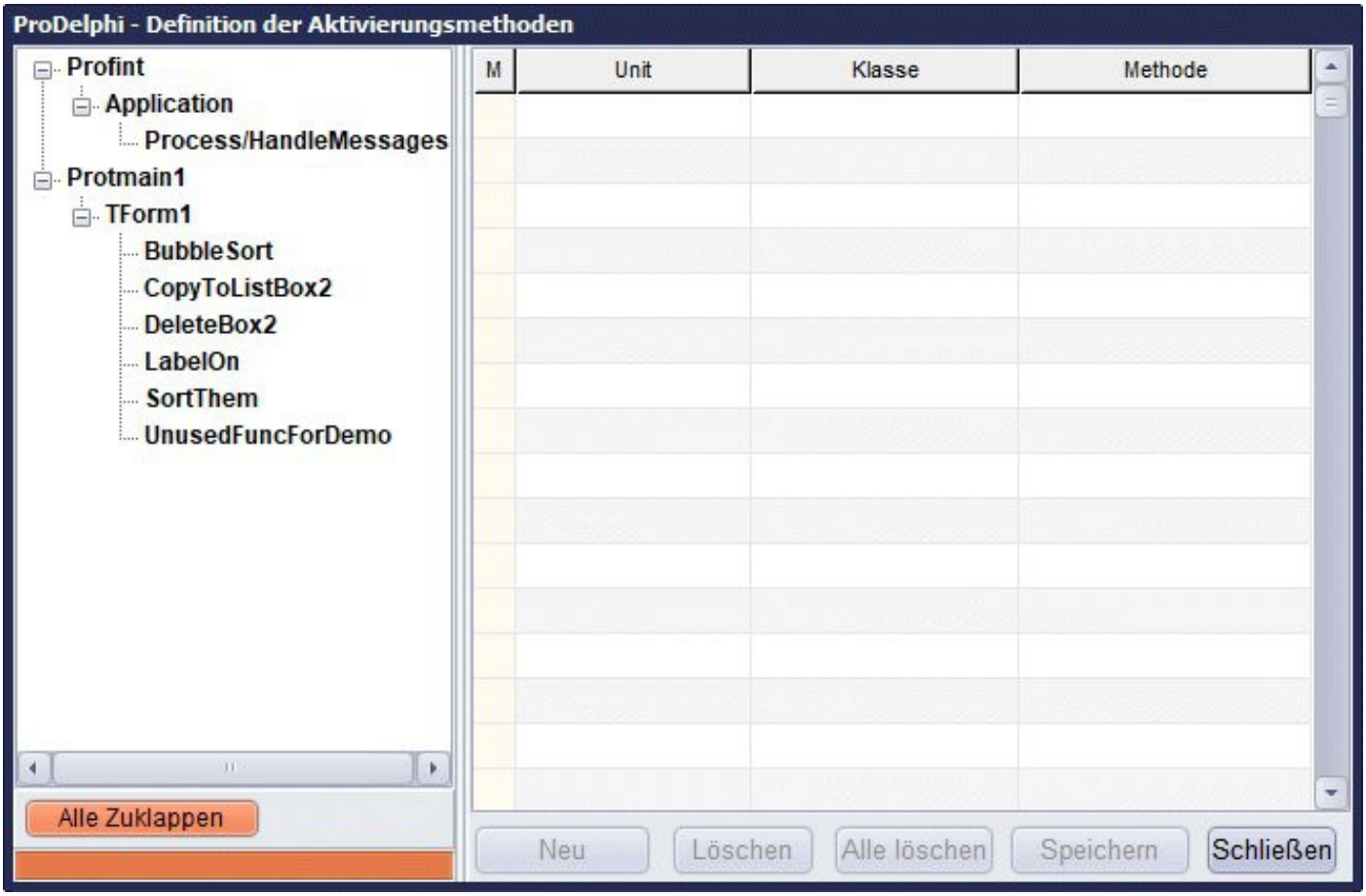
Bei Programmstart und per Online-Bedienung (Voreinstellung)

Wenn diese Option aktiviert ist, wird die Zeitmessung gestartet, sobald das Programm gestartet wird. In diesem Fall ist der 'Start'-Button im Online-Bedien-Fenster deaktiviert und der ‘Stop’-Button aktiviert. Wenn die Option nicht aktiviert ist, ist der 'Start'-Button aktiviert und der' Stop'-Button deaktiviert.

Bei Aufruf ausgewählter Methoden

Sie werden aufgefordert Aktivierungs-Methoden zu definieren und tun dies, außer Sie haben schon //PROFILE-ACTIVATE Aufrufe in den Quellcode eingefügt (siehe auch Kapitel A.5.2). Wenn Sie diese Option verwenden, sollten Sie nicht das Online-Bedien-Fenster benutzen.

Beispiel:



Durch API-Aufruf und Online-Bedienung

(Siehe Kapitel A.6 und A.8 für Details)

A.7.4. Allgemeine Optionen

Delphi-Version

Sie sollten die Delphi-Version auswählen, mit der Sie das Programm kompilieren wollen. Dies gewährleistet, dass Pro-Delphi64 die richtigen Compiler-Schalter berücksichtigt. Wenn ProDelphi64 über das Tools-Menü gestartet wird, wird die Delphi-Version automatisch richtig eingestellt.

Dateidatum

Die Option ‘Dateidatum und Attribute nicht ändern’ ist in der Professional-Version verfügbar. Aktivierung dieser Option bewirkt eine Erhöhung vom Dateidatum / Zeit um 2 Sekunden bei der Instrumentierung, genug, dass Delphi realisiert, dass sich eine Datei geändert hat. Deaktivieren bedeutet, dass aktuelles Datum und Zeit verwendet werden.

Bei der Entfernung der Instrumentierung werden Datum / Uhrzeit um 2 Sek. für jeden Instrumentierungs-Prozess erniedrigt. Dadurch ist wieder die ursprüngliche Zeit eingestellt. Das ermöglicht, dass die Datei das gleiche Datum zwischen Check-out und Check-In in einem Quellcodeverwaltungssystem hat.

Da einige Quellcodeverwaltungssysteme auch das Archiv-Attribut überprüfen, behält ProDelphi64 den Status von vor der Instrumentierung bei.

Build-Konfiguration

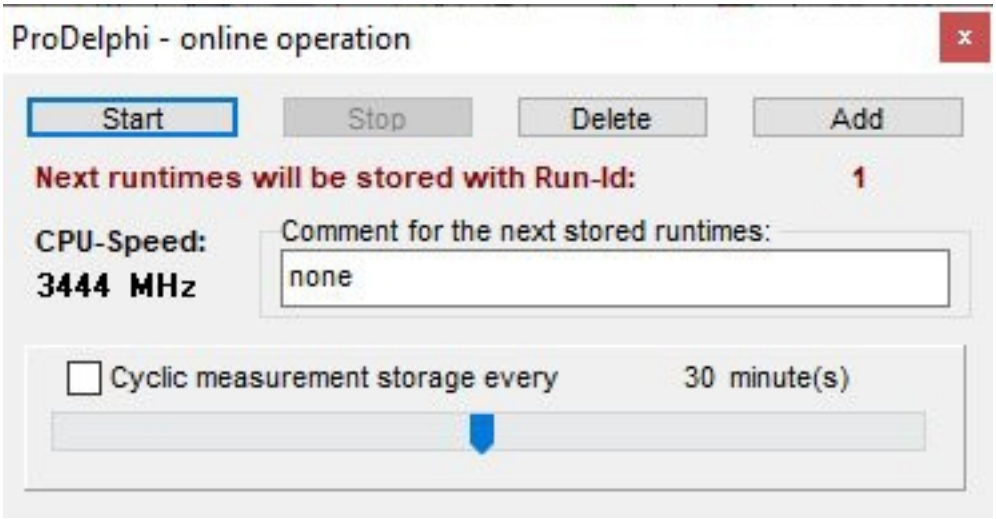
Stellen Sie die Build-Konfiguration ein, die Sie mit der nächsten Kompilierung verwenden werden. Wenn ProDelphi64 über das Tools-Menü gestartet wird, wird die Build-Konfiguration automatisch anhand der aktuellen Konfiguration im Projektfile festgelegt.

