NEWS · SEPTEMBER 22, 2026

RAD Studio 13.2 gibt Delphi endlich einen modernen Linux-Compiler

RAD Studio 13.2 hebt den Delphi-Linux-Compiler in einem einzigen Release von LLVM 3.3 auf LLVM 20.1 -- was dieser Sprung wirklich bringt, welche IDE- und VCL-Neuerungen mitkommen, und die eine Generics-Änderung, die Ihren Build anhalten wird.

By Dr. Holger Flick

Hin und wieder steht in einer Release-Notiz eine Zahl, bei der man zweimal hinsieht. Für mich war das in RAD Studio 13.2 die Zahl 3.3.

Das ist die LLVM-Version, auf der der Delphi-Compiler für Linux Intel 64-Bit bis vergangene Woche aufgebaut war. Marco Cantù hat es unmissverständlich formuliert, als er die Arbeit ankündigte: "The current compiler is based on LLVM 3.3, which was released a long time ago." Vor langer Zeit heißt Juni 2013. In der Zwischenzeit ist der Rest der Welt weitergezogen -- Clang, Rust, Swift und die Hälfte der Toolchains, auf die Sie sich täglich verlassen, arbeiten seit Jahren mit modernen LLVM-Releases. Delphis Linux-Backend blieb, wo es war. Mit RAD Studio 13.2 Florence, veröffentlicht am 17. September 2026, springt es direkt auf LLVM 20.1 -- dreizehn Jahre Compilerforschung in einem einzigen Update.

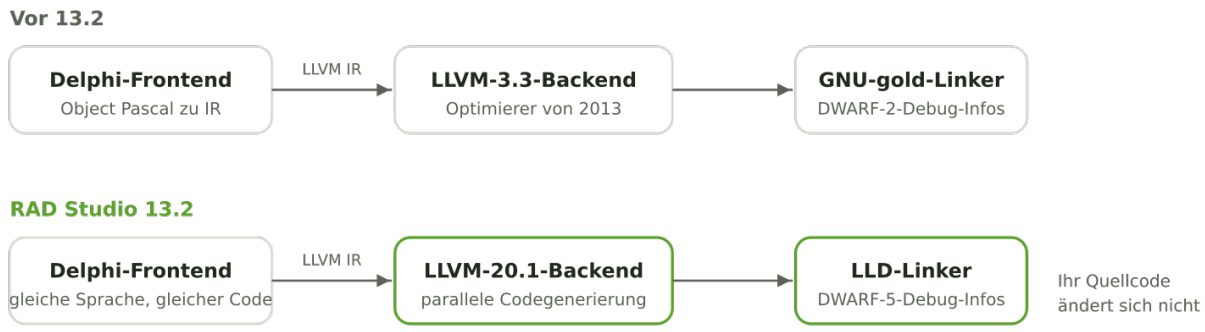
Bevor mir jemand Jubelberichterstattung vorwirft, grenze ich das sauber ein: LLVM 20.1 ist auch nicht die neueste LLVM-Version. Das Projekt steht inzwischen bei 23.x, Delphi liegt also weiterhin einige Hauptversionen hinter dem letzten Stand. Der Unterschied ist, dass drei Jahre Rückstand für eine kommerzielle Toolchain normal sind, während dreizehn Jahre Rückstand ein echtes Handicap waren.

Ich möchte durchgehen, was dieser Sprung tatsächlich bringt, denn die Marketing-Zahlen dazu sind ungewöhnlich gut -- und ausnahmsweise ungewöhnlich gut dokumentiert. Ich möchte aber ebenso ehrlich über die Teile sprechen, die Sie einen Nachmittag kosten: eine neue glibc-Untergrenze, eine Einschränkung bei den Editionen und eine kleine Änderung an den Generics-Regeln, die mit Linux nichts zu tun hat und Ihren Windows-Build genauso bereitwillig anhält. Unterwegs schauen wir uns die IDE-Arbeit und die zwei neuen VCL-Steuerelemente an, denn das sind die Teile, die Sie jeden Tag anfassen.

Warum LLVM 3.3 das eigentliche Problem war

Bevor wir über Geschwindigkeit reden, hilft es zu verstehen, woher die Obergrenze des alten Compilers kam.