Klassenbibliothek

Es kann zwischen VCL und FMX gewählt werden.

A.8. Online-Bedien-Fenster

Mit dem Online-Bedien Fenster



kann die Zeitmessung gestartet oder gestoppt werden. Dies ermöglicht es Ihnen, nur bestimmte Aktivitäten des Programms zu messen. Der ‘Start'-Button aktiviert die Messung, der ‘Stop'-Button deaktiviert sie. Mit dem ‘Delete'-Button werden alle Zähler auf 0 gesetzt. Der ‘Add’-Button speichert die aktuellen Zählerstände in der Ergebnisdatei und setzt die Zähler auf 0.

Sie können den Text bearbeiten, der als Überschrift für die Ergebnisse verwendet werden soll. Für den Betrachter wird bei jeder Speicherung der Ergebnisse die ‘Run-Id' erhöht und Sie können mit dem Viewer zwischen verschiedenen Runs wechseln (Run = Speicherung).

Der Standardwert für die Überschrift für Zwischenergebnisse ist: 'None'.

Auch eine automatische und zyklische Speicherung der Messergebnisse kann durchgeführt werden. Verwenden Sie den Schieberegler, um den Zyklus zwischen 1 und 60 Minuten einzustellen. Danach aktivieren Sie das Kontrollkästchen für zyklische Messwertspeicherung. Dann wird der Schieberegler unsichtbar und erst wieder sichtbar, wenn die zyklische Speicherung deaktiviert wird. Die Ergebnisse erhalten automatisch Datum und Uhrzeit als Überschrift. Im Viewer können Sie durch die Ergebnisse mit den Buttons '◀' und '▶' blättern.

Das Online-Bedien Fenster ist für Konsolen-Anwendungen nicht verfügbar!

A.9. Dynamic Link Libraries (DLL) und Packages

A.9.1. DLL's

DLL's können in gleicher Weise wie Programme gemessen werden

Einige Vorsichtsmaßnahmen sind nötig, um Probleme zu vermeiden:

DLL's können nur mit einem aufrufenden Programm verwendet werden, egal ob instrumentiert oder nicht, egal ob Sie die Messergebnisse für den Code im Programm brauchen oder nicht. Die DLL erwartet die Messoptionen im Verzeichnis der EXE des aufrufenden Programms. Auch werden die Messergebnisse in diesem Verzeichnis gespeichert.

Um ein problemloses Messen, das in allen Kombinationen funktioniert (nur EXE, nur DLL, EXE + DLL, EXE + mehrere DLL's) mit einem Minimum an Aufwand zu gewährleisten, sollte wie im Folgenden beschrieben vorgegangen werden:

1. Machen Sie den Units-Suchpfad aller betroffenen Projekte (EXE + DLL's) identisch.
2. Die Verzeichnisse zum Speichern von EXE- und DLL-Datei müssen identisch sein.
3. Die Verzeichnisse der DPR-Dateien müssen identisch sein.
4. Wenn die DPR-Dateien eine Datei 'uses', die sich nicht in einem der im Suchpfad genannten Verzeichnisse befindet, muss sie im Verzeichnis der DPR-Datei gespeichert sein.
5. Aktivieren Sie in ProDelphi die Option: 'Programm + DLL / Mult. DLL' im Profiler-Hauptfenster.
6. Öffnen Sie das Projekt des Programms und instrumentieren Sie es. Es werden sowohl das Programm als auch die DLL's instrumentiert.

Um Messergebnisse für eine DLL oder für das Programm oder beide zu erzeugen, definieren den Compiler-Schalter PROFILE für die entsprechenden Projekte und kompilieren Sie die Projekte. Für das Teil, dessen Messergebnisse Sie nicht wollen, löschen Sie das Symbol PROFILE und (re)kompilieren es. Nur durch Definition bzw. Nicht-Definition des Compiler-Symbols können Sie die unterschiedlichen Messergebnisse erzeugen.

Wenn Sie die Messwerte für die DLL und nicht für das Programm haben wollen UND das Online-Bedienfenster benötigen:

Fügen Sie diese Zeile am Anfang der DPR-Datei des Programms ein:

```
{$DEFINE PROFILE }
```

Löschen Sie bei den Compiler Bedingungen das Symbol PROFILE.

Re-Compilieren Sie das Programm.

A.9.2. Packages

Der beste Weg, um ein Paket zu profilieren, ist:
1. Legen Sie die Quellen des Pakets in ein eigenes Verzeichnis.
2. Fügen Sie das Verzeichnis in den Suchpfad des Programms ein.
3. Instrumentieren Sie das Programm. Es wird dann auch der Code des Pakets instrumentiert.
4. Kompilieren Sie das Paket mit dem definierten Compiler Symbol PROFILE.
5. Installieren Sie das Paket.
6. Kompilieren Sie das Programm.

Wenn Sie jetzt das Programm starten, erhalten Sie die Messergebnisse für Programm und Paket.

Vergessen Sie nicht:
Jedes Mal, wenn Sie nun das Programm durch Einfügen oder Löschen von Funktionen ändern, muss wieder instrumentiert werden. Außerdem muss auch Schritt 4 und 5 wieder ausgeführt werden.

Jedes Mal, wenn Sie jetzt das Package durch Einfügen oder Löschen von Funktionen ändern, muss wieder instrumentiert werden und zusätzlich zu Schritt 4 und 5 auch Schritt 6 wieder ausgeführt werden.

A.10. Behandlung spezieller Windows- und Delphi-API-Funktionen

Einige Funktionen setzen das Programm in den Ruhezustand, bis ein Ereignis eintritt und die Funktion zurückkehrt. Es ist nicht sinnvoll, diese Wartezeiten zu messen. Aus diesem Grund sind einige Funktionen in der Einheit ‘Profint.pas' neu definiert oder werden vom Profiler im Quelltext ersetzt. Das Ergebnis ist, dass die Totzeit der aufrufenden Methode nicht gezählt wird, aber auch andere in der Zeit aufgerufenen Prozeduren während der Wartezeit noch gezählt werden.

Neudefinitionen werden immer auf die gleiche Weise durchgeführt, hier gezeigt am Beispiel der Windows-Sleep-Funktion (definiert in 'Profint.pas):

```
PROZEDUR Sleep (Zeit: DWORD);
BEGIN
        StopCounting;
        Windows.Sleep (Zeit);
        ContinueCounting;
END;
```

Aufgrund dieser Neudefinition muss die Profint-Unit nach den Units Windows und Dialogs in der USES - Anweisung genannt werden. Dies geschieht normalerweise so. Die einzige Ausnahme ist, wenn diese Units von Ihnen in die Uses-Anweisung im Implementation-Teil versetzt werden. Delphi selbst legt sie in den Interface-Teil.

Wenn Sie Funktionen finden, die auch von der Zählung auszuschließen wollen, können Sie diese gemäß dem Beispiel definieren.

A.10.1. Umdefinierte Windows-API-Funktionen

- DialogBox, DialogBoxIndirect, MessageBox, MessageBoxEx, SignalObjectAndWait
- WaitForSingleObject, WaitForSingleObjectEx, WaitForMultipleObjects, WaitForMultipleObjectsEx
- MsgWaitForMultipleObjects, MsgWaitForMultipleObjectsEx, Sleep, SleepEx, WaitCommEvent
- WaitForInputIdle, WaitMessage und WaitNamedPipe.

A.10.2. Neu definierte Delphi-API-Funktionen

- ShowMessage,
- ShowMessageFmt,
- MessageDlg,
- MessageDlgPos und
- MessageDlgPosHelp.

A.10.3. Ersetzte Delphi-API-Aufrufe

- Application.MessageBox,
- Application.ProcessMessage und
- Application.Handle Nachricht.

A.10.4. Nicht ersetzte oder umdefinierte Delphi Funktionen

Es gibt einige VCL-Funktionen, die nicht ersetzt oder neu definiert werden, weil sie Klassenmethoden sind, es wäre es viel zu kompliziert. Wenn Sie bei der Messung auf Probleme stoßen, schließen Sie die in StopCounting und ContinueCounting ein. Ein Beispiel für eine solche Prozedur ist TControl.Show.

A.11. Bedingtes Kompilieren

Conditional Compilation wird, außer bei arithmetischen Ausdrücken (wie beim Vergleich mit Konstanten), unterstützt..

Die Direktiven $IFDEF, $IFNDEF, $ELSE, $ENDIF, $DEFINE und $UNDEF werden voll unterstützt.

Die Direktiven $IF, $ELSEIF, $DEFINED(Schalter) und $IFEND werden vollständig ausgewertet inklusive der booleschen Ausdrücke AND und NOT. Arithmetische Ausdrücke werden immer als TRUE ausgewertet.

Dies sind die Grenzen:

{$IF const > x}	wird als TRUE ausgewertet	Vergleich mit einer Konstanten
{$IF SizeOf(Integer) > 10}	wird als TRUE ausgewertet	arithmetischer Ausdruck

Dies wird richtig ausgewertet:

{$IF NOT DEFINED(Schalter1) AND (DEFINED(Schalter2))}

Dieses Beispiel führt zu Problemen:

```
CONST
  xxx = 6;
{$IF xxx > 5}
  PROZEDUR AddIt (VAR erste, zweite, summe : Int64);
  BEGIN
{$ELSE}
  PROZEDUR AddIt (VAR erste, zweite, summe : Comp);
  BEGIN          <- erste Profiler-Anweisung nach diesem BEGIN eingefügt statt nach dem vorherigen BEGIN
{$ENDIF}
  summe : = erste + zweite;       <- zweite Profiler-Anweisung wird richtig hier vor END eingefügt
END;
```

Das Vermeiden des Problems ist sehr einfach, codieren Sie auf diese Weise:

```
CONST
  xxx = 6;
{$IF xxx > 5}
  PROZEDUR AddIt (VAR erste, zweite, summe : Int64);
{$ELSE}
  PROZEDUR AddIt (VAR erste, zweite, summe : Comp);
{$ENDIF}
  BEGIN                           <- erste Profiler-Anweisung wird korrekt nach diesem BEGIN eingefügt
  Summe : = erste + zweite;       <- Profiler zweite Aussage richtig hier vor END eingefügt
  END;
```

A.12. Laufzeitmessung auf Kunden - PCs

Wenn ein Programm auf einem Kunden-PC statt auf einem Entwicklungs- PC gemessen werden soll, ist wie folgt zu verfahren.

Neben den zum zu messenden Programm gehörigen Dateien müssen auch die folgenden Dateien auf den Kunden-PC kopiert werden:

- Profmeas64.dll
- ProfOnFo64.dll
- ProDVer64.dll
- Profcali64.dll
- Proflst.asc
- Profile.ini

Alle Dateien wurden von ProDelphi in das Exe-Verzeichnis des Programms auf dem Entwicklungs-PC kopiert.

A.13. Verwendungs-Einschränkungen

A.13.1. Allgemeine

Konsole-Applikationen haben kein Online-Bedien-Fenster.

Methoden in einer DPR-Datei können nicht gemessen werden.

Anonyme Methoden werden nicht instrumentiert und somit nicht gemessen. Die Instrumentierung kann manuell durch Einfügen von 2 Kommentaren in den Rumpf der Methode erreicht werden. Diese Kommentare werden beim Reinigen der Sourcen nicht entfernt. Nach der BEGIN-Anweisung wird //PROFILE-BEGIN:comment einfügen, vor der END-Anweisung dann //PROFILE-END. Siehe auch A.5.4.

Die gemessenen Zeiten unterscheiden sich etwa 10 % von denen eines nicht instrumentierten Programms. Der Grund dafür ist, dass der Programmcode nicht so oft im Cache steht wie ohne Messung. Bei einer Multi-Prozessor-Maschine können sich die Ergebnisse noch stärker unterscheiden.

Für den Zweck der Instrumentierung des Quellcodes liest ProDelphi64 die Quellen. Es ist absolut notwendig, dass das Programm ohne Fehler kompiliert werden kann. ProDelphi64 erwartet, dass der Code syntaktisch korrekt ist.

Während der Messung wird ein User-Stack von der Mess-Unit verwendet. Die maximale Stack-Tiefe ist 16000 Aufrufe.

Die maximale Anzahl von Threads, die simultan von ProDelphi64 gemessen werden kann, ist 32.

In der Freeware-Version von ProDelphi64 können nur 20 Methoden gemessen werden, in der Professional-Version 65500.

Wenn die TForm Methoden WndProc oder DefaultHandler überschrieben sind, sollten Messungen für diese Methoden deaktiviert werden. Wenn das nicht der Fall ist, wird ein großer Mess-Overhead erzeugt. In Thread-Anwendungen kann die Laufzeit inklusive Kindzeit nicht richtig gemessen werden. Dies kann dazu führen, dass die gemessene Zeit für diese Methoden größer als die Laufzeit des gesamten Programms ist. Deshalb sollten diese Methoden von der Messung ausgeschlossen werden. Dies kann leicht dadurch geschehen, indem diese Methoden in //PROFILE-NO und //PROFILE-YES Aussagen eingeschlossen werden.

Ein Problem für die Messung ist Windows selbst. Weil es ein Multitasking-System ist, können andere Aufgaben neben den von Ihnen gemessenen ausgeführt werden. Vielleicht nur für wenige Mikrosekunden. So kann Ihr Programm durch einen Task-Switch zu einer anderen Anwendung unterbrochen werden.

Vergessen Sie nicht den Einfluss des Prozessor-Cache. Sie könnten unterschiedliche Ergebnisse für jede Messung bekommen, nur weil der Code manchmal im Cache steht und manchmal nicht. Dies ist vielleicht der Grund, dass auch eine leere Methode manchmal einige CPU-Zyklen benötigt, um den Entry-Code in den Cache zu laden. **Je größer der Cache, desto besser sind die Ergebnisse! Die Profiler-Prozeduren verwenden den Cache auch!**

Dann ist da die CPU selbst. Die modernen CPU's sind in der Lage, Anweisungen parallel auszuführen. Wenn Profiler-Anweisungen im Code stehen, ist die Parallelität anders ohne diese Anweisungen. Das ist ein weiterer Grund, warum sich die Laufzeit mit Messung von der ohne Messung unterscheidet.