Ein solcher Compiler hat zwei Hälften. Das Frontend liest Object Pascal, prüft es und überführt es in LLVMs eigene Zwischendarstellung -- LLVM IR, eine Art portabler Pseudo-Assembler. Das Backend nimmt diese IR, optimiert sie und macht daraus Maschinencode für die Ziel-CPU. Embarcadero schreibt das Frontend. LLVM ist das Backend. Somit existierte jede Optimierung, die LLVMs Mitwirkende zwischen 2013 und 2026 hinzugefügt haben -- und das waren sehr viele --, für Ihre Linux-Builds schlicht nicht. Sie haben modernen Delphi-Code durch einen dreizehn Jahre alten Optimierer geschickt.



Zwei Hälften des Compilers -- nur eine davon war in 2013 eingefroren

Achten Sie darauf, was sich in diesem Bild *nicht* ändert: das Frontend und damit Ihr Code. Das hier ist ein

Austausch des Backends. Dieselben Units, die Sie heute kompilieren, geben Sie der neuen Toolchain -- und genau deshalb ist die Performance-Geschichte glaubwürdig und nicht verdächtig.

Drei Dinge haben sich gemeinsam bewegt, so Embarcaderos eigener Beitrag zur neuen Linux-Toolchain: das

Compiler-Backend ging auf LLVM 20.1, der GNU-gold-Linker wurde durch "a customized version of LLVM's LLD linker" ersetzt, und die Debug-Informationen wechselten von DWARF 2 auf DWARF 5. DWARF ist das Format für Debug-Informationen, das Linux-Debugger lesen; Version 2 stammt aus dem Jahr 1993, und der Unterschied darin, was ein Debugger Ihnen über Ihre Variablen sagen kann, ist alles andere als subtil.

> **Woher die Zahlen stammen**
> Jede Zahl in diesem Beitrag stammt aus Embarcaderos eigenen internen Tests, veröffentlicht im Firmenblog und in der Pressemitteilung. Ich habe diese Benchmarks nicht selbst nachgestellt. Hersteller-Benchmarks sind Hersteller-Benchmarks -- nehmen Sie sie als ehrlichen Hinweis auf Richtung und Größenordnung, nicht als Versprechen für *Ihr* Projekt. Ich zeige Ihnen die veröffentlichte Hardware und die Einschränkungen, damit Sie selbst urteilen können.

Die Geschwindigkeit, mit allen Sternchen

Zwei verschiedene Verbesserungen werden als "schneller" gemeldet, und wer sie vermischt, wird enttäuscht. Trennen wir sie.

Die erste ist die **Build-Zeit**, und sie kommt von der parallelen Codegenerierung. Die 64-Bit-Compiler kön-

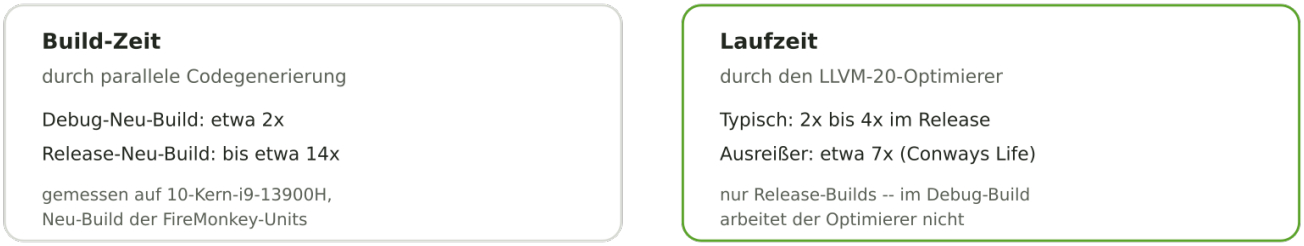
nen jetzt Code für mehrere Delphi-Units gleichzeitig erzeugen, statt eine nach der anderen abzuarbeiten. Embarcaderos Compiler-Beitrag nennt den konkreten Fall: Ein Neu-Build der FireMonkey-Units auf einem

10-Kern-Intel-Core-i9-13900H war für Debug etwa doppelt so schnell und für Release etwa vierzehnmal so schnell.

Diese 14x sind die Zahl, die alle zitieren, deshalb gleich das Sternchen dazu. Es handelt sich um einen Release-Neu-Build einer großen, ungewöhnlich gut parallelisierbaren Codebasis auf einer Maschine mit zehn Kernen. Ein Vier-Kern-Notebook zaubert keine vierzehn Worker herbei. Debug-Builds -- also das, was Sie den ganzen Tag tatsächlich machen -- zeigten im selben Test etwa 2x. Das Doppelte ist wirklich hervorragend. Es ist nur eben nicht vierzehn.

Die zweite Verbesserung ist die **Laufzeitgeschwindigkeit**, und sie kommt vom Optimierer selbst. Hier berichtet

Embarcadero von "runtime improvements in the two-to-four-times range", wobei ein Test mit Conways Spiel des Lebens rund das Siebenfache erreichte. Beachten Sie: Die Spanne von 2 bis 4 ist die ehrliche Schlagzeile, und die 7x sind der Ausreißer, den man immerhin transparent als konkreten Einzeltest benannt hat.



Zwei verschiedene Arten von 'schneller' -- bitte nicht verwechseln

Lesen Sie das Diagramm aus der Sicht Ihres eigenen Arbeitstags. Der linke Kasten ist das, was Sie bei jedem Druck auf F9 spüren. Der rechte Kasten ist das, was Ihr Kunde spürt, und zwar nur in Release-Builds -- ein guter Anlass zu prüfen, ob Ihr Nightly Build und Ihre Deployment-Pipeline wirklich Release-Konfigurationen erzeugen. Ich habe schon mehr als ein Team dabei erwischt, dass dem nicht so war.

Jetzt der Vorbehalt, den niemand auf eine Folie geschrieben hat. Aus demselben Compiler-Beitrag: Mit den neuen Optimierungen kann die *Kompilierung* im Release 34 bis 42 Prozent länger dauern, bevor die parallele

Codegenerierung das wieder ausgleicht. Anders gesagt: Der Optimierer kostet echte Zeit, und die Parallelität ist das, was sie bezahlt. Auf einer Maschine mit wenigen Kernen bekommen Sie am Ende vielleicht schnellere Anwendungen und langsamere Release-Builds. Diesen Tausch würde ich trotzdem eingehen -- Sie sollten nur wissen, dass Sie ihn eingehen.

Die parallele Codegenerierung ist konfigurierbar, und Dalija Prasnikar, die das Buch über Delphis Speicherverwaltung geschrieben hat und Komplimente nicht leichtfertig verteilt, hat die Einstellungen getestet: "You can

select the desired number of worker threads, where 0 leaves that selection to automatic process (I got the best results on the Auto setting), 1 is the same as old serial code generation as it uses only a single worker, and, of course, any other number will create that many workers." Lassen Sie es auf Auto. Setzen Sie es auf 1, wenn Sie das Feature bei der Fehlersuche einmal ausschließen müssen.

> **Die parallele Codegenerierung hat harte Grenzen**
> Sie braucht mehr Speicher als die serielle Kompilierung, und sie wird in den 32-Bit-Versionen der Compiler und in der 32-Bit-IDE nicht unterstützt. Falls Sie aus Gewohnheit immer noch die 32-Bit-IDE starten: hier

> ist ein weiterer Grund, damit aufzuhören. Ich habe dieses Argument ausführlich in Why You Need to Switch to Delphi 13 and its 64-bit IDE Yesterday gemacht, und 13.2 liefert soeben ein zweites.

Der Preis der Eintrittskarte: glibc, Editionen und wo Ihr Code läuft

Gute Nachrichten kommen selten ohne Bedingungen, und hier sind es drei, die Sie vor jeder Upgrade-Planung prüfen müssen.

Die neue Toolchain nimmt **glibc 2.31 als Laufzeit-Basis**. glibc ist die GNU-C-Bibliothek -- die Schicht zwischen

Ihrer Anwendung und dem Linux-Kernel -- und ihre Version ist faktisch eine Untergrenze dafür, auf welchen Distributionen Ihr Binary läuft. Embarcadero benennt die Folge klar: Distributionen, die älter sind als Ubuntu 20.04 aus dem Jahr 2020, fallen heraus. Wenn Sie auf eine langlebige CentOS-7-Maschine in irgendeinem Rechenzentrum deployen, ist das der Satz in den Release-Notes, der Sie am meisten betrifft, und kein Compiler-Schalter wird Sie davon freisprechen.

Die zweite Bedingung: Der Compiler für Linux Intel 64-Bit ist "only in the Enterprise and Architect editions of Delphi and RAD Studio" verfügbar. Das war bei Linux schon immer so, und es gilt weiterhin. Wenn Sie Professional einsetzen, gehört Ihnen das Hauptfeature von 13.2 nicht, und ich sage Ihnen das lieber hier als nach dem Download.

Die dritte ist die erfreuliche. Sie können Anwendungen für Linux Intel 64-Bit jetzt direkt aus der 64-Bit-IDE bauen, neben Win64 und Arm64EC. Der 64-Bit-Compiler räumt außerdem die Adressraumgrenzen ab, die bei sehr großen Projekten gebissen haben. Wer schon einmal zugesehen hat, wie einem 32-Bit-Compiler bei einer großen Unit der Speicher ausgeht, weiß genau, wie willkommen das ist.