Alle meine Tests haben gezeigt, dass je größer der Cache, desto kleiner die Differenz zwischen der tatsächlichen und der gemessenen Laufzeit.

Wenn Ihr gemessenes Programm Threads verwendet, sind die Ergebnisse weniger korrekt. Der Grund dafür ist, dass ein Threadwechsel nicht zum Zeitpunkt des Wechsels erkannt wird. Er wird beim nächsten Methodeneintritt erkannt.

Seien Sie sich bewusst, dass Sie bei Methoden, die I/O-Anrufe tätigen, auch bei jedem Aufruf unterschiedliche Ergebnisse erhalten können. Der Grund dafür ist der Disk-Cache von Windows. Manchmal schreibt Windows in den Cache, manchmal direkt auf die Festplatte.

A.13.2. Delphi SpeedUp / FastMM Units

Die DelphiSpeedUp Units RtlVclOptimize und VclFixUpPack werden von der Instrumentierung ausgeschlossen, weil sie ihren eigenen Code in die Delphi Standard-Units kopieren. Dies führt zu einem Absturz der Anwendung, wenn diese Units durch die Instrumentierung geändert werden.

FastMM Einheiten aus den oben genannten Gründen ebenfalls nicht instrumentiert werden.

A.13.3. Abgebrochene Methoden

Wenn Methoden abgebrochen werden, z. B. wenn ein Programm, ohne dass alle Threads beendet sind, endet, stehen keine Messergebnisse für die Methoden, deren Exit-Code nicht ausgeführt wurden, zur Verfügung.

A.13.4. Messen mehrerer Anwendungen

Wenn mehr als eine Anwendung gemessen werden muss, ist es wichtig, dass die EXE-Dateien in separaten Verzeichnissen gespeichert sind. Der Grund dafür ist, dass die Dateien mit den Einstellungen für die Messung und die mit den Methodennamen einen festen Namen haben (profile.ini und proflst.asc).

Es ist nicht möglich, mehr als eine Anwendung gleichzeitig zu messen! Nur eine instrumentierte Anwendung zur gleichen Zeit darf ausgeführt werden können, sonst werden falsche Ergebnisse produziert!

A.13.5. Ausschluss der Instrumentierung von Verzeichnissen für alle Projekte

Sie können dies dadurch erreichen, indem Sie diese Verzeichnisse als Zeichenfolge unter dem Registry-Schlüssel für ProDelphi64 eintragen:
- Fügen Sie die Zeichenfolge ‘GLOBALEX’ in der Registry unter HCU\Software\ProDelphi64 ein
- Setzen Sie den Wert auf die zu exkludierenden Verzeichnisse (z. B. ‘D:\components\Visual;D:\components\Graphic’)
Wenn ein Verzeichnis und alle seine Unterverzeichnisse exkludiert werden sollen, fügen Sie ‘*’ an den Verzeichnisnamen an (z.B. ‘D:\components*’).

A.14. Assembler-Code

Reine Assembler-Prozeduren können nicht gemessen werden.

A.15. Ändern von durch ProDelphi64 instrumentiertem Code

Während der Arbeit an der Optimierung des Programms können Sie Ihren Code ändern. Die einzige Einschränkung ist, dass, wenn Sie eine neue Methode definieren und wollen, dass sie gemessen wird, Sie Ihren Code von ProDelphi64 erneut instrumentieren lassen müssen. Es ist nicht notwendig, die alten von ProDelphi64 eingefügten Anweisungen zu löschen.

A.16. Versteckte Leistungsverluste / Tipps zur Optimierung

ProDelphi64 misst die Laufzeit der Methoden-Bodys. Dies bedeutet, dass der Entry-Code einer Methode, der z. B. Variablen auf den Stack schreibt, in der aufrufenden Methode gemessen wird! Die erste Möglichkeit, um einen Zeitstempel zu setzen, ist direkt hinter der BEGIN-Anweisung. Dies könnte als ein Nachteil gegenüber anderen Profilern gesehen werden. Aber wenn Sie das wissen, ist diese Tatsache kein Nachteil mehr. Wie auch immer, die Änderung der Anzahl der Parameter einer Methode ändert immer die Laufzeit der aufrufenden Methode (auch für andere Profiler).

Nachfolgend drei gemessene Beispiele dafür.

- Übergeben von Parametern:

```
FUNKTION TestFunction(s : String): Integer;      // Laufzeit 5 CPU-Cycles + 983 in der aufrufenden Methode
BEGIN
  Result: = Ord (s[1]);
END;

FUNKTION TestFunction(CONST s: String): Integer;      //Laufzeit 5 CPU-Cycles + 645 in der aufr. Methode (-33 %)
BEGIN
  Result: = Ord (s [1]);
END;
```

```
FUNKTION TestFunction : Integer;        // Laufzeit 159 CPU-Zyklen + 126 Zyklen in der aufrufenden Methode
VAR
  i : Integer;
BEGIN
  FOR i: = 1 TO 10 DO
      Result : = LastFunction;
      IF Result > 0 THEN
        exit
      ELSE
        Result: = -1;
END;
```

```
FUNKTION TestFunction : Integer;        // Laufzeit 159 CPU-Zyklen + 6.932.128 Zyklen in der aufrufenden Methode
VAR
  i      : Integer;
  YYS : array [1..32000] of Integer;    // increasing durch Initialisierung dieser lokalen Variablen verursacht!
  yyv   : array [1..32000] of String;
BEGIN
  FOR i: = 1 TO 10 DO
      Result: = LastFunction;
      IF result > 0 THEN
        exit
  ELSE
      Result: = -1;
END;
```

- GoTo-Anweisungen

```
FUNKTION TestFunction : Integer;        // Laufzeit 159 CPU-Zyklen + 126 Zyklen in der aufrufenden Methode
VAR
  i : Integer;
BEGIN
  FOR i : = 1 TO 10 DO
      Result : = LastFunction;
      IF Result > 0 THEN
        exit
      ELSE
       Result: = -1;
END;
```

```
FUNKTION TestFunction: Integer;         // Laufzeit 159 CPU-Zyklen + 177 Zyklen in der aufrufenden Methode (+
40 %)
VAR
  i      : Integer;
  Label Final;                          // Ursache der zusätzlichen Laufzeit
BEGIN
  FOR i : = 1 TO 10 DO
      Result : = LastFunction;
      IF Result > 0 THEN
        GoTo Final                      // im Zusammenhang mit diesem GoTo
      ELSE
        Result: = -1;
Final :
END;
```

A.17. Fehlermeldungen

Bei Fehlern wird eine Fehlermeldung von ProDelphi64 unten im Hauptfenster und im Log (z. B. file-I/O-error) angezeigt. Wenn das geschieht, schauen Sie in das Verzeichnis mit den zu instrumentierenden Units.

Instrumentieren einer Datei ist wie folgt implementiert:

- die Originaldatei *.pas wird umbenannt in *.pas_sav (oder *.dpr in *.dpr_sav bzw. *.inc in *.inc_sav).

- dann wird die umbenannte Datei analysiert und instrumentiert, die Ausgabe steht dann in einer *.pas-Datei.

- der letzte Schritt, eine Datei zu verarbeiten, ist die gesicherte Datei zu löschen, außer vorher tritt ein Fehler auf.

Dies wird für alle Dateien eines Verzeichnisses getan. Im Fall, dass ein Fehler auftritt, können Sie die gespeicherte Sicherungsdatei in *. Pas / *. DPR / *. Inc umbenennen.

Bevor Sie dies tun, lohnt es sich vielleicht, einen Blick in die Ausgabe-Datei zu werfen. Im Falle eines Parsing-Fehlers können Sie die Original-Datei + die unvollständige Ausgabe-Datei an den Autor zum Zweck der Analyse senden.

A.18. Sicherheitsaspekte

- Sichern Sie alle Ihre Quellen vor der Instrumentierung (z.B. durch Zippen).

- ProDelphi64 prüft vor der Instrumentierung, ob Sie genügend Platz auf der Festplatte haben, um die instrumentierte Datei beim Instrumentieren zu speichern. ProDelphi64 geht davon aus, dass die Ausgabedatei 3 mal den Platz der ursprünglichen Datei (normalerweise wird weniger benötigt) verwendet. Wenn nicht genügend Speicherplatz vorhanden ist, stoppt die Instrumentierung.

A.19. Instrumentieren, Reinigen oder Ergebnisanzeige per Kommandozeile

ProDelphi kann von der Kommandozeile aus (oder über eine Batch-Datei) gestartet werden. Je nach dem Kommandozeilenparameter wird ProDelphi die Instrumentierung ausführen, die Quelldateien reinigen oder den Viewer starten.

A.19.1. Instrumentierung

Wenn als Argumente die Delphi-Version und der Parameter /PROFILE angegeben ist, instrumentiert ProDelphi das genannte Programm und beendet sich danach.

Syntax:

Profiler pfad\programm.dpr /Ddelphi-Version /PROFILE
Profiler pfad\programm.dpr /Ddelphi-Version /PROFILE [/C=konfiguration] (ab Delphi 2009)
konfiguration: Debug, Release oder benutzerspezifische Konfiguration
Wenn nicht angegeben wird die im Projektfile gespeicherte aktive genommen.

Konfiguration: Debug, Release oder benutzerspezifische Konfiguration
Wenn nicht angegeben wird die im Projektfile gespeicherte aktive genommen.

Delphi Versionen:	5 .. 7	2005 .. 2007	2009	2010	XE	XE2 .. XE8	10.0 .. 10.4
/D parameter:	5 .. 7	9 .. 11	12	13	14	15 .. 21	22 .. 26

Vorsichtsmaßnahmen:

- Instrumentierung sollte vorher interaktiv erfolgen, um sicher zu sein, dass alle erforderlichen Dateien (z.B. DOF/dproj-Datei) vorhanden sind und keine Kompilierfehler mehr auftreten.
- Instrumentierung sollte nicht zu einer Warnung im Profiler Logbuch-Fenster führen.

Beispiel:

cd ProDelphi
Profiler F:\AppDir\Testprogram.dpr /D15 /PROFILE (für Delphi XE2)

A.19.2. Reinigung

Wenn als Argumente die Delphi-Version und der Parameter /CLEAN, entfernt ProDelphi die Instrumentierung des genannten Programms und beendet sich danach.

Syntax:

Profiler Pfad\programm.dpr /Ddelphi-Version /CLEAN

Beispiel:

```
cd ProDelphi
Profiler F:\AppDir\Testprogramm.dpr  /D21 /CLEAN          (für Delphi XE8)
```

A.19.3. Automatisches Öffnen des Viewers

Wenn als Argumente die Delphi-Version und der Parameter /VIEW angegeben werden, startet ProDelphi automatisch den Viewer und zeigt die Messergebnisse an.

Syntax:

Profiler Pfad\programm.dpr /Ddelphi-Version /VIEW

Beispiel:

```
cd ProDelphi
Profiler F:\AppDir \Testprogram.dpr /D23 /VIEW          (für Delphi 10.1)
```

A.20. Unterstützung der Landessprache

Derzeit wird eine englische und eine deutsche Benutzeroberfläche unterstützt. Die zu Verwendende ist bei der Installation von ProDelphi64 mit dem Setup-Programm anzugeben. Aber auch später kann ProDelphi64 zwischen diesen beiden Sprachen wechseln. Dies kann durch das System-Menü erfolgen. Es verfügt über zwei zusätzliche Einträge: 'English' und 'German' (= deutsch).

Wenn eine andere Sprache verwendet werden soll: Es ist eine Datei mit dem Namen 'TranslateMe.LAN' installiert. Es enthält die englischen Text-Strings. Durch die Übersetzung der Texte und Umbenennen der Datei in ‘Deutsch.LAN’ kann man eine Benutzeroberfläche für seine eigene Sprache erzeugen.

B. Post Mortem Review

Wie eingangs erwähnt, kann ProDelphi64 Ihre Quellen mit Aufrufen zur Ermöglichung einer Post-Mortem-Analyse instrumentieren. Die Aufrufe werden am Anfang und am Ende einer Methode eingefügt.

Im Falle eines Abbruchs wegen einer Exception, erscheint eine MessageBox in der Ihnen die Datei, in der der Aufruf-Stack aufgelistet ist mitgeteilt (ProgramName.PMR).

Auch hier können die Kommentare //PROFILE-NO und //PROFILE-YES Teile des Quellcodes auszuschließen.

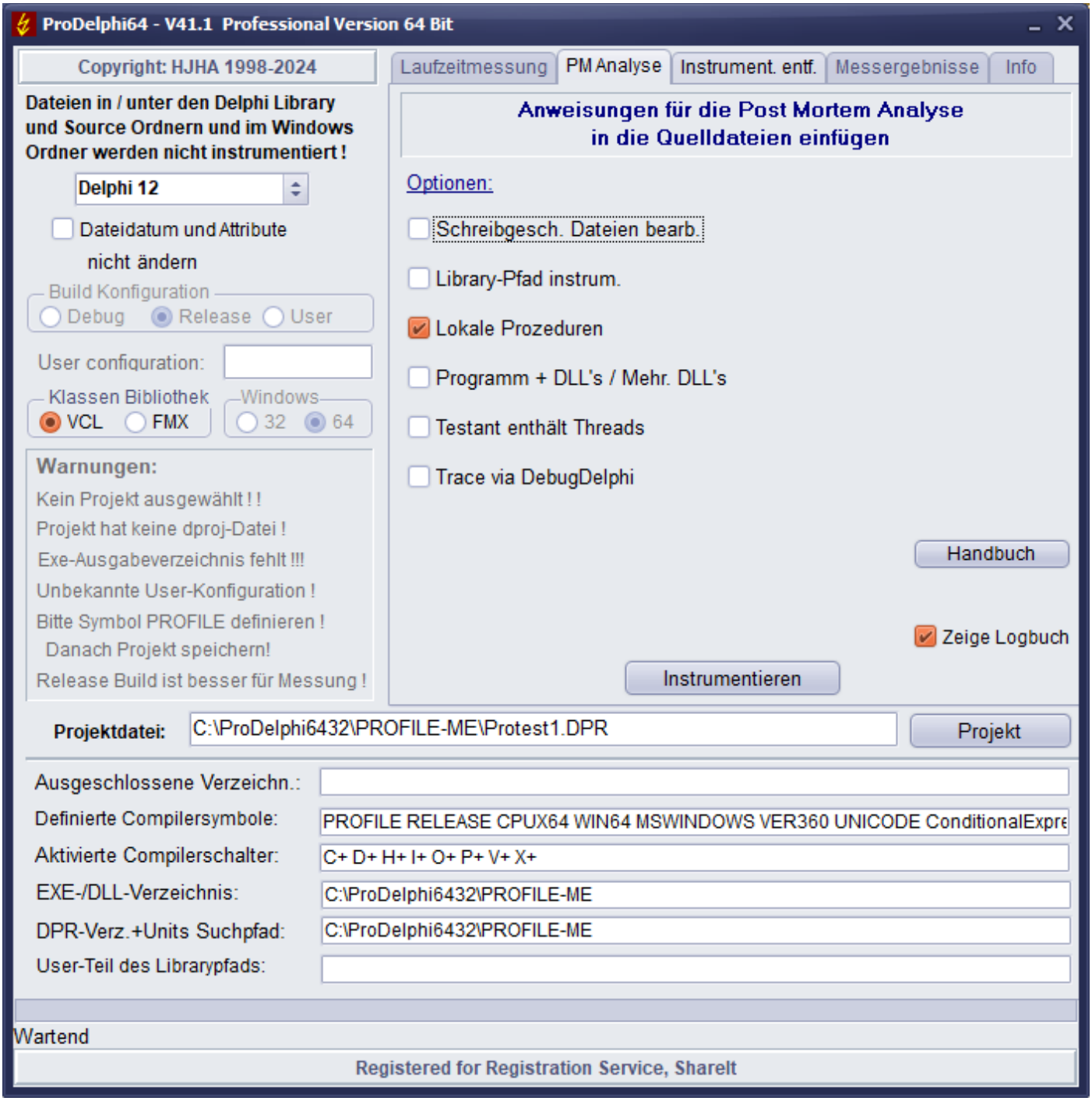
Für mögliche Optionen siehe Kapitel A4.