> **Der Open-Source-Weg für Object Pascal unter Linux**
>
> Es wäre unredlich, einen Beitrag über Object Pascal unter Linux zu schreiben, ohne die Alternative zu nennen. Free Pascal mit der Lazarus-IDE zielt auf Linux x86-64 -- und auf eine geradezu absurde
>
> Liste weiterer Architekturen --, ist kostenlos, quelloffen und in seinem Object-Pascal-Dialekt bewusst Delphi-kompatibel. Lazarus 4.6.0 erschien im Februar 2026; die aktuelle stabile FPC-Version ist 3.2.2 aus dem Jahr 2021, mit reichlich Entwicklung seither. Das ist eine echte Option, besonders für serverseitigen Code, für Hobbyprojekte und für Teams, die eine Enterprise-Lizenz nicht rechtfertigen können. Was Sie aufgeben, ist die RAD-Studio-Toolchain darum herum: FireMonkey, FireDAC, das Komponenten-Ökosystem und kommerziellen Support. Was Sie gewinnen, sind null Lizenzkosten und ein Compiler, der überall läuft. Wählen Sie nach Ihrer Situation, nicht nach dem Logo.

Die IDE-Arbeit, die Sie am Montagmorgen spüren

Der Compiler ist die Schlagzeile, aber eine IDE-Verbesserung kommt anders an: Sie begegnen ihr hundertmal am Tag.

Die größte strukturelle Änderung ist, dass der **Navigator jetzt in die IDE eingebaut ist**. Früher war er ein

separates Plugin aus GetIt, was bedeutete, dass etwa die Hälfte aller Delphi-Entwickler nie von seiner Existenz erfuhr. Jetzt ist er ab Werk dabei. Er bringt ein Goto-Fenster mit, das Units, Typen, Methoden und andere Symbole durchsucht und beim Tippen filtert, sowie eine Minimap im Editor. Embarcaderos Navigator-Beitrag

beschreibt die Minimap treffend: "The shapes of declarations, methods, blank areas, and comments make it possible to recognize parts of a long unit and move between them quickly." Die integrierte Fassung ergänzt eine Markierung der aktiven Zeile sowie Anzeigen für Haltepunkte und Fehler. Weil sie jetzt Teil der IDE ist und kein Plugin, teilt sie sich den Editor-Zustand, statt ihn zu erraten.

Dazu kommt eine Liste kleiner Korrekturen, die mich mehr freut, als sie eigentlich dürfte. Aus dem IDE-Beitrag:

1. Die **Grenze von 2.000 Zeichen für String-Eigenschaften im Objektinspektor** ist weg, zusammen mit einem mehrzeiligen Editor für Collection-Einträge mit langen Strings. Wer jemals ein SQL-Statement in eine Eigenschaft einfügen wollte, kennt diese Grenze.
2. Die **Klammerpaar-Hervorhebung ignoriert jetzt Klammern in Strings und Kommentaren.** In Object Pascal, wo `{` einen Kommentar öffnet, war das eine echte Quelle der Verwirrung.
3. Rund **zwanzig Fehler beim Einfügen von Code in Dateien mit Linux-Zeilenenden** wurden behoben. Wer ein Repository mit Leuten auf macOS oder Linux teilt -- und wer Git nutzt, tut das vermutlich --, ist darüber gestolpert.
4. Ein neuer Befehl **Reload Open Modules** lädt auf der Festplatte geänderte Dateien neu. Nach einem Branch-Wechsel ist das genau der Befehl, den Sie sich gewünscht haben.
5. Die Formularvererbung unter High DPI wurde korrigiert, und IDE Insight wurde aufgeräumt.