Die Handhabung von ProDelphi64 ist die gleiche wie für die Laufzeitmessung. Sie müssen auch das Compiler Symbol PROFILE definieren.

Wenn Sie Ihr Programm für die Post-Mortem-Analyse instrumentiert haben, es aus der IDE heraus starten, eine Exception auftritt und Delphi eine MessageBox ausgibt, müssen Sie Ihr Programm fortsetzen. Wenn Sie die Option 'Stopp at exception' deaktiviert haben, wird Delphi nicht aktiv.

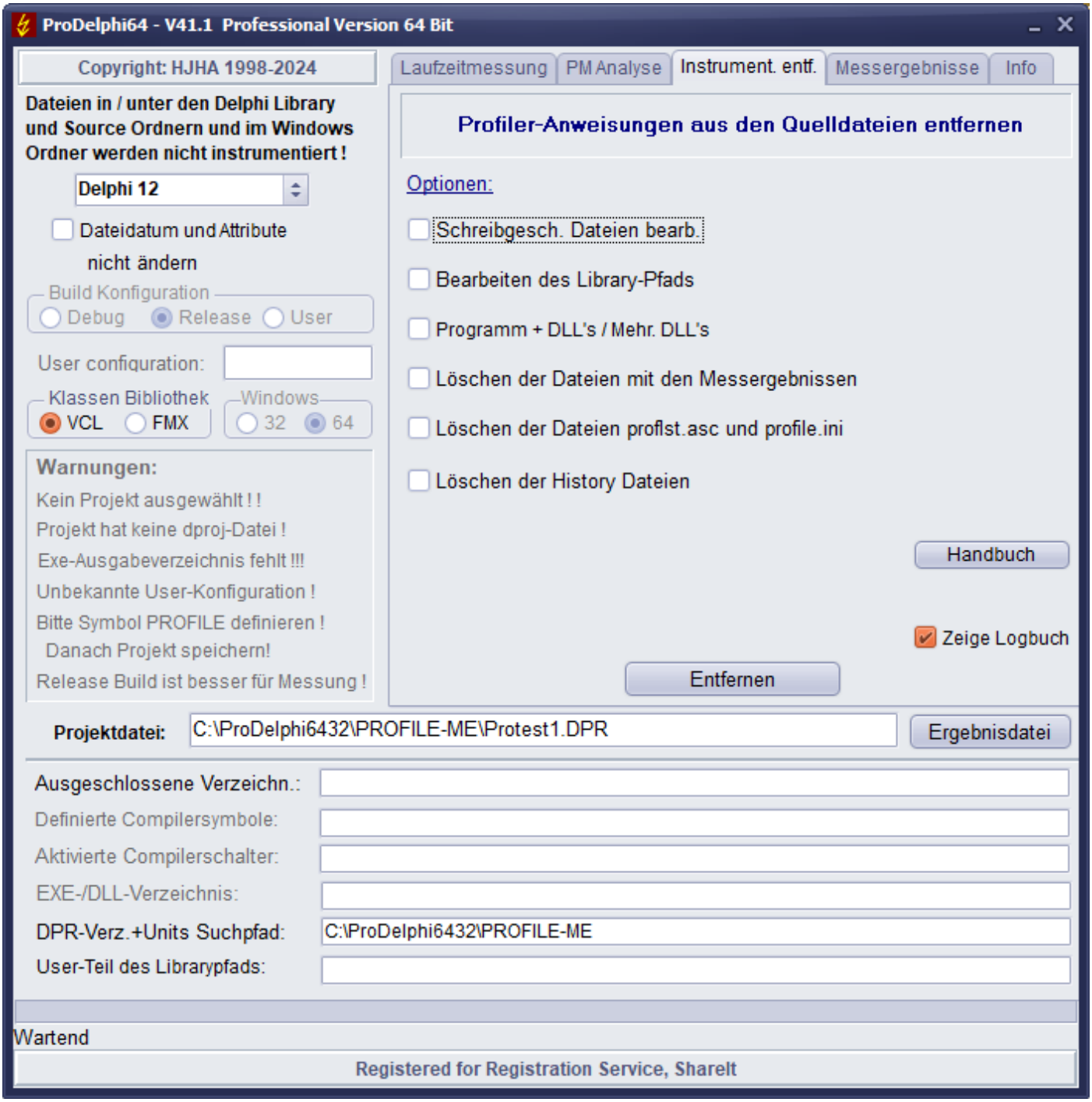
Nutzungsbeschränkung: Ein Stack-Overflow kann nicht abgefangen werden, weil ProDelphi64 selbst Stack braucht. Und wenn kein Stack-Speicher mehr frei ist, kann auch ProDelphi64 nicht richtig arbeiten. Der Überlauf kann genauso gut in den ProDelphi64 Stack-Trace-Routinen auftreten. Damit kann ProDelphi64 nicht umgehen.

Wenn die **Trace-Option** aktiviert ist, werden zusätzliche WriteLn Anrufe in der Quelldatei eingefügt. Diese WriteLn Anrufe produzieren Trace-Informationen, die mit DebugDelphi angesehen werden können. Zu diesem Zweck muss DebugDelphi installiert und gestartet werden.


ProDelphi64 - V41.1 Professional Version 64 Bit
Copyright: HJHA 1998-2024
Laufzeitmessung
PM Analyse
Instrument. entf.
Messergebnisse
Info
Dateien in / unter den Delphi Library und Source Ordnern und im Windows Ordner werden nicht instrumentiert !
Delphi 12
Dateidatum und Attribute nicht ändern
Build Konfiguration
Debug
Release
User
User configuration:
Klassen Bibliothek
VCL
FMX
Windows
32
64
Warnungen:
Kein Projekt ausgewählt ! !
Projekt hat keine dproj-Datei !
Exe-Ausgabeverzeichnis fehlt !!!
Unbekannte User-Konfiguration !
Bitte Symbol PROFILE definieren !
Danach Projekt speichern!
Release Build ist besser für Messung !
Anweisungen für die Post Mortem Analyse in die Quelldateien einfügen
Optionen:
Schreibgesch. Dateien bearb.
Library-Pfad instrum.
Lokale Prozeduren
Programm + DLL's / Mehr. DLL's
Testant enthält Threads
Trace via DebugDelphi
Handbuch
Zeige Logbuch
Instrumentieren
Projektdatei:
C:\ProDelphi6432\PROFILE-ME\Protest1.DPR
Projekt
Ausgeschlossene Verzeichn.:
Definierte Compilersymbole:
PROFILE RELEASE CPUX64 WIN64 MSWINDOWS VER360 UNICODE ConditionalExpre
Aktivierte Compilerschalter:
C+ D+ H+ I+ O+ P+ V+ X+
EXE-/DLL-Verzeichnis:
C:\ProDelphi6432\PROFILE-ME
DPR-Verz.+Units Suchpfad:
C:\ProDelphi6432\PROFILE-ME
User-Teil des Librarypfads:
Wartend
Registered for Registration Service, SharelT

C. Entfernen der Instrumentierung (Reinigung der Quellen)

Wenn Sie alle Zeilen löschen wollen, die ProDelphi64 in Ihre Quellen eingefügt werden hat, verwenden Sie 'Instrumentierung entfernen'.



Es ist nicht notwendig, die Quellen zu reinigen, um Ihr Programm ohne Zeitmessung laufen zu lassen. In diesem Fall löschen Sie einfach das Compiler-Symbol ‘PROFILE’ in Ihren Projekt-Optionen.

Es ist auch nicht erforderlich, die Quellen zu reinigen, wenn Sie die Quellen neu instrumentieren wollen. Jede Instrumentierung löscht automatisch alle alten ProDelphi64 Aufrufe im Quellcode und fügt neue ein. Zu diesem Zweck scannt ProDelphi64 den Code nach Zeilen, die beginnen mit

{$IFDEF PROFILE} und {$IFNDEF PROFILE}

und löscht sie komplett (außer Sie habe mehr als 1 Leerzeichen zwischen $IFDEF/$IFNDEF und PROFILE).

Die Option ‘Dateidatum und Attribute nicht ändern’ bewirkt, dass das Dateidatum bei der Instrumentierung um 2 Sek. inkrementiert und bei der Reinigung um 2 Sek. dekrementiert wird. Dies ermöglicht es, dass in einem Quellcodeverwaltungssystem die Datei beim Check-In das gleiche Datum hat wie beim vorherigen Check-out.

D. Kompatibilität

ProDelphi64 wurde getestet unter
- Windows 7, Windows 10, Windows 11
- AMD Athlon X2 64, Intel Core i5

E. Installation von ProDelphi64

ProDelphi64 wird am bequemsten mit dem mitgelieferten Setup-Programm (setup.exe) installiert. Dieses Programm kopiert alle notwendigen DLL’s in das Installationsverzeichnis und alle benötigten Units ins Delphi-LIB-Verzeichnis. Die Editor-Schnittstelle wird registriert. Auch erstellt es einen Eintrag in der Liste der Programme (Windows Startmenü / Programme) und integriert ProDelphi64 in das Delphi-Tools-Menü.

F. Beschreibung der Ergebnis-Dateien (für Datenbank-Export und Viewer)

Die Ergebnis-Datei kann auch für den Import in eine Datenbank (z.B. Paradox) oder ein Tabellenkalkulationsprogramm wie OpenOffice Calc verwendet werden.

Datei-Inhalt 'progname.txt' (eine Zeile für jede Methode):

run; unit; Klasse; Prozedur; % der RT; Aufrufe; min. RT exkl. Kind o. 0; durchschn. RT exkl. Kind; max. RT exkl. Kind o. 0;
RT-Summe exkl. Kind; min. RT inkl. Kind o. 0; durchschn. RT inkl. Kind, max. RT inkl. Kind oder 0; RT-Summe inkl. Kind;
% der RT inkl. Kind; Prozedur-Nr; Aufrufgraph Daten;

Beschreibung von Call-Graph-Daten:

0; 0; wenn keine Daten vorhanden sind

oder

Aufruf-Von-Informationen Aufruf-Für-Informationen

Beschreibung der Aufruf-Von-Information:

0;	wenn nicht aufgerufen (Top-Level-Prozedur)
1 .. 15; zwischen 1 und 15 Sätze von Typ-1-Info	wenn bis von 15 Methoden aufgerufen
16; 15 Sätze von Typ-1-Info und 1 Satz Typ-3-Info	wenn sie von 16 oder mehr Methoden aufgerufen

Beschreibung der Calling-To-Information:

0;	wenn keine Methoden aufgerufen
1 .. 15; zwischen 1 und 15 Sätze von Typ-2-Info	wenn bis zu 15 aufgerufene Methoden
16; 15 Sätze von Typ-1-Info und 1 Satz Typ-4-Info	bei 16 oder mehr aufgerufenen Methoden

Beschreibung von Art-1-Info:
Nummer aufrufende Methode; Anzahl Aufrufe; Runtime inkl. Kindzeit der aufrufenden Methode;

Beschreibung von Art-2-Info:
Nummer der aufgerufenen Methode; Anzahl Aufrufe; Runtime inkl. Kindzeit der aufgerufenen Methode;

Beschreibung der Typ-3-Info:
0; Anzahl der Aufrufer die nicht in den ersten 15 Methoden enthalten sind; Runtime inkl. Kind dieser Prozeduren;

Beschreibung von Art-4-Info:
0; Anzahl der aufgerufenen Methoden, die nicht in den ersten 15 Methoden enthalten sind; Runtime inkl. Kindzeit dieser Methoden;

Die Methoden-Namen können durch die Suche der Methodennummer in Proflist.Asc gefunden werden. Für instrumentierte Methoden gibt es folgende Informationen:

Methodennummer; Unit Name; Klassenname; Methodenname; Dateiname; Zeilennummer in der Datei

Datei-Inhalt 'progname.tx2' (eine Zeile für jeden Lauf):

run; CPU-Taktrate, Schlüssel, Überschrift für diesen Lauf //Schlüsselwort ist entweder MINIMAXON oder MINIMAXOFF

G. Update / Upgrade von ProDelphi64

Update bzw. Upgrade der Freeware-Version kann über die Autoren-Homepage geladen werden. Jede neue Version wird automatisch dort gespeichert.

H. Wie bestelle ich die Professional-Version

Kunden, die die Professional-Version verwenden wollen, können sie über den Digistore24 Registrierungs Service bestellen. Ein spezieller Download-Link für das Herunterladen der Professional-Version und einen Registrierungsschlüssel werden per E-Mail versandt. Starten Sie einfach die professionelle Version von ProDelphi64 (Profiler64.exe), wählen Sie die Seite für die Registrierung und geben Sie die Registrierungsnummer ein. Beim nächsten Start von ProDelphi64 wird dann der Profi-Modus freigeschaltet. Dieser Schlüssel gilt auch für ein Upgrade auf neuere Versionen.