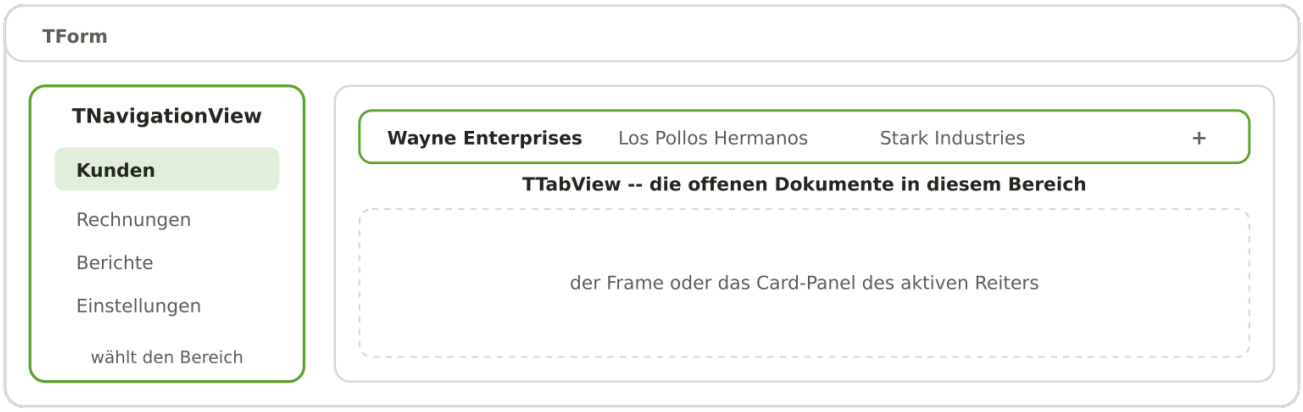
Marco Cantùs Einordnung dieser Liste trifft es: Diese Änderungen "simply let the IDE get out of the way." Kein einzelner Punkt davon würde ein Release verkaufen. Zusammen räumen sie ein Dutzend kleiner Reibungspunkte weg, die Sie nicht mehr wahrgenommen haben, weil Sie gelernt hatten, sie zu umgehen.

Zwei neue VCL-Steuerelemente und eine kleine App als Beweis

Die Konventionen für Windows-Oberflächen sind weitergezogen, und die VCL holt auf. 13.2 ergänzt zwei Steuerelemente -- `TTabView` und `TNavigationView` --, die "patterns used by current Windows applications (and the WinUI3 user interface paradigm)" folgen, wie der Beitrag zu den VCL-Steuerelementen es formuliert, und dabei ganz gewöhnliche VCL-Komponenten bleiben.

`TNavigationView` ist die vertikale Navigationsleiste am linken Fensterrand -- das Element, das zwischen den Hauptbereichen Ihrer Anwendung umschaltet. Es hat einen normalen und einen kompakten Modus, unterstützt Bilder und mehrzeilige Beschriftungen, optionales Umsortieren per Drag-and-drop und Scrollen mit dem Mausrad. `TTabView` ist die horizontale Reiterleiste für Dokumente oder Ansichten innerhalb eines Bereichs, mit Bildern, mehrzeiligem Text, Schließen-Schaltflächen, einer Schaltfläche für neue Reiter, einem Dropdown mit allen Reitern und Scrollen, wenn es eng wird. Beide unterstützen VCL Styles und eigene Zeichenereignisse, und `TTabView` lässt sich mit `TTitleBarPanel` kombinieren, um die Reiter wie im Browser in die Titelleiste zu setzen.

Das Interessante ist, dass sie sich kombinieren lassen. Die Hülle einer modernen Desktop-Anwendung besteht meist aus genau diesen zwei ineinander verschachtelten Dingen.



Die Standardhülle: eine Navigationsleiste wählt den Bereich, Reiter halten die Dokumente darin

Das Muster kennen Sie schon aus Ihrem Browser und aus Visual Studio Code: Die Leiste links sagt, *wo* Sie sind, die Reiter sagen, *was* Sie dort offen haben. Vor 13.2 haben Sie das aus einem `TPanel`, einer `TListBox`, einem `TPageControl` und einer ordentlichen Portion Owner-Draw-Code gebaut. Jetzt setzen Sie zwei Komponenten ab.

Hier ist ein kleines Formular, das beide verdrahtet und aus Code befüllt. Nichts Ausgefallenes, nur die Grundlagen -- das ist das Skelett, keine Anwendung.

```
unit MainFormU;

interface

uses
  Winapi.Windows, System.SysUtils, System.Classes, Vcl.Graphics, Vcl.Controls,
  Vcl.Forms, Vcl.ComCtrls, Vcl.ExtCtrls, Vcl.TabView, Vcl.NavigationView;

type
  TMainForm = class(TForm)
    NavigationView: TNavigationView;
    TabView: TTabView;
    ContentPanel: TPanel;
    procedure FormCreate(Sender: TObject);
    procedure NavigationViewChange(Sender: TObject);
    procedure TabViewChange(Sender: TObject);
  private
    procedure AddArea(const AName: string);
    procedure OpenDocument(const ACaption: string);
    procedure ShowStatus(const AText: string);
  end;

var
  MainForm: TMainForm;

implementation

{$R *.dfm}

procedure TMainForm.FormCreate(Sender: TObject);
begin
  // 1. Die vertikale Leiste: die Hauptbereiche der Anwendung.
  AddArea('Kunden');
  AddArea('Rechnungen');
  AddArea('Berichte');
  AddArea('Einstellungen');
```

```
  // 2. Die horizontalen Reiter: was im gewählten Bereich offen ist.
  OpenDocument('Wayne Enterprises');
  OpenDocument('Los Pollos Hermanos');

  // 3. Mit dem ersten Bereich starten, damit das Formular nie leer erscheint.
  if NavigationView.Items.Count > 0 then
    NavigationView.ItemIndex := 0;
end;

procedure TMainForm.AddArea(const AName: string);
begin
  NavigationView.Items.Add.Caption := AName;
end;

procedure TMainForm.OpenDocument(const ACaption: string);
var
  LItem: TTabViewItem;
begin
  LItem := TabView.Items.Add;
  LItem.Caption := ACaption;
  TabView.ItemIndex := LItem.Index;
end;

procedure TMainForm.NavigationViewChange(Sender: TObject);
begin
  // 4. Den Inhalt für den gewählten Bereich austauschen. In einer echten
  //    Anwendung zeigen Sie hier einen Frame oder ein Card-Panel je Bereich.
  ShowStatus(Format('Bereich: %s', [NavigationView.Items[NavigationView.ItemIndex].Caption]));
end;

procedure TMainForm.TabViewChange(Sender: TObject);
begin
  ShowStatus(Format('Dokument: %s', [TabView.Items[TabView.ItemIndex].Caption]));
end;

procedure TMainForm.ShowStatus(const AText: string);
begin
  Caption := AText;
end;

end.
```

Gehen wir es durch:

1. `AddArea` füllt die Navigationsleiste. Jeder Eintrag ist ein Element einer Items-Collection, genau wie die Collections, die Sie im Objektinspektor bei einer `TListView` oder einer Toolbar ohnehin bearbeiten -- es gibt hier kein neues Konzept zu lernen.
2. `OpenDocument` fügt einen Reiter hinzu und wählt ihn sofort aus, denn das erwarten Anwender, wenn sich etwas öffnet.
3. Das Setzen von `ItemIndex` beim Start ist wichtiger, als es aussieht. Eine Navigationshülle, die ohne Auswahl startet, wirkt kaputt, und Ihre Anwender werden Ihnen das auch sagen.
4. In den Change-Handlern lebt Ihre Anwendung. Hier setzen sie nur die Fensterüberschrift; in einem echten Projekt tauscht der Bereichswechsel den sichtbaren Frame aus, und der Reiterwechsel das Dokument, an das dieser Frame gebunden ist.

> **Prüfen Sie die genauen Unit- und Eigenschaftsnamen an Ihrer Installation**
>
> Der Code oben zeigt die Form der Sache auf Basis der Ankündigung, in dem Stil, dem diese Collection-basierten VCL-Steuerelemente seit jeher folgen. Ich habe ihn gegen das dokumentierte Verhalten geschrieben und nicht gegen eine 13.2-Installation auf meinem eigenen Rechner, und Embarcaderos eigene Release-Seite steckt hinter einer Bot-Sperre, die ich nicht direkt lesen konnte. Bevor Sie ihn in ein Projekt übernehmen, bestätigen Sie die Unit-Namen und die genauen Member-Namen im DocWiki zu

> 13.2 oder schlicht, indem Sie die Komponenten auf ein Formular setzen und den Objektinspektor lesen.
>
> Das Muster wird tragen; ein Eigenschaftsname vielleicht nicht.

Die Änderung, die Ihren Build wirklich anhalten wird

Jetzt zu dem Teil der Release-Notes, den ich ausdrucken und an den Monitor kleben würde -- und er hat nichts mit Linux zu tun.

Der Compiler hat einen Fehler darin behoben, wie er einen generischen Typ `T` behandelt, der auf eine Klasse oder ein Interface eingeschränkt ist. Dalija Prasnikar beschreibt es präzise: "Where the compiler previously allowed automatic conversion from `TObject` or `IInterface` to `T`, you will now have to use variables of type `T` or typecast." Code, der gestern noch kompilierte, begrüßt Sie heute womöglich mit einem `E2010 Incompatible types`.