I. Verfasser

Helmuth J.H. Adolph (Dipl.-Inform.)
Am Grünerpark 17
90766 Fürth
Deutschland

E-Mail: helado@prodelphi.de
Homepage: https://www.prodelphi.de

J. Geschichte

Version 1.0: 9/97	Erste Veröffentlichung.
Version 2.0: 2/98	Erfolgreich eingesetzt, um VICOS P500 für Sixth Railways Projekt (China) zu optimieren.
Version 3.0: 4/98	Erhöhte Genauigkeit, über Compuserve veröffentlicht
Version 3.1: 5/98	Verbesserte Granularität (1 CPU - Zyklus), Torry Delphi Pages veröffentlicht
Version 4.0: 10/98	Viewer hinzugefügt, Export in Datenbank, Unterstützung von Delphi 4.
Version 5.3: 12/98	DLL-Unterstützung hinzugefügt
Version 6.0: 2/99	Behandlung von Nur-Lese-Attribut, DLL-Unterstützung verbessert, ProDelphi64 Homepage
Version 25.0: 8/10	Anpassung an Delphi XE.
Version 26.0: 9/11	Anpassung an Delphi XE2
Version 27.0: 2/12	Anpassung an Delphi XE3.
Version 28.0: 4/13	Anpassung an Delphi XE4.
Version 29.0: 4/13	Anpassung an Delphi XE5.
Version 29.1: 10/13	Environment variable im EXE-Pfad wurde nicht ausgewertet, gefixt. Für Programme, die mit Delphi 2007 kompiliert werden sollen, kann Debug- oder Release-Build gewählt werden (sie haben verschiedene Conditionals: DEBUG/RELEASE).
Version 29.2: 11/13	Sourcecode Navigation für XE5 korrigiert.
Version 30.0: 4/14	Anpassung an Delphi XE6.
Version 31.0: 5/14	Exkludierung von Verzeichnissen für alle Projekte.
Version 30.2 5/14	Viewer Exception und Größenproblem gefixt.
Version 30.3 (6/14)	Eine Fehlermeldung eingefügt, eine gefixt.
Version 30.4 (7/14)	Fehlende deutsche Übersetzung eingefügt. Warnung ‘Kein Online-Bedienfenster für Konsolen-Applikationen eingefügt. Fehlenden Unit-Namen bei Konsolen-Applikationen eingefügt.
Version 31.0 (9/14)	Anpassung an Delphi XE7.
Version 31.0a (1/15)	Bugfix im Interface-File (MessageDLG call ging nicht mit Delphi XE6 und XE7). Default für Messungsstart auf automatisch gesetzt (für fehlende Profile.ini) Warnung im Online Bedien Fenster, wenn profile.ini fehlt. Setup fand Delphi 2007 nicht.
Version 31.1 (2//15)	Historische Schreibweise von Compiler Direktiven [e.g. (*$I filename *) statt of {$I filename }] implementiert.
Version 31.2 (4/15)	Viewer verbessert.
Version 32.0 (4/15)	Anpassung an Delphi XE8.
Version 33.0 (9/15)	Anpassung an Delphi 10 - Seattle.
Version 34.0 (4/16)	Anpassung an Delphi 10.1 - Berlin.
Version 34.1 (8/16)	Kommandozeilenparameter vereinfacht.

Version 35.0 (3/17)	Anpassung an Delphi 10.2 - Tokyo.
Version 35.1 (4/17)	Viewer-Bedienung verbessert, Profiling-Log abschaltbar.
Version 36.0 (3/18)	Optische Verbesserung
Version 36.1 (7/18)	Druckfunktion für Tabellen korrigiert
Version 37.0 (11/18)	Anpassung an Delphi 10.3 Rio
Version 37.1 (3/19)	Problem mit sehr langem Units Search Path beseitigt
Version 37.2 (4/19)	Exclude-Directory Window jetzt mit möglicher Größenänderung
Version 37.3 (5/19)	Start schneller gemacht
Version 37.4 (6/19)	Fehlende Überschriften im Export-File und im Excludierunsfenster eingebracht
Version 37.5 (6/19)	Bugfix: Keine Messwerte, wenn HandleMessages oder ProcessMessages aufgerufen wurde und die Messung durch ausgewählte Methoden gestartet wurde.
Version 38.0 (1/20)	Anpassung an Delphi 10.4 Sydney
Version 38.1 (9/20)	Bugfix für Messung von Programmen mit Threads
Version 38.2 (10/20)	Automatsches Instrumentieren verbessert
Version 38.3 (3/21)	Das Call Graph Fenster in Größe änderbar, Call Graph kann mit ESC-Taste geschlossen werden. Im Viewer selektierte Methode wird durch blauen Hintergrund markiert. Ctrl + Home und Ctrl + End für Viewer realisiert Suche im Viewer verbessert.
Version 38.4 (8/21)	Problem mit Skalierung des Textes auf 150 % durch Windows gelöst. Anzeigeprobleme betreffend exkludierter Verzeichnisse beseitigt.
Version 39.0 (11/21)	Anpassung an Delphi 11
Version 39.1 (12/21)	Interface Unit verbessert Schließen des Online Operation Windows verursachte Exception bei Programmende, gefixt.
Version 39.2 (3/22)	Interne Optimierungen
Version 39.3 (6/22)	UI verschönert
Version 40.0 (10/22)	Hinweis für Messen auf PC's mit Prozessoren mit Turbo Boost, Messen von Wartezeiten
Version 40.2 (4/23)	Bugfix: Schließen des Online Operation erzeugte Fehler wenn das gemessene Programm beendet wurde Verbesserung: Viewer zeigt Werte mit lokalem Dezimaltrenner (. ,)
Version 40.3 (4/23)	Falsche Warnmeldung gelöscht
Version 40.4 (5/23)	Ein-Klick Aufklappen der Browser implementiert Callgraph- und Viewer-Fenster können jetzt verbreitert werden
Version 40.5 (7/23)	Kosmetische Verbesserung des Viewers Logbuch Verbesserung Einige falsche Hint-Texte korrigiert Progressbar gefixt Abschaltung der Hints im Viewer deaktiviert auch die Hints im Callgraph Englisch korrigiert Anzahl der Nachkommastellen im Viewer wählbar: 0, 1, 2 oder 3 Stellen
Version 41.0 (11/23)	Unterstützung von Delphi 12 Viewer Font für bessere Lesbarkeit geändert Protokoll Font ebenfalls
Version 41.1 (2/24)	Einstellungen, die aus der Projektdatei eines Programms gelesen wurden und von ProDelphi angezeigt werden, können nicht von ProDelphi geändert werden,
Version 41.2 (4/24)	Fonts im Callgraph verbessert
Version 42.0 (10/24)	UI in Niederländisch, Französisch und Italienisch
Version 42.1 (11/24)	Maximal 65500 Methoden können gemessen werden, statt 64000.
Version 43.0 (3/25)	Unterstützung der 64 Bit IDE von Delphi 12.3
Version 43.1 (4/25)	Es können Programme mit mehr als 65500 Prozeduren gemessen werden. Dabei werden maximal 65500 Methoden instrumentiert und somit nicht alle gemessen. Die Zeiten der nicht gemessenen Methoden werden evtl. in gemessenen Methoden als Laufzeit mitgemessen.

K. Literatur

How to optimize for the Pentium family of microprocessors by Agner Fog / 1998-08-01
C/C++ user journal 'A Testjig Tool for Pentium Optimization' by Steve Durham (December 1996).