Sehen Sie sich die Form davon an. So etwas ist früher durchgerutscht:

```
type
  TCache<T: class> = class
  private
    FItems: TList<TObject>;
  public
    function Get(const AIndex: Integer): T;
  end;

function TCache<T>.Get(const AIndex: Integer): T;
begin
  // Kompilierte früher. Jetzt: E2010 Incompatible types.
  Result := FItems[AIndex];
end;
```

Die Korrektur ist mechanisch -- ein Typecast, `Result := T(FItems[AIndex]);` -- und genau deshalb dürfen Sie sie nicht mechanisch vornehmen. Prasnikars Warnung ist die wichtigere Hälfte der Geschichte: "You should carefully inspect the code you are fixing, because that implicit conversion allowed completely incorrect conversions." Das alte Verhalten ließ Sie einen `TCustomer` an etwas übergeben, das einen `TInvoice` erwartete, und Sie erfuhren davon zur Laufzeit, per Zugriffsverletzung an einer Stelle, die mit dem Fehler nichts zu tun hatte. Der Compiler sagt Ihnen jetzt, wo diese Konvertierungen sitzen. Jedes `E2010`, das diese Änderung zutage fördert, ist eine Frage, die eine Antwort verdient, statt zum Schweigen gebracht zu werden.

Meiner Meinung nach ist das die wertvollste Änderung in 13.2, und mir ist bewusst, dass das eine seltsame Aussage über ein Release ist, dessen Schlagzeile eine vierzehnfache Build-Beschleunigung ist. Meine Begründung: Ein schnellerer Build spart Ihnen Minuten, während ein Typ-Loch, das erst zur Laufzeit auffällt, ein Wochenende und das Vertrauen eines Kunden kosten kann. Dennoch möchte ich die Aussage ehrlich eingrenzen. Wenn Ihr Code wenig mit Generics arbeitet oder nur mit den Standard-Collections, treffen Sie das vielleicht überhaupt nicht, und der Compiler ist für Sie zweifellos die bessere Geschichte. Und es gibt ein faires Gegenargument: Eine brechende Änderung in einem Punkt-Release ist für Teams mit großen Altbeständen und engen Terminen ein realer Kostenfaktor, und "es war ein Bugfix" macht eine ungeplante Migration nicht kostenlos. Beides stimmt. Ich halte die Korrektur trotzdem für richtig.

> **Arbeiten Sie sich nicht mit pauschalen Typecasts aus E2010 heraus**
>
> Die Versuchung ist groß, jede Beschwerde in einen harten Cast zu packen, bis der Build grün ist. Damit verwandeln Sie einen Compile-Fehler, den Sie sehen, in einen Laufzeitfehler, den Sie nicht sehen. Wenn ein Cast nicht offensichtlich sicher ist, nehmen Sie eine geprüfte Konvertierung (`as`, oder ein `Supports` bei Interfaces) und lassen Sie sie an der Stelle des Fehlers laut scheitern.

Was sonst noch in der Schachtel liegt

Das Release ist breiter als Linux, und ein paar Punkte verdienen wenigstens eine Erwähnung, damit Sie von ihnen wissen.

Auf der Seite von Windows on Arm unterstützt Arm64EC jetzt dynamische Laufzeitpakete, sodass Anwendungen, die auf Delphis Package-Architektur aufbauen, auf Arm umziehen können, ohne diese Architektur aufzugeben. Der Standard-Speichermanager basiert dort nun auf FastMM4 statt auf dem langsameren Allokator der Plattform, und `rpmalloc` wird als Option für Arm64EC und Linux angeboten -- laut Bericht 2x bis 30x wert bei allokationslastigen, mehrfädigen Arbeitslasten, zum Preis von etwa doppeltem Speicherverbrauch. Das ist ein konkreter Tausch für eine konkrete Art von Anwendung und keine Einstellung, die man aus einer Laune heraus umlegt.

FireDAC ergänzt Unterstützung für PostgreSQL 18.4, konfigurierbares Oracle-CLOB-Prefetching und Informix-Anbindung unter Linux über den Informix-ODBC-Treiber. Auf der Webseite gibt es OAuth 2 Authorization Code Flow mit PKCE, besseres Verhalten bei Socket-Timeouts und Zugriff auf HTTP-Zertifikatsinformationen. Android hebt das voreingestellte Mindest-SDK auf API-Level 24 und frischt die AdMob- und Maps-Bibliotheken auf. Skia4Delphi geht auf 7.3. Der FireMonkey Style Designer lässt sich jetzt aus der IDE starten und kann vorhandene VCL-Style-Dateien importieren.

Es gibt auch eine KI-Geschichte: Das SmartCore AI Components Pack wurde überarbeitet und erreicht OpenAI, Anthropic Claude, Google Gemini und Ollama hinter einer anbieterunabhängigen Architektur, und `MCPConnect` hilft Ihnen beim Bau von Model-Context-Protocol-Servern, damit KI-Agenten Ihre vorhandene Geschäftslogik aufrufen können. Beides kommt über GetIt. Regelmäßige Leser wissen, dass ich zu lokalen Modellen eine Meinung habe -- siehe Give Your Delphi App a Brain --, und ein offiziell unterstützter Weg zu Ollama ist eine erfreuliche Sache in der Schachtel.

Und C++Builder hatte ein eigenes, substanzielles Release: Der moderne Win64x-Compiler läuft jetzt in-process in der IDE mit genauer Fortschrittsanzeige, reaktionsfähigem Abbrechen von Builds und paralleler Kompilierung, besserer Package-Unterstützung samt Design-Time-Packages sowie verbesserter LLDB-Inspektion moderner C++-Datenstrukturen. Das ist echte Arbeit für die Leute, die auf dieser Seite des Produkts leben, und ich freue mich darüber -- es ist nur nicht meine Seite des Hauses.

Fazit

Ein Punkt-Release verdient normalerweise nicht so viel Aufmerksamkeit. Dieses hier schon, und zwar aus einem Grund, der mehr damit zu tun hat, was repariert wurde, als damit, was hinzugekommen ist.

Der Wechsel des Linux-Compilers von LLVM 3.3 auf 20.1 ist kein Feature; es ist das Abschaffen eines dreizehn Jahre alten Handicaps. Delphi auf dem Server kompiliert jetzt durch dieselbe Klasse von Optimierer, die alle anderen seit Jahren nutzen, linkt mit LLD und erzeugt Debug-Informationen, die ein moderner Debugger auch wirklich lesen kann. Die 2x- bis 4x-Zahlen bei der Laufzeit sind das sichtbare Ergebnis davon, und die 14x beim Release-Neu-Build sind das, was passiert, wenn parallele Codegenerierung zehn Kerne bekommt. Prüfen Sie die glibc-2.31-Grenze und Ihre Edition, bevor Sie irgendetwas planen.

> Die Schlagzeile ist ein Compiler. Was Ihren Nachmittag verändert, sind vier Worte in den Generics-Regeln.

Planen Sie einen Nachmittag für `E2010` ein und behandeln Sie jedes einzelne als Frage an Ihren Code statt als Hindernis auf dem Weg zum grünen Build. Und dann genießen Sie die Kleinigkeiten -- einen Objektinspektor, der nicht mehr bei 2.000 Zeichen abschneidet, eine Klammerhervorhebung, die Kommentare versteht, und einen Navigator, von dem man nicht mehr vorher wissen muss, dass es ihn gibt.

In einem künftigen Beitrag würde ich gern etwas Echtes auf `TNavigationView` und `TTabView` bauen statt eines Skeletts -- eine richtige Anwendungshülle mit Frames, Zustand und den unbequemen Stellen, die in Ankündigungs-Screenshots niemand zeigt. Wenn Sie vor mir aktualisieren und auf eine scharfe Kante stoßen, erzählen Sie mir davon.

INDEPENDENT · AD-FREE · READER-SUPPORTED

Free to read. Not free to make.

Every tutorial, article, and line of example code on Holger's Code is published without a paywall and without ads. This page is about what keeps it that way.

Professional work, published free

The content here is held to the same standard as professional client work: every tutorial is built as a real project, tested against current versions, documented step by step, and revisited when the frameworks move on. A single in-depth tutorial routinely takes days — and that's before the bills behind it: servers, storage, build infrastructure, licenses, and the hardware everything is verified on.

If an article or tutorial here has saved you an afternoon of debugging, taught you a technique, or helped you ship, it created real value for you. Supporting it isn't charity — it's paying for professional work you found useful, at whatever price you choose. And if you can't, keep reading anyway. That's what the free part means.

Ways to support

KO-FI

Buy me a coffee

A one-time tip — no account, no subscription, any amount.

BOOKS & COURSES

Buy the paid work

The paid tier of everything published free here — and you get something lasting in return.

SPREAD THE WORD

Share an article

Send a tutorial to a colleague, cite an article, subscribe to the RSS feed.

Nothing here goes behind a paywall, whether you chip in or not. But if something on this site earned its keep today, this is where to say so.

Support on Ko-fi —